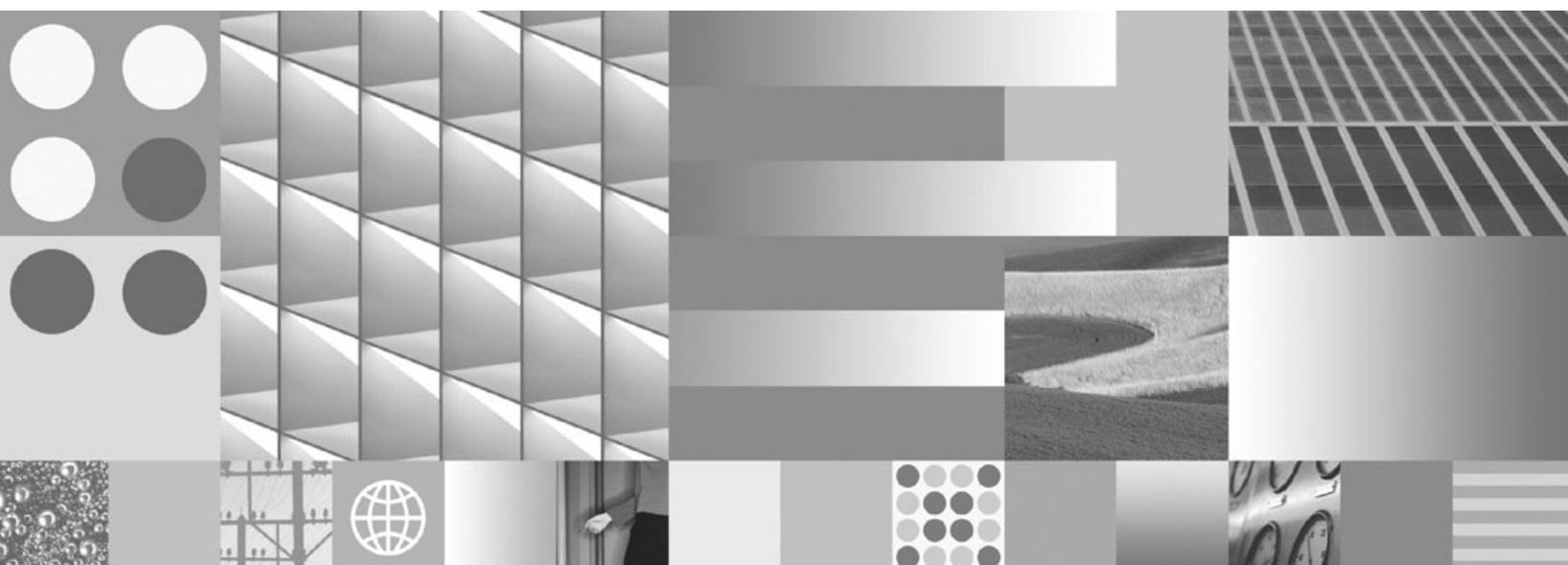
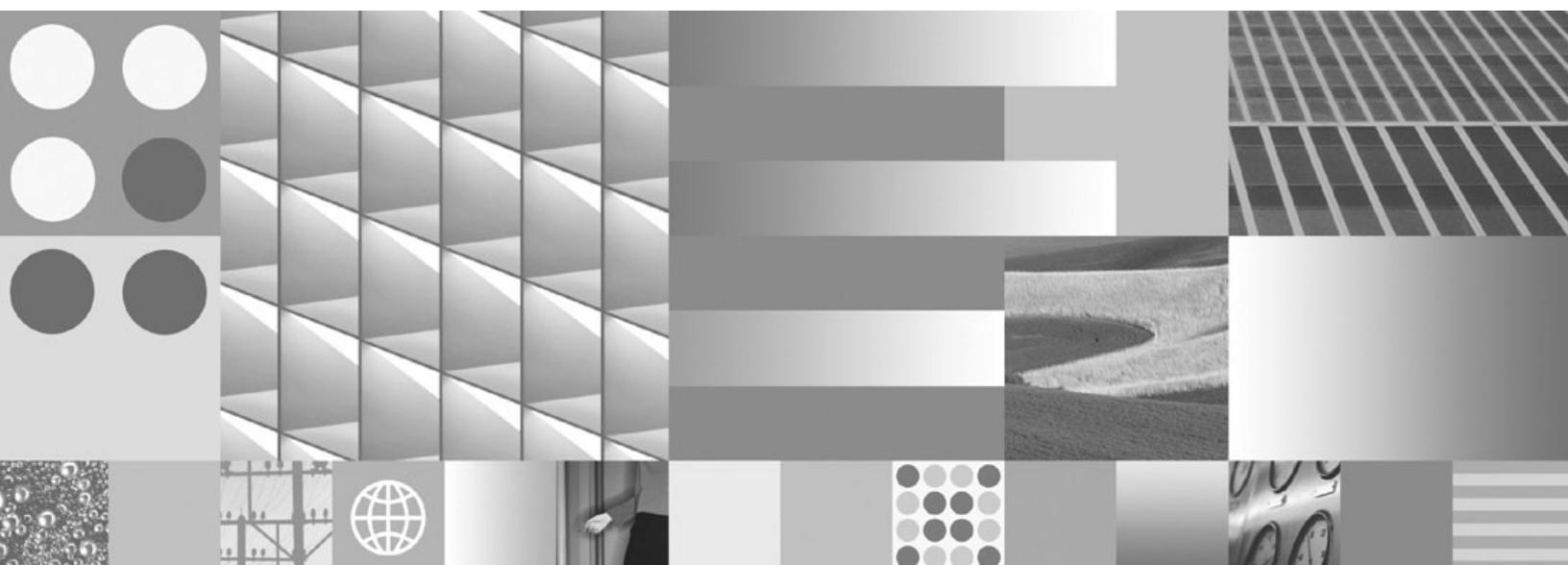


Version 8 Release 1



Parallel Job Advanced Developer Guide

Version 8 Release 1



Parallel Job Advanced Developer Guide

Note

Before using this information and the product that it supports, read the information in "Notices" on page 817.

© Ascential Software Corporation 2001, 2005.

© Copyright International Business Machines Corporation 2006, 2008. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1. Terminology	1
Chapter 2. Job design tips	3
WebSphere DataStage Designer interface	3
Processing large volumes of data	3
Modular development	3
Designing for good performance.	4
Combining data	4
Sorting data	5
Default and explicit type conversions	5
Using Transformer stages	7
Using Sequential File stages	7
Using Database stages	8
Database sparse lookup vs. join	8
DB2 database tips.	8
Oracle database tips	9
Teradata Database Tips	9
Chapter 3. Improving performance	11
Understanding a flow	11
Score dumps	11
Example score dump	11
Tips for debugging	12
Performance monitoring	12
Job monitor	12
Iostat	13
Load average.	13
Runtime information	13
Performance data	14
OS/RDBMS specific tools.	15
Performance analysis	15
Selectively rewriting the flow	16
Identifying superfluous repartitions	16
Identifying buffering issues	16
Resource estimation	17
Creating a model	17
Making a projection	19
Generating a resource estimation report	20
Examples of resource estimation	20
Resolving bottlenecks	23
Choosing the most efficient operators.	23
Partitioner insertion, sort insertion.	24
Combinable Operators.	24
Disk I/O	24
Ensuring data is evenly partitioned	25
Buffering	25
Platform specific tuning	26
HP-UX	26
AIX	26
Disk space requirements of post-release 7.0.1 data sets	26
Chapter 4. Link buffering	27
Buffering assumptions.	27
Controlling buffering	28
Buffering policy	28

Overriding default buffering behavior	28
Operators with special buffering requirements	29
Chapter 5. Specifying your own parallel stages	31
Defining custom stages	31
Defining custom stages	35
Defining build stages	38
Build stage macros	43
Informational macros	43
Flow-control macros	43
Input and output macros	43
Transfer Macros	44
How your code is executed	44
Inputs and outputs	44
Example Build Stage	46
Defining wrapped stages	46
Example wrapped stage	50
Chapter 6. Environment Variables	51
Buffering	55
APT_BUFFER_FREE_RUN	55
APT_BUFFER_MAXIMUM_MEMORY	56
APT_BUFFER_MAXIMUM_TIMEOUT	56
APT_BUFFER_DISK_WRITE_INCREMENT	56
APT_BUFFERING_POLICY	56
APT_SHARED_MEMORY_BUFFERS	56
Building Custom Stages	56
DS_OPERATOR_BUILDDOP_DIR	57
OSH_BUILDDOP_CODE	57
OSH_BUILDDOP_HEADER	57
OSH_BUILDDOP_OBJECT	57
OSH_BUILDDOP_XLC_BIN	57
OSH_CBUILDOP_XLC_BIN	57
Compiler	57
APT_COMPILER	57
APT_COMPILEOPT	58
APT_LINKER	58
APT_LINKOPT	58
DB2 Support	58
APT_DB2INSTANCE_HOME	58
APT_DB2READ_LOCK_TABLE	58
APT_DBNAME	58
APT_RDBMS_COMMIT_ROWS	58
DB2DBDFT	58
Debugging	58
APT_DEBUG_OPERATOR	59
APT_DEBUG_MODULE_NAMES	59
APT_DEBUG_PARTITION	59
APT_DEBUG_SIGNALS	59
APT_DEBUG_STEP	59
APT_DEBUG_SUBPROC	60
APT_EXECUTION_MODE	60
APT_PM_DBX	60
APT_PM_GDB	60
APT_PM_LADEBUG	60
APT_PM_SHOW_PIDS	60
APT_PM_XLDB	61
APT_PM_XTERM	61
APT_SHOW_LIBLOAD	61
Decimal support	61

APT_DECIMAL_INTERM_PRECISION	61
APT_DECIMAL_INTERM_SCALE	61
APT_DECIMAL_INTERM_ROUND_MODE	61
Disk I/O	61
APT_BUFFER_DISK_WRITE_INCREMENT	61
APT_CONSISTENT_BUFFERIO_SIZE	61
APT_EXPORT_FLUSH_COUNT	62
APT_IO_MAP/APT_IO_NOMAP and APT_BUFFERIO_MAP/APT_BUFFERIO_NOMAP	62
APT_PHYSICAL_DATASET_BLOCK_SIZE	62
General Job Administration	62
APT_CHECKPOINT_DIR	62
APT_CLOBBER_OUTPUT	62
APT_CONFIG_FILE	62
APT_DISABLE_COMBINATION	62
APT_EXECUTION_MODE	63
APT_ORCHHOME	63
APT_STARTUP_SCRIPT	63
APT_NO_STARTUP_SCRIPT	63
APT_STARTUP_STATUS	64
APT_THIN_SCORE	64
Job Monitoring	64
APT_MONITOR_SIZE	64
APT_MONITOR_TIME	64
APT_NO_JOBMON	64
APT_PERFORMANCE_DATA	64
Look up support	64
APT_LUTCREATE_MMAP	64
APT_LUTCREATE_NO_MMAP	65
Miscellaneous	65
APT_COPY_TRANSFORM_OPERATOR	65
APT_DATE_CENTURY_BREAK_YEAR	65
APT_EBCDIC_VERSION	65
APT_IMPEXP_ALLOW_ZERO_LENGTH_FIXED_NULL	65
APT_IMPORT_REJECT_STRING_FIELD_OVERRUNS	65
APT_INSERT_COPY_BEFORE MODIFY	66
APT_OLD_BOUNDED_LENGTH	66
APT_OPERATOR_REGISTRY_PATH	66
APT_PM_NO_SHARED_MEMORY	66
APT_PM_NO_NAMED_PIPES	66
APT_PM_SOFT_KILL_WAIT	66
APT_PM_STARTUP_CONCURRENCY	66
APT_RECORD_COUNTS	66
APT_SAVE_SCORE	67
APT_SHOW_COMPONENT_CALLS	67
APT_STACK_TRACE	67
APT_WRITE_DS_VERSION	67
OSH_PRELOAD_LIBS	67
Network	67
APT_IO_MAXIMUM_OUTSTANDING	68
APT_IOMGR_CONNECT_ATTEMPTS	68
APT_PM_CONDUCTOR_HOSTNAME	68
APT_PM_NO_TCPIP	68
APT_PM_NODE_TIMEOUT	68
APT_PM_SHOWRSH	68
APT_PM_STARTUP_PORT	68
APT_PM_USE_RSH_LOCALLY	68
NLS Support	68
APT_COLLATION_SEQUENCE	69
APT_COLLATION_STRENGTH	69
APT_ENGLISH_MESSAGES	69
APT_IMPEXP_CHARSET	69

APT_INPUT_CHARSET	69
APT_OS_CHARSET	69
APT_OUTPUT_CHARSET	69
APT_STRING_CHARSET	69
Oracle Support	70
APT_ORACLE_LOAD_DELIMITED	70
APT_ORACLE_LOAD_OPTIONS	70
APT_ORACLE_NO_OPS	70
APT_ORACLE_PRESERVE_BLANKS	70
APT_ORA_IGNORE_CONFIG_FILE_PARALLELISM	70
APT_ORA_WRITE_FILES	70
APT_ORAUPINSERT_COMMIT_ROW_INTERVAL APT_ORAUPINSERT_COMMIT_TIME_INTERVAL	71
Partitioning	71
APT_NO_PART_INSERTION	71
APT_PARTITION_COUNT	71
APT_PARTITION_NUMBER	71
Reading and writing files	71
APT_DELIMITED_READ_SIZE	71
APT_FILE_IMPORT_BUFFER_SIZE	72
APT_FILE_EXPORT_BUFFER_SIZE	72
APT_IMPORT_PATTERNUSESFILESET	72
APT_MAX_DELIMITED_READ_SIZE	72
APT_PREVIOUS_FINAL_DELIMITER_COMPATIBLE	72
APT_STRING_PADCHAR	72
Reporting	72
APT_DUMP_SCORE	72
APT_ERROR_CONFIGURATION	72
APT_MSG_FILELINE	74
APT_PM_PLAYER_MEMORY	74
APT_PM_PLAYER_TIMING	74
APT_RECORD_COUNTS	74
OSH_DUMP	75
OSH_ECHO	75
OSH_EXPLAIN	75
OSH_PRINT_SCHEMAS	75
SAS Support	75
APT_HASH_TO_SASHASH	75
APT_NO_SASOUT_INSERT	75
APT_NO_SAS_TRANSFORMS	75
APT_SAS_ACCEPT_ERROR	75
APT_SAS_CHARSET	76
APT_SAS_CHARSET_ABORT	76
APT_SAS_COMMAND	76
APT_SASINT_COMMAND	76
APT_SAS_DEBUG	76
APT_SAS_DEBUG_IO	76
APT_SAS_DEBUG_LEVEL	76
APT_SAS_DEBUG_VERBOSE	76
APT_SAS_NO_PSDS_USTRING	77
APT_SAS_S_ARGUMENT	77
APT_SAS_SCHEMASCOURCE_DUMP	77
APT_SAS_SHOW_INFO	77
APT_SAS_TRUNCATION	77
Sorting	77
APT_NO_SORT_INSERTION	77
APT_SORT_INSERTION_CHECK_ONLY	77
Sybase support	78
APT_SYBASE_NULL_AS_EMPTY	78
APT_SYBASE_PRESERVE_BLANKS	78
Teradata Support	78
APT_TERA_64K_BUFFERS	78

APT_TERA_NO_ERR_CLEANUP	78
APT_TERA_NO_SQL_CONVERSION	78
APT_TERA_NO_PERM_CHECKS	78
APT_TERA_SYNC_DATABASE	78
APT_TERA_SYNC_PASSWORD	78
APT_TERA_SYNC_USER	79
Transport Blocks	79
APT_AUTO_TRANSPORT_BLOCK_SIZE	79
APT_LATENCY_COEFFICIENT	79
APT_DEFAULT_TRANSPORT_BLOCK_SIZE	79
APT_MAX_TRANSPORT_BLOCK_SIZE/ APT_MIN_TRANSPORT_BLOCK_SIZE	79
Guide to setting environment variables	79
Environment variable settings for all jobs	80
Optional environment variable settings	80
Chapter 7. Operators.	81
Stage to Operator Mapping	81
Changeapply operator	84
Data flow diagram	85
changeapply: properties	85
Schemas	86
Changeapply: syntax and options	87
Example	90
Changecapture operator	92
Data flow diagram	92
Key and value fields	92
Changecapture: syntax and options	93
Changecapture example 1: all output results	96
Example 2: dropping output results	97
Checksum operator	98
Data flow diagram	98
Properties	98
Checksum: syntax and options	99
Checksum: example	99
Compare operator	100
Data flow diagram	100
compare: properties	100
Compare: syntax and options	101
Compare example 1: running the compare operator in parallel	103
Example 2: running the compare operator sequentially	103
Copy operator	104
Data flow diagram	104
Copy: properties	104
Copy: syntax and options	105
Preventing WebSphere DataStage from removing a copy operator	105
Copy example 1: The copy operator	106
Example 2: running the copy operator sequentially	107
Diff operator	107
Data flow diagram	108
diff: properties	108
Transfer behavior	108
Diff: syntax and options	110
Diff example 1: general example	113
Example 2: Dropping Output Results	113
Encode operator	115
Data flow diagram	115
encode: properties	115
Encode: syntax and options	115
Encoding WebSphere DataStage data sets	116
Example	117
Filter operator	117

Data flow diagram	117
filter: properties	117
Filter: syntax and options	118
Job monitoring information	119
Expressions	120
Input data types	121
Filter example 1: comparing two fields	122
Example 2: testing for a null	123
Example 3: evaluating input records	123
Job scenario: mailing list for a wine auction	123
Funnel operators	125
Data flow diagram	125
sortfunnel: properties	125
Funnel operator	126
Sort funnel operators	126
Generator operator	129
Data flow diagram	130
generator: properties	130
Generator: syntax and options	130
Using the generator operator	131
Example 1: using the generator operator	132
Example 2: executing the operator in parallel	133
Example 3: using generator with an input data set	133
Defining the schema for the operator	134
Timestamp fields	139
Head operator	140
Data flow diagram	141
head: properties	141
Head: syntax and options	141
Head example 1: head operator default behavior	142
Example 2: extracting records from a large data set	143
Example 3: locating a single record	143
Lookup operator	143
Data flow diagrams	144
lookup: properties	144
Lookup: syntax and options	145
Partitioning	149
Create-only mode	149
Lookup example 1: single lookup table record	150
Example 2: multiple lookup table record	150
Example 3: interest rate lookup example	151
Example 4: handling duplicate fields example	152
Merge operator	153
Data flow diagram	154
merge: properties	154
Merge: syntax and options	154
Merging records	156
Understanding the merge operator	159
Example 1: updating national data with state data	162
Example 2: handling duplicate fields	164
Job scenario: galactic industries	166
Missing records	168
Modify operator	170
Data flow diagram	170
modify: properties	171
Modify: syntax and options	171
Transfer behavior	172
Avoiding contiguous modify operators	172
Performing conversions	172
Allowed conversions	203
pcompress operator	205

Data flow diagram	205
pccompress: properties	205
Pcompress: syntax and options	205
Compressed data sets	206
Example	207
Peek operator	208
Data flow diagram	208
peek: properties	208
Peek: syntax and options	209
Using the operator	210
PFTP operator	211
Data flow diagram	212
Operator properties	212
Pftp: syntax and options.	213
Restartability	218
pivot operator	219
Properties: pivot operator	219
Pivot: syntax and options	219
Pivot: examples	220
Remdup operator	221
Data flow diagram	221
remdup: properties	222
Remdup: syntax and options	222
Removing duplicate records	223
Using options to the operator	224
Using the operator	226
Example 1: using remdup	226
Example 2: using the -last option	226
Example 3: case-insensitive string matching	226
Example 4: using remdup with two keys	226
Sample operator	227
Data flow diagram	227
sample: properties.	227
Sample: syntax and options	228
Example sampling of a data set	229
Sequence operator.	229
Data flow diagram	230
sequence: properties	230
Sequence: syntax and options	230
Example of Using the sequence Operator	230
Switch operator	231
Data flow diagram	231
switch: properties	232
Switch: syntax and options	233
Job monitoring information.	237
Example metadata and summary messages	237
Customizing job monitor messages	237
Tail operator.	238
Data flow diagram	238
tail: properties	238
Tail: syntax and options	238
Tail example 1: tail operator default behavior.	239
Example 2: tail operator with both options	239
Transform operator	240
Running your job on a non-NFS MPP	240
Data flow diagram	240
transform: properties	240
Transform: syntax and options.	241
Transfer behavior	250
The transformation language	251
The transformation language versus C	287

Using the transform operator	288
Example 1: student-score distribution	288
Example 2: student-score distribution with a letter grade added to example	291
Example 3: student-score distribution with a class field added to example	294
Example 4. student record distribution with null score values and a reject	297
Example 5. student record distribution with null score values handled	299
Example 6. student record distribution with vector manipulation	302
Example 7: student record distribution using sub-record	306
Example 8: external C function calls	309
Writerangemap operator.	311
Data flow diagram	311
writerangemap: properties	311
Writerangemap: syntax and options	312
Using the writerange operator	313
Chapter 8. The import/export library	315
Record schemas	316
Import example 1: import schema	316
Example 2: export schema	316
Field and record properties	317
Complete and partial schemas.	318
Implicit import and export	322
Error handling during import/export	325
ASCII and EBCDIC conversion tables	326
Import operator	332
Data flow diagram	332
import: properties	332
Import: syntax and options.	333
How to import data	340
Example 1: importing from a single data file	342
Example 2: importing from multiple data files	343
Export operator	344
Data flow diagram	344
export: properties	345
Export: syntax and options	346
How to export data	350
Export example 1: data set export to a single file	354
Example 2: Data Set Export to Multiple files	355
Import/export properties	356
Setting properties	356
Properties	357
Properties: reference listing	364
Chapter 9. The partitioning library	411
The entire partitioner	411
Using the partitioner	412
Data flow diagram	412
entire: properties	412
Syntax.	413
The hash partitioner	413
Specifying hash keys	414
Example	414
Using the partitioner	415
Data flow diagram	415
hash: properties	415
Hash: syntax and options	416
The modulus partitioner.	417
Data flow diagram	417
modulus: properties	418
Modulus: syntax and options	418

Example	418
The random partitioner	419
Using the partitioner	420
Data flow diagram	420
random: properties	420
Syntax.	421
The range Partitioner	421
Considerations when using range partitioning	422
The range partitioning algorithm	422
Specifying partitioning keys	422
Creating a range map	423
Example: configuring and using range partitioner	425
Using the partitioner	425
Data flow diagram	426
range: properties	426
Range: syntax and options	426
Writerangemap operator.	428
Data flow diagram	428
writerangemap: properties	429
Writerangemap: syntax and options	429
Using the writerange operator.	430
The makerangemap utility	431
Makerangemap: syntax and options	431
Using the makerangemap utility	433
The roundrobin partitioner	433
Using the partitioner	434
Data flow diagram	434
roundrobin: properties	434
Syntax.	434
The same partitioner	435
Using the partitioner	435
Data flow diagram	435
same: properties	436
Syntax.	436
Chapter 10. The collection library.	437
The ordered collector.	437
Ordered collecting.	437
ordered Collector: properties	438
Syntax.	438
The roundrobin collector	438
Round robin collecting	439
roundrobin collector: properties	439
Syntax.	440
The sortmerge collector	440
Understanding the sortmerge collector	440
Data flow diagram	440
Specifying collecting keys	441
sortmerge: properties	443
Sortmerge: syntax and options.	443
Chapter 11. The restructure library	445
The aggtorec operator	445
Output formats.	445
aggtorec: properties	446
Aggtorec: syntax and options	446
Aggtorec example 1: the aggtorec operator without the toplevelkeys option	448
Example 2: the aggtorec operator with multiple key options.	448
Example 3: The aggtorec operator with the toplevelkeys option	449
The field_export operator	450

Data flow diagram	450
field_export: properties	451
Field_export: syntax and options	451
Example	452
The field_import operator	453
Data flow diagram	453
field_import: properties	454
Field_import: syntax and options	454
Example	455
The makesubrec operator	457
Data flow diagram	457
makesubrec: properties	458
Transfer behavior	458
Subrecord length	458
Makesubrec: syntax and options	459
The makevect operator	460
Data flow diagram	460
makevect: properties	460
Transfer Behavior	461
Non-consecutive fields	461
Makevect: syntax and options	461
Makevect example 1: The makevect operator	462
Example 2: The makevect operator with missing input fields	462
The promotesubrec Operator	463
Data Flow Diagram	463
promotesubrec: properties	464
Promotesubrec: syntax and options	464
Example	464
The splitsubrec Operator	465
Data Flow Diagram	465
splitsubrec properties	465
Splitsubrec: syntax and options	466
Example	466
The splitvect operator	467
Data flow diagram	467
splitvect: properties	468
Splitvect: syntax and options	468
Example	468
The tagbatch operator	469
Tagged fields and operator limitations	469
Operator action and transfer behavior	470
Data flow diagram	470
tagbatch: properties	471
Added, missing, and duplicate fields	471
Input data set requirements	472
Tagbatch: syntax and options	472
Tagbatch example 1: simple flattening of tag cases	474
Example 2: The tagbatch operator, missing and duplicate cases	475
Example 3: The tagbatch operator with multiple keys	476
The tagswitch operator	477
Data flow diagram	477
tagswitch: properties	478
Input and output interface schemas	478
The case option	478
Using the operator	478
Tagswitch: syntax and options	479
Tagswitch example 1: default behavior	480
Example 2: the tagswitch operator, one case chosen	481
Chapter 12. The sorting library	485
The tsort operator	485

Configuring the tsort operator	487
Using a sorted data set	487
Specifying sorting keys	488
Data flow diagram	489
tsort: properties	489
Tsort: syntax and options	490
Example: using a sequential tsort operator	493
Example: using a parallel tsort operator	494
Performing a total sort	495
Example: performing a total sort	497
The psort operator	499
Performing a partition sort	499
Configuring the partition sort operator	501
Using a sorted data set	501
Data Flow Diagram	503
psort: properties	503
Psort: syntax and options	504
Example: using a sequential partition sort operator	506
Example: using a parallel partition sort operator	507
Performing a total sort	508
Range partitioning	510
Example: Performing a Total Sort	511
Chapter 13. The join library	515
Data flow diagrams	515
Join: properties	516
Transfer behavior	517
Input data set requirements	517
Memory use	517
Job monitor reporting	517
Comparison with other operators	517
Input data used in the examples	518
innerjoin operator	518
Innerjoin: syntax and options	519
Example	519
leftouterjoin operator	520
Leftouterjoin: syntax and options	520
Example	521
rightouterjoin operator	522
Rightouterjoin: syntax and options	522
Example	523
fullouterjoin operator	524
Fullouterjoin: syntax and options	524
Example	525
Chapter 14. The ODBC interface library	527
Accessing ODBC from WebSphere DataStage	527
National Language Support	527
ICU character set options	528
Mapping between ODBC and ICU character sets	528
The odbcread operator	528
Data flow diagram	529
odbcread: properties	529
Odbclookup: syntax and options	529
Operator action	531
Column name conversion	531
Data type conversion	532
External data source record size	532
Reading external data source tables	532
Join operations	533

Odbcread example 1: reading an external data source table and modifying a field name	533
The odbcwrite operator	534
Writing to a multibyte database	535
Data flow diagram	535
odbcwrite: properties	535
Operator action	535
Where the odbcwrite operator runs	536
Odbcwrite: syntax and options	538
Example 1: writing to an existing external data source table	540
Example 2: creating an external datasource table	541
Example 3: writing to an external data source table using the modify operator	542
Other features	543
The odbcupsert operator	543
Data flow diagram	543
odbcupsert: properties	544
Operator action	544
Odbcupsert: syntax and options	545
Example	546
The odbclookup operator	547
Data flow diagram	548
odbclookup: properties	549
Odbclookup: syntax and options	549
Example	551
Chapter 15. The SAS interface library	553
Using WebSphere DataStage to run SAS code	553
Writing SAS programs	553
Using SAS on sequential and parallel systems	553
Pipeline parallelism and SAS	555
Configuring your system to use the SAS interface operators	555
An example data flow	556
Representing SAS and non-SAS Data in DataStage	557
Getting input from a SAS data set	558
Getting input from a WebSphere DataStage data set or a SAS data set	559
Converting between data set types	560
Converting SAS data to WebSphere DataStage data.	561
a WebSphere DataStage example	562
Parallelizing SAS steps	563
Executing PROC steps in parallel.	569
Some points to consider in parallelizing SAS code	573
Using SAS with European languages	574
Using SAS to do ETL	575
The SAS interface operators	576
Specifying a character set and SAS mode	576
Parallel SAS data sets and SAS International	578
Specifying an output schema	579
Controlling ustring truncation	579
Generating a proc contents report	580
WebSphere DataStage-inserted partition and sort components	580
Long name support	580
Environment variables	581
The sasin operator	582
Data flow diagram	583
sasin: properties	583
Sasin: syntax and options	583
The sas operator	587
Data flow diagram	587
sas: properties	587
SAS: syntax and options.	588
The sasout operator	592
Data flow diagram	593

sasout: properties	593
Sasout: syntax and options	593
The sascontents operator	595
Data flow diagram	595
sascontents: properties	596
sascontents: syntax and options	596
Example reports	597
Chapter 16. The Oracle interface library	599
Accessing Oracle from WebSphere DataStage	599
Changing library paths	599
Preserving blanks in fields	599
Handling # and \$ characters in Oracle column names	599
National Language Support	600
ICU character set options	600
Mapping between ICU and Oracle character sets	600
The oraread operator	601
Data flow diagram	601
oraread: properties	601
Operator action.	602
Where the oraread operator runs	602
Column name conversion	603
Data type conversion.	603
Oracle record size	604
Targeting the read operation	604
Join operations	605
Oraread: syntax and options	605
Oraread example 1: reading an Oracle table and modifying a field name	608
Example 2: reading from an Oracle table in parallel with the query option	608
The orawrite operator	609
Writing to a multibyte database	609
Data flow diagram	609
orawrite: properties	610
Operator action.	610
Data type conversion	611
Write modes.	612
Matched and unmatched fields	613
Orawrite: syntax and options	613
Example 1: writing to an existing Oracle table	620
Example 2: creating an Oracle table	621
Example 3: writing to an Oracle table using the modify operator	621
The oraupsert operator	622
Data flow diagram	623
oraupsert: properties	623
Operator Action	623
Associated environment variables	624
Oraupsert: syntax and options.	625
Example	627
The oralookup operator	627
Data flow diagram	629
Properties	629
Oralookup: syntax and options	629
Example	631
Chapter 17. The DB2 interface library	633
Configuring WebSphere DataStage access	633
Establishing a remote connection to a DB2 server	634
Handling # and \$ characters in DB2 column names.	634
Using the -padchar option	635
Running multiple DB2 interface operators in a single step	635

National Language Support	636
Specifying character settings	636
Preventing character-set conversion	636
The db2read operator	637
Data flow diagram	637
db2read: properties	637
Operator action.	637
Conversion of a DB2 result set to a WebSphere DataStage data set	638
Targeting the read operation	639
Specifying open and close commands	640
Db2read: syntax and options	641
Db2read example 1: reading a DB2 table with the table option	643
Example 2: reading a DB2 table sequentially with the -query option	644
Example 3: reading a table in parallel with the -query option	644
The db2write and db2load operators	645
Data flow diagram	645
db2write and db2load: properties.	645
Actions of the write operators	646
How WebSphere DataStage writes the table: the default SQL INSERT statement	646
Field conventions in write operations to DB2.	647
Data type conversion.	647
Write modes.	648
Matched and unmatched fields	649
Db2write and db2load: syntax and options	649
db2load special characteristics.	656
Db2write example 1: Appending Data to an Existing DB2 Table	657
Example 2: writing data to a DB2 table in truncate mode.	658
Example 3: handling unmatched WebSphere DataStage fields in a DB2 write operation.	659
Example 4: writing to a DB2 table containing an unmatched column	660
The db2upsert operator	661
Partitioning for db2upsert	661
Data flow diagram	661
db2upsert: properties.	661
Operator action.	662
Db2upsert: syntax and options	663
The db2part operator.	665
Db2upsert: syntax and options	666
Example	667
The db2lookup operator.	668
Data flow diagram	669
db2lookup: properties	669
Db2lookup: syntax and options	669
Example	671
Considerations for reading and writing DB2 tables	672
Data translation anomalies	672
Using a node map.	672
Chapter 18. The Informix interface library	675
Configuring the INFORMIX user environment	675
Read operators for Informix	675
Data flow diagram	676
Read operator action	676
Execution mode	677
Column name conversion	677
Data type conversion.	677
Informix example 1: Reading all data from an Informix table	678
Write operators for Informix	679
Data flow diagram	680
Operator action.	680
Execution mode	680
Column name conversion	680

Data type conversion	681
Write modes	681
Matching WebSphere DataStage fields with columns of Informix table	682
Limitations	682
Example 2: Appending data to an existing Informix table.	683
Example 3: writing data to an INFORMIX table in truncate mode	684
Example 4: Handling unmatched WebSphere DataStage fields in an Informix write operation	685
Example 5: Writing to an INFORMIX table with an unmatched column	686
hpread operator	687
Special operator features	687
Establishing a remote connection to the hpread operator.	687
Data flow diagram	688
Properties of the hpread operator	688
Hpread: syntax and options	689
Example	690
hplwrite operator for Informix.	690
Special operator features	690
Data flow diagram	690
Properties of the hplwrite operator	691
hplwrite: syntax and options	691
Examples.	692
infread operator	693
Data Flow Diagram	693
infread: properties	693
Infread: syntax and Options	694
Example	695
infwrite operator	695
Data flow diagram	696
Properties of the infwrite operator	696
infwrite: syntax and options	696
Examples.	698
xpread operator	698
Data flow diagram	698
Properties of the xpread operator	698
Xpread: syntax and options	699
Example	700
xpswrite operator	700
Data flow diagram	701
Properties of the xpswrite operator	701
Xpswrite: syntax and options	701
Examples.	703
Chapter 19. The Teradata interface library	705
National language support	705
Teradata database character sets	705
Japanese language support	705
Specifying a WebSphere DataStage ustring character set	706
Teraread operator	706
Data flow diagram	707
teraread: properties	707
Specifying the query	707
Column name and data type conversion	708
teraread restrictions	709
Teraread: syntax and Options	709
Terawrite Operator	711
Data flow diagram	712
terawrite: properties	712
Column Name and Data Type Conversion.	712
Correcting load errors	713
Write modes.	714
Writing fields	714

Limitations	714
Restrictions	715
Terawrite: syntax and options	715
Chapter 20. The Sybase interface library.	719
Accessing Sybase from WebSphere DataStage	719
Sybase client configuration	719
National Language Support	720
The asesybasereade and sybasereade Operators	720
Data flow diagram	720
asesybasereade and sybaseread: properties	720
Operator Action	721
Where asesybasereade and sybasereade Run	721
Column name conversion	721
Data type conversion	721
Targeting the read operation	722
Join Operations	723
Asesybasereade and sybasereade: syntax and Options	723
Sybasereade example 1: Reading a Sybase Table and Modifying a Field Name.	725
The asesybasewrite and sybasewrite Operators	726
Writing to a Multibyte Database	726
Data flow diagram	726
asesybasewrite and sybasewrite: properties	727
Operator Action	727
Where asesybasewrite and sybasewrite Run	727
Data conventions on write operations to Sybase	728
Data type conversion	728
Write Modes	729
Matched and unmatched fields	729
asesybasewrite and sybasewrite: syntax and Options	730
Example 1: Writing to an Existing Sybase Table	733
Example 2: Creating a Sybase Table	734
Example 3: Writing to a Sybase Table Using the modify Operator	735
The asesybaseupsert and sybaseupsert Operators	736
Data flow diagram	737
asesybaseupsert and sybaseupsert: properties.	737
Operator Action	737
Asesybaseupsert and sybaseupsert: syntax and Options	738
Example	740
The asesybaselookup and sybaselookup Operators	741
Data flow diagram	742
asesybaselookup and sybaselookup: properties	742
asesybaselookup and sybaselookup: syntax and Options	742
Example	747
Chapter 21. The SQL Server interface library.	749
Accessing SQL Server from WebSphere DataStage	749
UNIX	749
Windows.	749
National Language Support	749
The sqlsvrread operator	750
Data flow diagram	750
sqlsvrread: properties	750
Operator action.	750
Where the sqlsvrread operator runs	751
Column name conversion	751
Data type conversion.	751
SQL Server record size	752
Targeting the read operation	752
Join operations	753

Sqlsvrread: syntax and options	753
Sqlsvrread example 1: Reading a SQL Server table and modifying a field name	755
The sqlsrvrwrite operator	756
Writing to a multibyte database	756
Data flow diagram	756
sqlsrvrwrite: properties	756
Operator action	757
Where the sqlsrvrwrite operator runs	757
Data conventions on write operations to SQL Server	757
Write modes	758
Sqlsvrwrite: syntax and options	759
Example 1: Writing to an existing SQL Server table	761
Example 2: Creating a SQL Server table	762
Example 3: Writing to a SQL Server table using the modify operator	763
The sqlsrvrupsert operator	764
Data flow diagram	764
sqlsrvrupsert: properties	764
Operator action	764
Sqlsrvrupsert: syntax and options	765
Example	767
The sqlsrvrlookup operator	768
Data flow diagram	769
sqlsrvrlookup: properties	769
Sqlsrvrlookup: syntax and options	770
Example	772
Chapter 22. The iWay interface library	773
Accessing iWay from WebSphere DataStage	773
National Language Support	773
The iwayread operator	773
Data flow diagram	774
iwayread: properties	774
Operator action	774
Data type conversion	774
Iwayread: syntax and options	775
Example: Reading a table via iWay	777
The iwaylookup operator	777
Data flow diagram	779
iwaylookup: properties	779
Iwaylookup: syntax and options	779
Example: looking up a table via iWay	782
Chapter 23. The Netezza Interface Library	783
Netezza write operator	783
Netezza data load methods	783
nzload method	783
External table method	783
Write modes	783
Limitations of write operation	784
Character set limitations	784
Bad input records	784
Error logs	784
Syntax for nzwrite operation	785
Chapter 24. The Classic Federation interface library	787
Accessing the federated database from WebSphere DataStage	787
National language support	787
International components for unicode character set parameter	788
Mapping between federated and ICU character sets	788
Read operations with classicfedread	788

classicfedread: properties	789
Classicfedread: syntax and options	789
Column name conversion	791
Data type conversion	791
Reading external data source tables	792
Write operations with classicfedwrite	792
Matched and unmatched fields	793
Classicfedwrite: syntax and options	793
Writing to multibyte databases	797
Insert and update operations with classicfedupsert	797
classicfedupsert: Properties	798
Classicfedupsert: syntax and options	798
Example of a federated table when a classicfedupsert operation is performed	799
Lookup Operations with classicfedlookup	800
classicfedlookup: properties	801
classicfedlookup: syntax and options	802
Example of a classicfedlookup operation	803
Chapter 25. Header files	805
C++ classes - sorted by header file	805
C++ macros - sorted by header file	809
Product documentation	811
Contacting IBM.	811
How to read syntax diagrams	813
Product accessibility	815
Notices	817
Trademarks	819
Index	821

Chapter 1. Terminology

Because of the technical nature of some of the descriptions in this manual, it sometimes talks about details of the engine that drives parallel jobs. This involves the use of terms that might be unfamiliar to ordinary parallel job users.

- **Operators.** These underlie the stages in an IBM® WebSphere® DataStage™ job. A single stage might correspond to a single operator, or a number of operators, depending on the properties you have set, and whether you have chosen to partition or collect or sort data on the input link to a stage. At compilation, WebSphere DataStage evaluates your job design and will sometimes optimize operators out if they are judged to be superfluous, or insert other operators if they are needed for the logic of the job.
- **OSH.** This is the scripting language used internally by the WebSphere DataStage parallel engine.
- **Players.** Players are the workhorse processes in a parallel job. There is generally a player for each operator on each node. Players are the children of section leaders; there is one section leader per processing node. Section leaders are started by the conductor process running on the conductor node (the conductor node is defined in the configuration file).

Chapter 2. Job design tips

These topics give some hints and tips for the good design of parallel jobs.

WebSphere DataStage Designer interface

The following are some tips for smooth use of the WebSphere DataStage Designer when actually laying out your job on the canvas.

- To re-arrange an existing job design, or insert new stage types into an existing job flow, first disconnect the links from the stage to be changed, then the links will retain any meta data associated with them.
- A Lookup stage can only have one input stream, one output stream, and, optionally, one reject stream. Depending on the type of lookup, it can have several reference links. To change the use of particular Lookup links in an existing job flow, disconnect the links from the Lookup stage and then right-click to change the link type, for example, Stream to Reference.
- The Copy stage is a good placeholder between stages if you anticipate that new stages or logic will be needed in the future without damaging existing properties and derivations. When inserting a new stage, simply drag the input and output links from the Copy placeholder to the new stage. Unless the Force property is set in the Copy stage, WebSphere DataStage optimizes the actual copy out at runtime.

Processing large volumes of data

The ability to process large volumes of data in a short period of time depends on all aspects of the flow and the environment being optimized for maximum throughput and performance. Performance tuning and optimization are iterative processes that begin with job design and unit tests, proceed through integration and volume testing, and continue throughout the production life cycle of the application. Here are some performance pointers:

- When writing intermediate results that will only be shared between parallel jobs, always write to persistent data sets (using Data Set stages). You should ensure that the data is partitioned, and that the partitions, and sort order, are retained at every stage. Avoid format conversion or serial I/O.
- Data Set stages should be used to create restart points in the event that a job or sequence needs to be rerun. But, because data sets are platform and configuration specific, they should not be used for long-term backup and recovery of source data.
- Depending on available system resources, it might be possible to optimize overall processing time at run time by allowing smaller jobs to run concurrently. However, care must be taken to plan for scenarios when source files arrive later than expected, or need to be reprocessed in the event of a failure.
- Parallel configuration files allow the degree of parallelism and resources used by parallel jobs to be set dynamically at runtime. Multiple configuration files should be used to optimize overall throughput and to match job characteristics to available hardware resources in development, test, and production modes.

The proper configuration of scratch and resource disks and the underlying filesystem and physical hardware architecture can significantly affect overall job performance.

Within clustered ETL and database environments, resource-pool naming can be used to limit processing to specific nodes, including database nodes when appropriate.

Modular development

You should aim to use modular development techniques in your job designs in order to maximize the reuse of parallel jobs and components and save yourself time.

- Use job parameters in your design and supply values at run time. This allows a single job design to process different data in different circumstances, rather than producing multiple copies of the same job with slightly different arguments.
- Using job parameters allows you to exploit the WebSphere DataStage Director's multiple invocation capability. You can run several invocations of a job at the same time with different runtime arguments.
- Use shared containers to share common logic across a number of jobs. Remember that shared containers are inserted when a job is compiled. If the shared container is changed, the jobs using it will need recompiling.

Designing for good performance

Here are some tips for designing good performance into your job from the outset.

Avoid unnecessary type conversions.

Be careful to use proper source data types, especially from Oracle. You can set the OSH_PRINT_SCHEMAS environment variable to verify that runtime schemas match the job design column definitions.

If you are using stage variables on a Transformer stage, ensure that their data types match the expected result types.

Use Transformer stages sparingly and wisely

Do not have multiple stages where the functionality could be incorporated into a single stage, and use other stage types to perform simple transformation operations (see "Using Transformer Stages" for more guidance).

Increase sort performance where possible

Careful job design can improve the performance of sort operations, both in standalone Sort stages and in on-link sorts specified in the Inputs page Partitioning tab of other stage types. See "Sorting Data" for guidance.

Remove unneeded columns

Remove unneeded columns as early as possible within the job flow. Every additional unused column requires additional buffer memory, which can impact performance and make each row transfer from one stage to the next more expensive. If possible, when reading from databases, use a select list to read just the columns required, rather than the entire table.

Avoid reading from sequential files using the Same partitioning method.

Unless you have specified more than one source file, this will result in the entire file being read into a single partition, making the entire downstream flow run sequentially unless you explicitly repartition (see "Using Sequential File Stages" for more tips on using Sequential file stages).

Combining data

The two major ways of combining data in a WebSphere DataStage job are via a Lookup stage or a Join stage. How do you decide which one to use?

Lookup and Join stages perform equivalent operations: combining two or more input data sets based on one or more specified keys. When one unsorted input is very large or sorting is not feasible, Lookup is preferred. When all inputs are of manageable size or are pre-sorted, Join is the preferred solution.

The Lookup stage is most appropriate when the reference data for all Lookup stages in a job is small enough to fit into available physical memory. Each lookup reference requires a contiguous block of physical memory. The Lookup stage requires all but the first input (the primary input) to fit into physical memory.

If the reference to a lookup is directly from a DB2® or Oracle table and the number of input rows is significantly smaller than the reference rows, 1:100 or more, a Sparse Lookup might be appropriate.

If performance issues arise while using Lookup, consider using the Join stage. The Join stage must be used if the data sets are larger than available memory resources.

Sorting data

Look at job designs and try to reorder the job flow to combine operations around the same sort keys if possible, and coordinate your sorting strategy with your hashing strategy. It is sometimes possible to rearrange the order of business logic within a job flow to leverage the same sort order, partitioning, and groupings.

If data has already been partitioned and sorted on a set of key columns, specify the "don't sort, previously sorted" option for the key columns in the Sort stage. This reduces the cost of sorting and takes greater advantage of pipeline parallelism.

When writing to parallel data sets, sort order and partitioning are preserved. When reading from these data sets, try to maintain this sorting if possible by using Same partitioning method.

The stable sort option is much more expensive than non-stable sorts, and should only be used if there is a need to maintain row order other than as needed to perform the sort.

The performance of individual sorts can be improved by increasing the memory usage per partition using the Restrict Memory Usage (MB) option of the Sort stage. The default setting is 20 MB per partition. Note that sort memory usage can only be specified for standalone Sort stages, it cannot be changed for inline (on a link) sorts.

Default and explicit type conversions

When you are mapping data from source to target you might need to perform data type conversions. Some conversions happen automatically, and these can take place across the output mapping of any parallel job stage that has an input and an output link. Other conversions need a function to explicitly perform the conversion. These functions can be called from a Modify stage or a Transformer stage, and are listed in Appendix B of *WebSphere DataStage Parallel Job Developer Guide*. (Modify is the preferred stage for such conversions - see "Using Transformer Stages".)

The following table shows which conversions are performed automatically and which need to be explicitly performed. "d" indicates automatic (default) conversion, "m" indicates that manual conversion is required, a blank square indicates that conversion is not possible:

Table 1. Default and explicit type conversions, part 1

	Target									
Source	int8	uint8	int16	uint16	int32	uint32	int64	uint64	sfloat	dfloat
int8		d	d	d	d	d	d	d	d	d m
uint8	d		d	d	d	d	d	d	d	d
int16	d m	d		d	d	d	d	d	d	d
uint16	d	d	d		d	d	d	d	d	d

Table 1. Default and explicit type conversions, part 1 (continued)

	Target									
Source	int8	uint8	int16	uint16	int32	uint32	int64	uint64	sfloat	dfloat
int32	d m	d	d	d		d	d	d	d	d
uint32	d	d	d	d	d		d	d	d	d
int64	d m	d	d	d	d	d		d	d	d
uint64	d	d	d	d	d	d	d		d	d
sfloat	d m	d	d	d	d	d	d	d		d
dfloat	d m	d	d	d	d	d	d	d	d	
decimal	d m	d	d	d	d m	d	d m	d m	d	d m
string	d m	d	d m	d	d	d m	d	d	d	d m
ustring	d m	d	d m	d	d	d m	d	d	d	d m
raw	m				m					
date	m		m		m	m				
time	m				m					m
timestamp	m				m					m

Table 2. Default and explicit type conversions, part 2

	Target								
Source	decimal	string	ustring	raw	date	time	timestamp		
int8	d	d m	d m		m	m		m	
uint8	d	d	d						
int16	d	d m	d m						
uint16	d	d m	d m						
int32	d	d m	d m		m			m	
uint32	d	m	m		m				
int64	d	d	d						
uint64	d	d	d						
sfloat	d	d	d						
dfloat	d m	d m	d m			m		m	
decimal		d m	d m						
string	d m		d		m	m		m	
ustring	d m	d				m		m	
raw									
date		m	m					m	
time		m	m						d m
timestamp		m	m		m	m			

d = default conversion; m = modify operator conversion; blank = no conversion needed or provided

You should also note the following points about type conversion:

- When converting from variable-length to fixed-length strings using default conversions, parallel jobs pad the remaining length with NULL (ASCII zero) characters.
- The environment variable APT_STRING_PADCHAR can be used to change the default pad character from an ASCII NULL (0x0) to another character; for example, an ASCII space (0x20) or a unicode space (U+0020).
- As an alternate solution, the PadString function can be used to pad a variable-length (Varchar) string to a specified length using a specified pad character. Note that PadString does not work with fixed-length (Char) string types. You must first convert Char to Varchar before using PadString.

Using Transformer stages

In general, it is good practice not to use more Transformer stages than you have to. You should especially avoid using multiple Transformer stages where the logic can be combined into a single stage.

It is often better to use other stage types for certain types of operation:

- Use a Copy stage rather than a Transformer for simple operations such as:
 - Providing a job design placeholder on the canvas. (Provided you do not set the Force property to True on the Copy stage, the copy will be optimized out of the job at run time.)
 - Renaming columns.
 - Dropping columns.
 - Implicit type conversions (see "Default and Explicit Type Conversions").
 Note that, if runtime column propagation is disabled, you can also use output mapping on a stage to rename, drop, or convert columns on a stage that has both inputs and outputs.
- Use the Modify stage for explicit type conversion (see "Default and Explicit Type Conversions") and null handling.
- Where complex, reusable logic is required, or where existing Transformer-stage based job flows do not meet performance requirements, consider building your own custom stage (see [c_deadvrf_Specifying_Your_Own_Parallel_Stages.dita](#).)
- Use a BASIC Transformer stage where you want to take advantage of user-defined functions and routines.

Using Sequential File stages

Certain considerations apply when reading and writing fixed-length fields using the Sequential File stage.

- If reading columns that have an inherently variable-width type (for example, integer, decimal, or varchar) then you should set the Field Width property to specify the actual fixed-width of the input column. Do this by selecting **Edit Row...** from the shortcut menu for a particular column in the Columns tab, and specify the width in the Edit Column Meta Data dialog box.
- If writing fixed-width columns with types that are inherently variable-width, then set the Field Width property and the Pad char property in the Edit Column Meta Data dialog box to match the width of the output column.

Other considerations are as follows:

- If a column is nullable, you must define the null field value and length in the Edit Column Meta Data dialog box.
- Be careful when reading delimited, bounded-length varchar columns (that is, varchars with the length option set). If the source file has fields which are longer than the maximum varchar length, these extra characters are silently discarded.
- Avoid reading from sequential files using the Same partitioning method. Unless you have specified more than one source file, this will result in the entire file being read into a single partition, making the entire downstream flow run sequentially unless you explicitly repartition.

Using Database stages

The best choice is to use connector stages if available for your database. The next best choice are the Enterprise' database stages as these give maximum parallel performance and features when compared to 'plug-in' stages. The Enterprise stages are:

- DB2/UDB Enterprise
- Informix® Enterprise
- Oracle Enterprise
- Teradata Enterprise
- SQLServer Enterprise
- Sybase Enterprise
- ODBC Enterprise
- iWay Enterprise
- Netezza Enterprise

You should avoid generating target tables in the database from your WebSphere DataStage job (that is, using the Create write mode on the database stage) unless they are intended for temporary storage only. This is because this method does not allow you to, for example, specify target table space, and you might inadvertently violate data-management policies on the database. If you want to create a table on a target database from within a job, use the Open command property on the database stage to explicitly create the table and allocate tablespace, or any other options required.

The Open command property allows you to specify a command (for example some SQL) that will be executed by the database before it processes any data from the stage. There is also a Close property that allows you to specify a command to execute after the data from the stage has been processed. (Note that, when using user-defined Open and Close commands, you might need to explicitly specify locks where appropriate.)

Database sparse lookup vs. join

Data read by any database stage can serve as the reference input to a Lookup stage. By default, this reference data is loaded into memory like any other reference link.

When directly connected as the reference link to a Lookup stage, both DB2/UDB Enterprise and Oracle Enterprise stages allow the lookup type to be changed to Sparse and send individual SQL statements to the reference database for each incoming Lookup row. Sparse Lookup is only available when the database stage is directly connected to the reference link, with no intermediate stages.

It is important to note that the individual SQL statements required by a Sparse Lookup are an expensive operation from a performance perspective. In most cases, it is faster to use a WebSphere DataStage Join stage between the input and DB2 reference data than it is to perform a Sparse Lookup.

For scenarios where the number of input rows is significantly smaller (1:100 or more) than the number of reference rows in a DB2 or Oracle table, a Sparse Lookup might be appropriate.

DB2 database tips

If available, use the DB2 connector. Otherwise, always use the DB2 Enterprise stage in preference to the DB2/API plugin stage for reading from, writing to, and performing lookups against a DB2 Enterprise Server Edition with the Database Partitioning Feature (DBF). The DB2 Enterprise stage is designed for maximum performance and scalability against very large partitioned DB2 UNIX® databases.

The DB2/API plugin should only be used to read from and write to DB2 on other, non-UNIX platforms. You might, for example, use it to access mainframe editions through DB2 Connect™.

Write vs. load

The DB2 Enterprise stage offers the choice between SQL methods (insert, update, upsert, delete) or fast loader methods when writing to a DB2 database. The choice between these methods depends on the required performance, database log usage, and recoverability considerations as follows:

- The write method (using insert, update, upsert, or delete) communicates directly with DB2 database nodes to execute instructions in parallel. All operations are logged to the DB2 database log, and the target table(s) can be accessed by other users. Time and row-based commit intervals determine the transaction size and availability of new rows to other applications.
- The load method requires that the user running the job has DBADM privilege on the target database. During a load operation an exclusive lock is placed on the entire DB2 tablespace into which the data is being loaded, and so this tablespace cannot be accessed by anyone else while the load is taking place. The load is also non-recoverable: if the load operation is terminated before it is completed, the contents of the table are unusable and the tablespace is left in the load pending state. If this happens, the WebSphere DataStage job must be re-run with the stage set to truncate mode to clear the load pending state.

Oracle database tips

When designing jobs that use Oracle sources or targets, note that the parallel engine will use its interpretation of the Oracle meta data (for example, exact data types) based on interrogation of Oracle, overriding what you might have specified in the Columns tab. For this reason it is best to import your Oracle table definitions using the **Import → Orchestrate Schema Definitions** command from the WebSphere DataStage Designer. Choose the Database table option and follow the instructions from the wizard.

Loading and indexes

When you use the Load write method in an Oracle Enterprise stage, you are using the Parallel Direct Path load method. If you want to use this method to write tables that have indexes on them (including indexes automatically generated by primary key constraints), you must specify the Index Mode property (you can set it to Maintenance or Rebuild). An alternative is to set the environment variable APT_ORACLE_LOAD_OPTIONS to "OPTIONS (DIRECT=TRUE, PARALLEL=FALSE)". This allows the loading of indexed tables without index maintenance, but the load is performed sequentially.

You can use the upsert write method to insert rows into an Oracle table without bypassing indexes or constraints. In order to automatically generate the SQL needed, set the Upsert Mode property to Auto-generated and identify the key column(s) on the Columns tab by selecting the **Key** check boxes.

Teradata Database Tips

You can use the Additional Connections Options property in the Teradata Enterprise stage (which is a dependent of DB Options Mode) to specify details about the number of connections to Teradata. The possible values of this are:

- sessionsperplayer. This determines the number of connections each player in the job has to Teradata. The number should be selected such that:
$$(\text{sessions per player} * \text{number of nodes} * \text{players per node}) = \text{total requested sessions}$$
The default value is 2. Setting this too low on a large system can result in so many players that the job fails due to insufficient resources.
- requestedsessions. This is a number between 1 and the number of vprocs in the database. The default is the maximum number of available sessions.

Chapter 3. Improving performance

Use the information in these topics to help resolve any performance problems.

These topics assume that basic steps to assure performance have been taken: a suitable configuration file has been set up, reasonable swap space configured and so on, and that you have followed the design guidelines laid down in Chapter 2, "Job design tips," on page 3.

Understanding a flow

In order to resolve any performance issues it is essential to have an understanding of the flow of WebSphere DataStage jobs.

Score dumps

To help understand a job flow you might take a score dump. Do this by setting the APT_DUMP_SCORE environment variable true and running the job (APT_DUMP_SCORE can be set in the Administrator client, under the **Parallel** → **Reporting** ranch). This causes a report to be produced which shows the operators, processes and data sets in the job. The report includes information about:

- Where and how data is repartitioned.
- Whether WebSphere DataStage had inserted extra operators in the flow.
- The degree of parallelism each operator runs with, and on which nodes.
- Information about where data is buffered.

The dump score information is included in the job log when you run a job.

The score dump is particularly useful in showing you where WebSphere DataStage is inserting additional components in the job flow. In particular WebSphere DataStage will add partition and sort operators where the logic of the job demands it. Sorts in particular can be detrimental to performance and a score dump can help you to detect superfluous operators and amend the job design to remove them.

Example score dump

The following score dump shows a flow with a single data set, which has a hash partitioner, partitioning on key "a". It shows three operators: generator, tsort, and peek. Tsort and peek are "combined", indicating that they have been optimized into the same process. All the operators in this flow are running on one node.

```
##I TFSC 004000 14:51:50(000) <main_program>
This step has 1 data set:
ds0: {op0[1p]} (sequential generator)
  e0ther(APT_HashPartitioner { key={ value=a } })
})->eCollectAny
  op1[2p] (parallel APT_CombinedOperatorController:tsort)
It has 2 operators:
op0[1p] {(sequential generator)
  on nodes (
    1emond.torrent.com[op0,p0]
  )}
op1[2p] {(parallel APT_CombinedOperatorController:
  (tsort)
  (peek)
  ) on nodes (
```

```
lemond.torrent.com[op1,p0]
lemond.torrent.com[op1,p1]
)}
It runs 3 processes on 2 nodes.
```

Tips for debugging

- Use the Data Set Management utility, which is available in the **Tools** menu of the WebSphere DataStage Designer to examine the schema, look at row counts, and delete a Parallel Data Set. You can also view the data itself.
- Check the WebSphere DataStage job log for warnings. These might indicate an underlying logic problem or unexpected data type conversion.
- Enable the APT_DUMP_SCORE and APT_RECORD_COUNTS environment variables. Also enable OSH_PRINT_SCHEMAS to ensure that a runtime schema of a job matches the design-time schema that was expected.
- The UNIX command od -xc displays the actual data contents of any file, including any embedded ASCII NULL characters.
- The UNIX command, wc -lc *filename*, displays the number of lines and characters in the specified ASCII text file. Dividing the total number of characters by the number of lines provides an audit to ensure that all rows are the same length. It is important to know that the wc utility works by counting UNIX line delimiters, so if the file has any binary columns, this count might be incorrect.

Performance monitoring

There are various tools you can use to aid performance monitoring, some provided with WebSphere DataStage and some general UNIX tools.

Job monitor

You access the WebSphere DataStage job monitor through the WebSphere DataStage Director (see *WebSphere DataStage Director Client Guide*). You can also use certain dsjob commands from the command line to access monitoring functions (see "Retrieving Information" for details).

The job monitor provides a useful snapshot of a job's performance at a moment of execution, but does not provide thorough performance metrics. That is, a job monitor snapshot should not be used in place of a full run of the job, or a run with a sample set of data. Due to buffering and to some job semantics, a snapshot image of the flow might not be a representative sample of the performance over the course of the entire job.

The CPU summary information provided by the job monitor is useful as a first approximation of where time is being spent in the flow. However, it does not include any sorts or similar that might be inserted automatically in a parallel job. For these components, the score dump can be of assistance. See "Score Dumps".

A worst-case scenario occurs when a job flow reads from a data set, and passes immediately to a sort on a link. The job will appear to hang, when, in fact, rows are being read from the data set and passed to the sort.

The operation of the job monitor is controlled by two environment variables: APT_MONITOR_TIME and APT_MONITOR_SIZE. By default the job monitor takes a snapshot every five seconds. You can alter the time interval by changing the value of APT_MONITOR_TIME, or you can have the monitor generate a new snapshot every so-many rows by following this procedure:

1. Select APT_MONITOR_TIME on the WebSphere DataStage Administrator environment variable dialog box, and press the **set to default** button.
2. Select APT_MONITOR_SIZE and set the required number of rows as the value for this variable.

Iostat

The UNIX tool Iostat is useful for examining the throughput of various disk resources. If one or more disks have high throughput, understanding where that throughput is coming from is vital. If there are spare CPU cycles, IO is often the culprit.

The specifics of Iostat output vary slightly from system to system. Here is an example from a Linux® machine which shows a relatively light load:

(The first set of output is cumulative data since the machine was booted)

```
Device: tps Blk_read/s Blk_wrtn/s Blk_read Blk_wrtn
dev8-0 13.50 144.09      122.33    346233038 293951288
...
Device: tps Blk_read/s Blk_wrtn/s Blk_read Blk_wrtn
dev8-0 4.00 0.00       96.00      0          96
```

Load average

Ideally, a performant job flow should be consuming as much CPU as is available. The load average on the machine should be two to three times the value as the number of processors on the machine (for example, an 8-way SMP should have a load average of roughly 16-24). Some operating systems, such as HPUX, show per-processor load average. In this case, load average should be 2-3, regardless of number of CPUs on the machine.

If the machine is not CPU-saturated, it indicates a bottleneck might exist elsewhere in the flow. A useful strategy in this case is to over-partition your data, as more partitions cause extra processes to be started, utilizing more of the available CPU power.

If the flow causes the machine to be fully loaded (all CPUs at 100%), then the flow is likely to be CPU limited, and some determination needs to be made as to where the CPU time is being spent (setting the APT_PM_PLAYER_TIMING environment variable can be helpful here - see the following section).

The commands top or uptime can provide the load average.

Runtime information

When you set the APT_PM_PLAYER_TIMING environment variable, information is provided for each operator in a job flow. This information is written to the job log when the job is run.

An example output is:

```
##I TFPMP 000324 08:59:32(004) <generator,0> Calling runLocally: step=1, node=rh73dev04, op=0, ptn=0
##I TFPMP 000325 08:59:32(005) <generator,0> Operator completed. status: APT_StatusOk elapsed: 0.04
user: 0.00 sys: 0.00 suser: 0.09 ssy: 0.02 (total CPU: 0.11)
##I TFPMP 000324 08:59:32(006) <peek,0> Calling runLocally: step=1, node=rh73dev04, op=1, ptn=0
##I TFPMP 000325 08:59:32(012) <peek,0> Operator completed. status: APT_StatusOk elapsed: 0.01 user:
0.00 sys: 0.00 suser: 0.09 ssy: 0.02 (total CPU: 0.11)
##I TFPMP 000324 08:59:32(013) <peek,1> Calling runLocally: step=1, node=rh73dev04a, op=1, ptn=1
##I TFPMP 000325 08:59:32(019) <peek,1> Operator completed. status: APT_StatusOk elapsed: 0.00 user:
0.00 sys: 0.00 suser: 0.09 ssy: 0.02 (total CPU: 0.11)"
```

This output shows that each partition of each operator has consumed about one tenth of a second of CPU time during its runtime portion. In a real world flow, we'd see many operators, and many partitions.

It is often useful to see how much CPU each operator (and each partition of each component) is using. If one partition of an operator is using significantly more CPU than others, it might mean the data is partitioned in an unbalanced way, and that repartitioning, or choosing different partitioning keys might be a useful strategy.

If one operator is using a much larger portion of the CPU than others, it might be an indication that you've discovered a problem in your flow. Common sense is generally required here; a sort is going to use dramatically more CPU time than a copy. This will, however, give you a sense of which operators are the CPU hogs, and when combined with other metrics presented in this document can be very enlightening.

Setting the environment variable APT_DISABLE_COMBINATION might be useful in some situations to get finer-grained information as to which operators are using up CPU cycles. Be aware, however, that setting this flag will change the performance behavior of your flow, so this should be done with care.

Unlike the job monitor cpu percentages, setting APT_PM_PLAYER_TIMING will provide timings on every operator within the flow.

Performance data

You can record performance data about job objects and computer resource utilization in parallel job runs.

You can record performance data in these ways:

- At design time, with the Designer client
- At run time, with either the Designer client or the Director client

Performance data is written to an XML file that is in the default directory C:\IBM\InformationServer\Server\Performance. You can override the default location by setting the environment variable APT_PERFORMANCE_DATA. Use the Administrator client to set a value for this variable at the project level, or use the Parameters page of the Job Properties window to specify a value at the job level.

Recording performance data at design time

At design time, you can set a flag to specify that you want to record performance data when the job runs.

To record performance data at design time:

1. Open a job in the Designer client.
2. Click **Edit** > **Job Properties**.
3. Click the **Execution** page.
4. Select the **Record job performance data** check box.

Performance data is recorded each time that the job runs successfully.

Recording performance data at run time

You can use the Designer client or the Director client to record performance data at run time.

To record performance data at run time:

1. Open a job in the Designer client, or select a job in the display area of the Director client.
2. Click the **Run** button on the toolbar to open the **Job Run Options** window.
3. Click the **General** page.
4. Select the **Record job performance data** check box.

Performance data is recorded each time that the job runs successfully.

Viewing performance data

Use the Performance Analysis window to view charts that interpret job performance and computer resource utilization.

First you must record data about job performance. See "Recording performance data at design time" or "Recording performance data at run time" for more information.

You can view performance data in either the Designer client or the Director client.

To view performance data:

1. Open the Performance Analysis window by using one of the following methods:
 - In the Designer client, click **File** → **Performance Analysis**.
 - In the Director client, click **Job** → **Analyze Performance**.
 - In either client, click the **Performance Analysis** toolbar button.
2. In the Performance Data group in the left pane, select the job run that you want to analyze. Job runs are listed in descending order according to the timestamp.
3. In the Charts group, select the chart that you want to view.
4. If you want to exclude certain job objects from a chart, use one of the following methods:
 - For individual objects, clear the check boxes in the **Job Tree** group.
 - For all objects of the same type, clear the check boxes in the **Partitions**, **Stages**, and **Phases** groups.
5. Optional: In the **Filters** group, change how data is filtered in a chart.
6. Click **Save** to save the job performance data in an archive. The archive includes the following files:
 - Performance data file named `performance.xxxx` (where `xxxx` is the suffix that is associated with the job run)
 - Computer descriptions file named `description.xxxx`
 - Computer utilization file named `utilization.xxxx`
 - Exported job definition named `exportedjob.xxxx`

When you open a performance data file, the system creates a mapping between the job stages that are displayed on the Designer client canvas and the operating system processes that define a job. The mapping might not create a direct relationship between stages and processes for these reasons:

- Some stages compile into many processes.
- Some stages are combined into a single process.

You can use the check boxes in the **Filters** area of the Performance Analysis window to include data about hidden operators in the performance data file. For example, Modify stages are combined with the previous stage in a job design. If you want to see the percentage of elapsed time that is used by a modify operator, clear the **Hide Inserted Operators** check box. Similarly, you can clear the **Hide Composite Operators** check box to expose performance data about composite operators.

You can delete performance data files by clicking **Delete**. All of the data files that belong to the selected job run, including the performance data, utilization data, computer description data, and job export data, are deleted from the server.

OS/RDBMS specific tools

Each OS and RDBMS has its own set of tools which might be useful in performance monitoring. Talking to the sysadmin or DBA might provide some useful monitoring strategies.

Performance analysis

Once you have carried out some performance monitoring, you can analyze your results.

Bear in mind that, in a parallel job flow, certain operators might complete before the entire flow has finished, but the job isn't successful until the slowest operator has finished all its processing.

Selectively rewriting the flow

One of the most useful mechanisms in detecting the cause of bottlenecks in your flow is to rewrite portions of it to exclude stages from the set of possible causes. The goal of modifying the flow is to see the new, modified, flow run noticeably faster than the original flow. If the flow is running at roughly an identical speed, change the flow further.

While editing a flow for testing, it is important to keep in mind that removing one stage might have unexpected affects in the flow. Comparing the score dump between runs is useful before concluding what has made the performance difference.

When modifying the flow, be aware of introducing any new performance problems. For example, adding a Data Set stage to a flow might introduce disk contention with any other data sets being read. This is rarely a problem, but might be significant in some cases.

Moving data into and out of parallel operation are two very obvious areas of concern. Changing a job to write into a Copy stage (with no outputs) will throw the data away. Keep the degree of parallelism the same, with a nodemap if necessary. Similarly, landing any read data to a data set can be helpful if the data's point of origin is a flat file or RDBMS.

This pattern should be followed, removing any potentially suspicious operators while trying to keep the rest of the flow intact. Removing any custom stages should be at the top of the list.

Identifying superfluous repartitions

Superfluous repartitioning should be identified. Due to operator or license limitations (import, export, RDBMS ops, SAS, and so on) some stages will run with a degree of parallelism that is different than the default degree of parallelism. Some of these cannot be eliminated, but understanding the where, when and why these repartitions occur is important for flow analysis. Repartitions are especially expensive when the data is being repartitioned on an MPP system, where significant network traffic will result.

Sometimes you might be able to move a repartition upstream in order to eliminate a previous, implicit repartition. Imagine an Oracle stage performing a read (using the oraread operator). Some processing is done on the data and it is then hashed and joined with another data set. There might be a repartition after the oraread operator, and then the hash, when only one repartition is really necessary.

Similarly, specifying a nodemap for an operator might prove useful to eliminate repartitions. In this case, a transform stage sandwiched between a DB2 stage reading (db2read) and another one writing (db2write) might benefit from a nodemap placed on it to force it to run with the same degree of parallelism as the two db2 operators to avoid two repartitions.

Identifying buffering issues

Buffering is one of the more complex aspects to parallel job performance tuning. Buffering is described in detail in "Buffering",

The goal of buffering on a specific link is to make the producing operator's output rate match the consumption rate of the downstream operator. In any flow where this is incorrect behavior for the flow (for example, the downstream operator has two inputs, and waits until it had exhausted one of those inputs before reading from the next) performance is degraded. Identifying these spots in the flow requires an understanding of how each operator involved reads its record, and is often only found by empirical observation.

You can diagnose a buffering tuning issue when a flow runs slowly when it is one massive flow, but each component runs quickly when broken up. For example, replacing an Oracle write stage with a copy stage vastly improves performance, and writing that same data to a data set, then loading via an Oracle stage, also goes quickly. When the two are put together, performance is poor.

"Buffering" details specific, common buffering configurations aimed at resolving various bottlenecks.

Resource estimation

New in this release, you can estimate and predict the resource utilization of parallel job runs by creating models and making projections in the Resource Estimation window.

A model estimates the system resources for a job, including the amount of scratch space, disk space, and CPU time that is needed for each stage to run on each partition. A model also estimates the data set throughput in a job. You can generate these types of models:

- *Static models* estimate disk space and scratch space only. These models are based on a data sample that is automatically generated from the record schema. Use static models at compilation time.
- *Dynamic models* predict disk space, scratch space, and CPU time. These models are based on a sampling of actual input data. Use dynamic models at run time.

An input projection estimates the size of all of the data sources in a job. You can project the size in megabytes or in number of records. A default projection is created when you generate a model.

The resource utilization results from a completed job run are treated as an actual model. A job can have only one actual model. In the Resource Estimation window, the actual model is the first model in the **Models** list. Similarly, the total size of the data sources in a completed job run are treated as an actual projection. You must select the actual projection in the **Input Projections** list to view the resource utilization statistics in the actual model. You can compare the actual model to your generated models to calibrate your modeling techniques.

Creating a model

You can create a static or dynamic model to estimate the resource utilization of a parallel job run.

You can create models in the Designer client or the Director client. You must compile a job before you create a model.

To create a model:

1. Open a job in the Designer client, or select a job in the Director client.
2. Open the Resource Estimation window by using one of the following methods:
 - In the Designer, click **File** → **Estimate Resource**.
 - In the Director, click **Job** → **Estimate Resource**.
 - Click the **Resource Estimation** toolbar button.

The first time that you open the Resource Estimation window for a job, a static model is generated by default.

3. Click the **Model** toolbar button to display the Create Resource Model options.
4. Type a name in the **Model Name** field. The specified name must not already exist.
5. Select a type in the **Model Type** field.
6. If you want to specify a data sampling range for a dynamic model, use one of the following methods:
 - Click the **Copy Previous** button to copy the sampling specifications from previous models, if any exist.
 - Clear the **Auto** check box for a data source, and type values in the **From** and **To** fields to specify a record range.
7. Click **Generate**.

After the model is created, the Resource Estimation window displays an overview of the model that includes the model type, the number of data segments in the model, the input data size, and the data sampling description for each input data source.

Use the controls in the left pane of the Resource Estimation window to view statistics about partition utilization, data set throughput, and operator utilization in the model. You can also compare the model to other models that you generate.

Static and dynamic models

Static models estimate resource utilization at compilation time. Dynamic models predict job performance at run time.

The following table describes the differences between static and dynamic models. Use this table to help you decide what type of model to generate.

Table 3. Static and dynamic models

Characteristics	Static models	Dynamic models
Job run	Not required.	Required.
Sample data	Requires automatic data sampling. Uses the actual size of the input data if the size can be determined. Otherwise, the sample size is set to a default value of 1000 records on each output link from each source stage.	<p>Accepts automatic data sampling or a data range:</p> <ul style="list-style-type: none"> • Automatic data sampling determines the sample size dynamically according to the stage type: <ul style="list-style-type: none"> – For a database source stage, the sample size is set to 1000 records on each output link from the stage. – For all other source stage types, the sample size is set to the minimum number of input records among all sources on all partitions. • A data range specifies the number of records to include in the sample for each data source. <p>If the size of the sample data exceeds the actual size of the input data, the model uses the entire input data set.</p>
Scratch space	Estimates are based on a worst-case scenario.	Estimates are based on linear regression.
Disk space	Estimates are based on a worst-case scenario.	Estimates are based on linear regression.
CPU utilization	Not estimated.	Estimates are based on linear regression.
Number of records	Estimates are based on a best-case scenario. No record is dropped. Input data is propagated from the source stages to all other stages in the job.	Dynamically determined. Best-case scenario does not apply. Input data is processed, not propagated. Records can be dropped. Estimates are based on linear regression.
Record size	Solely determined by the record schema. Estimates are based on a worst-case scenario.	Dynamically determined by the actual record at run time. Estimates are based on linear regression.

Table 3. Static and dynamic models (continued)

Characteristics	Static models	Dynamic models
Data partitioning	Data is assumed to be evenly distributed among all partitions.	Dynamically determined. Estimates are based on linear regression.

When a model is based on a worst-case scenario, the model uses maximum values. For example, if a variable can hold up to 100 characters, the model assumes that the variable always holds 100 characters. When a model is based on a best-case scenario, the model assumes that no single input record is dropped anywhere in the data flow.

The accuracy of a model depends on these factors:

Schema definition

The size of records with variable-length fields cannot be determined until the records are processed. Use fixed-length or bounded-length schemas as much as possible to improve accuracy.

Input data

When the input data contains more records with one type of key field than another, the records might be unevenly distributed across partitions. Specify a data sampling range that is representative of the input data.

Parallel processing environment

The availability of system resources when you run a job can affect the degree to which buffering occurs. Generate models in an environment that is similar to your production environment in terms of operating system, processor type, and number of processors.

Custom stages and dynamic models

To be estimated in dynamic models, Custom stages must support the end-of-wave functionality in the parallel engine.

If a Custom stage serves as an import operator or needs scratch space or disk space, the stage must declare its type by calling the function APT_Operator::setExternalDataDirection() when the stage overrides the APT_Operator::describeOperator() function. Define the external data direction by using the following enumerated type:

```
enum externalDataDirection{
    eNone,
    /** Data "source" operator - an import or database read. */
    eDataSource,
    /** Data "sink" operator - an export or database write. */
    eDataSink,
    /** Data "scratch" operator - a tsort or buffer operator. */
    eDataScratch,
    /** Data "disk" operator - an export or dataset/fileset. */
    eDataDisk
};
```

Custom stages that need disk space and scratch space must call two additional functions within the dynamic scope of APT_Operator::runLocally():

- For disk space, call APT_Operator::setDiskSpace() to describe actual disk space usage.
- For scratch space, call APT_Operator::setScratchSpace() to describe actual scratch space usage.

Both functions accept values of APT_Int64.

Making a projection

You can make a projection to predict the resource utilization of a job by specifying the size of the data sources.

You must generate at least one model before you make a projection. Projections are applied to all existing models, except the actual model.

To make a projection:

1. Open a job in the Designer client, or select a job in the Director client.
2. Open the Resource Estimation window by using one of the following methods:
 - In the Designer, click **File** → **Estimate Resource**.
 - In the Director, click **Job** → **Estimate Resource**.
 - Click the **Resource Estimation** toolbar button.
3. Click the **Projection** toolbar button to display the Make Resource Projection options.
4. Type a name in the **Projection Name** field. The specified name must not already exist.
5. Select the unit of measurement for the projection in the **Input Units** field.
6. Specify the input size upon which to base the projection by using one of the following methods:
 - Click the **Copy Previous** button to copy the specifications from previous projections, if any exist.
 - If the **Input Units** field is set to Size in Megabytes, type a value in the **Megabytes (MB)** field for each data source.
 - If the **Input Units** field is set to Number of Records, type a value in the **Records** field for each data source.
7. Click **Generate**.

The projection applies the input data information to the existing models, excluding the actual model, to predict the resource utilization for the given input data.

Generating a resource estimation report

You can generate a report to see a resource estimation summary for a selected model.

Reports contain an overview of the job, the model, and the input projection. The reports also give statistics about the partition utilization and data set throughput for each data source.

To generate a report:

1. In the Resource Estimation window, select a model in the **Models** list.
2. Select a projection in the **Input Projections** list. If you do not select a projection, the default projection is used.
3. Click the **Report** toolbar button.

By default, reports are saved in the following directory:

C:\IBM\InformationServer\Clients\Classic\Estimatation\server_name\project_name\job_name\html\report.html

You can print the report or rename it by using the controls in your Web browser.

Examples of resource estimation

You can use models and projections during development to optimize job design and to configure your environment for more efficient processing.

The following examples show how to use resource estimation techniques for a job that performs these tasks:

- Consolidates input data from three distributed data sources
- Merges the consolidated data with records from a fourth data source

- Updates records in the fourth data source with the current date
- Saves the merged records to two different data sets based on the value of a specific field

In this example, each data source has 5 million records.

You can use resource estimation models and projections to answer questions such as these:

- Which stage merges data most efficiently?
- When should data be sorted?
- Are there any performance bottlenecks?
- What are the disk and scratch space requirements if the size of the input data increases?

Example - Find the best stage to merge data

In this example, you create models to determine whether a Lookup, Merge, or Join stage is the most efficient stage to merge data.

The example job consolidates input data from three data sources and merges this data with records from a fourth data source. You can use Lookup, Merge, or Join stages to merge data. Lookup stages do not require the input data to be sorted, but the stage needs more memory to create a dynamic lookup table. Merge and Join stages require that the input data is sorted, but use less memory than a Lookup stage. To find out which stage is most efficient, design three jobs:

- Job 1 uses a Lookup stage to merge data. One input link to the Lookup stage carries the consolidated data from the three data sources. The other input link carries the data from the fourth data source and includes an intermediate Transformer stage to insert the current date.
- Job 2 uses a Merge stage to merge data. One input link to the Merge stage carries the consolidated data from the three data sources. The other input link carries the data from the fourth data source. This job includes an intermediate Transformer stage to insert the current date and a Sort stage to sort the data.
- Job 3 uses a Join stage to merge data by using a left outer join. One input link to the Join stage carries the consolidated data from the three data sources. The other input link carries the data from the fourth data source. This job includes an intermediate Transformer stage to insert the current date and a Sort stage to sort the data.

The next step is to generate an automatic dynamic model for each job. The models are based on a single-node configuration on Windows XP, with a 1.8 GHz processor and 2 GB of RAM. The following table summarizes the resource utilization statistics for each job:

Table 4. Resource utilization statistics

Job	CPU (seconds)	Disk (MB)	Scratch (MB)
Job 1 (Lookup stage)	229.958	801.125	0
Job 2 (Merge stage)	219.084	801.125	915.527
Job 3 (Join stage)	209.25	801.125	915.527

By comparing the models, you see that Job 1 does not require any scratch space, but is the slowest of the three jobs. The Lookup stage also requires memory to build a lookup table for a large amount of reference data. Therefore, the optimal job design uses either a Merge stage or a Join stage to merge data.

Example - Decide when to sort data

In this example, you create models to decide whether to sort data before or after the data is consolidated from three data sources.

The previous example demonstrated that a Merge or Join stage is most efficient to merge data in the example job. These stage types require that the input data is sorted. Now you need to decide whether to sort the input data from your three data sources before or after you consolidate the data. To understand the best approach, design two jobs:

- Job 4 sorts the data first:
 1. Each data source is linked to a separate Sort stage.
 2. The sorted data is sent to a single Funnel stage for consolidation.
 3. The Funnel stage sends the data to the Merge or Join stage, where it is merged with the data from the fourth data source.
- Job 5 consolidates the data first:
 1. The three source stages are linked to a single Funnel stage that consolidates the data.
 2. The consolidated data is sent to a single Sort stage for sorting.
 3. The Sort stage sends the data to the Merge or Join stage, where it is merged with the data from the fourth data source.

Use the same processing configuration as in the first example to generate an automatic dynamic model for each job. The resource utilization statistics for each job are shown in the table:

Table 5. Resource utilization statistics

Job	CPU (seconds)	Disk (MB)	Scratch (MB)
Job 4 (Sort before Funnel)	74.6812	515.125	801.086
Job 5 (Sort after Funnel)	64.1079	515.125	743.866

You can see that sorting data after consolidation is a better design because Job 5 uses approximately 15% less CPU time and 8% less scratch space than Job 4.

Example - Find bottlenecks

In this example, you use the partition utilization statistics in a model to identify any performance bottlenecks in a job. Then, you apply a job parameter to remove the bottleneck.

Models in the previous examples describe how to optimize the design of the example job. The best performance is achieved when you:

1. Consolidate the data from your three sources.
2. Sort the data.
3. Use a Merge or a Join stage to merge the data with records from a fourth data source.

An intermediate Transformer stage adds the current date to the records in the fourth data source before the data is merged. The Transformer stage appends the current date to each input record by calling the function `DateToString(CurrentDate())` and assigning the returned value to a new output field. When you study the partition utilization statistics in the models, you notice a performance bottleneck in the Transformer stage:

- In Job 2, the Transformer stage uses 139.145 seconds out of the 219.084 seconds of total CPU time for the job.
- In Job 3, the Transformer stage uses 124.355 seconds out of the 209.25 seconds of total CPU time for the job.

A more efficient approach is to assign a job parameter to the new output field. After you modify the Transformer stage in each job, generate automatic dynamic models to compare the performance:

Table 6. Resource utilization statistics

Job	CPU (seconds)	Disk (MB)	Scratch (MB)
Job 6 (Merge stage with job parameter in Transformer stage)	109.065	801.125	915.527
Job 7 (Join stage with job parameter in Transformer stage)	106.5	801.125	915.527

Job performance is significantly improved after you remove the bottleneck in the Transformer stage. Total CPU time for Jobs 6 and 7 is about half of the total CPU time for Jobs 2 and 3. CPU time for the Transformer stage is a small portion of total CPU time:

- In Job 6, the Transformer stage uses 13.8987 seconds out of the 109.065 seconds of total CPU time for the job.
- In Job 7, the Transformer stage uses 13.1489 seconds out of the 106.5 seconds of total CPU time for the job.

These models also show that job performance improves by approximately 2.4% when you merge data by using a Join stage rather than a Merge stage.

Example - Project resource requirements

In this example, you make projections to find out how much disk space and scratch space are needed when the input data size increases.

Each data source in the example job has 5 million records. According to your previous models, Job 7 requires approximately 800 MB of disk space and 915 MB of scratch space. Suppose the size of each data source increases as follows:

- 18 million records for data source 1
- 20 million records for data source 2
- 22 million records for data source 3
- 60 million records for data source 4

Make a projection by specifying the increased number of records for each data source. When the projection is applied to the model for Job 7, the estimation shows that approximately 3204 Mb of disk space and 5035 Mb of scratch space are needed. By estimating the disk allocation, the projection helps you prevent a job from stopping prematurely due to a lack of disk space.

Resolving bottlenecks

Choosing the most efficient operators

Because WebSphere DataStage offers a wide range of different stage types, with different operators underlying them, there can be several different ways of achieving the same effects within a job. This section contains some hint as to preferred practice when designing for performance is concerned. When analyzing your flow you should try substituting preferred operators in particular circumstances.

Modify and transform

Modify, due to internal implementation details, is a particularly efficient operator. Any transformation which can be implemented in the Modify stage will be more efficient than implementing the same operation in a Transformer stage. Transformations that touch a single column (for example, keep/drop, type conversions, some string manipulations, null handling) should be implemented in a Modify stage rather than a Transformer.

Lookup and join

Lookup and join perform equivalent operations: combining two or more input data sets based on one or more specified keys.

Lookup requires all but one (the first or primary) input to fit into physical memory. Join requires all inputs to be sorted.

When one unsorted input is very large or sorting isn't feasible, lookup is the preferred solution. When all inputs are of manageable size or are pre-sorted, join is the preferred solution.

Partitioner insertion, sort insertion

Partitioner insertion and sort insertion each make writing a flow easier by alleviating the need for a user to think about either partitioning or sorting data. By examining the requirements of operators in the flow, the parallel engine can insert partitioners, collectors and sorts as necessary within a dataflow.

However, there are some situations where these features can be a hindrance.

If data is pre-partitioned and pre-sorted, and the WebSphere DataStage job is unaware of this, you could disable automatic partitioning and sorting for the whole job by setting the following environment variables while the job runs:

- APT_NO_PART_INSERTION
- APT_NO_SORT_INSERTION

You can also disable partitioning on a per-link basis within your job design by explicitly setting a partitioning method of Same on the Input page Partitioning tab of the stage the link is input to.

To disable sorting on a per-link basis, insert a Sort stage on the link, and set the **Sort Key Mode** option to **Don't Sort (Previously Sorted)**.

We advise that average users leave both partitioner insertion and sort insertion alone, and that power users perform careful analysis before changing these options.

Combinable Operators

Combined operators generally improve performance at least slightly (in some cases the difference is dramatic). There might also be situations where combining operators actually hurts performance, however. Identifying such operators can be difficult without trial and error.

The most common situation arises when multiple operators are performing disk I/O (for example, the various file stages and sort). In these sorts of situations, turning off combination for those specific stages might result in a performance increase if the flow is I/O bound.

Combinable operators often provide a dramatic performance increase when a large number of variable length fields are used in a flow.

Disk I/O

Total disk throughput is often a fixed quantity that WebSphere DataStage has no control over. It can, however, be beneficial to follow some rules.

- If data is going to be read back in, in parallel, it should never be written as a sequential file. A data set or file set stage is a much more appropriate format.
- When importing fixed-length data, the Number of Readers per Node property on the Sequential File stage can often provide a noticeable performance boost as compared with a single process reading the data.

- Some disk arrays have read ahead caches that are only effective when data is read repeatedly in like-sized chunks. Setting the environment variable APT_CONSISTENT_BUFFERIO_SIZE=N will force stages to read data in chunks which are size N or a multiple of N.
- Memory mapped I/O, in many cases, contributes to improved performance. In certain situations, however, such as a remote disk mounted via NFS, memory mapped I/O might cause significant performance problems. Setting the environment variables APT_IO_NOMAP and APT_BUFFERIO_NOMAP true will turn off this feature and sometimes affect performance. (AIX® and HP-UX default to NOMAP. Setting APT_IO_MAP and APT_BUFFERIO_MAP true can be used to turn memory mapped I/O on for these platforms.)

Ensuring data is evenly partitioned

Because of the nature of parallel jobs, the entire flow runs only as fast as its slowest component. If data is not evenly partitioned, the slowest component is often slow due to data skew. If one partition has ten records, and another has ten million, then a parallel job cannot make ideal use of the resources.

Setting the environment variable APT_RECORD_COUNTS displays the number of records per partition for each component. Ideally, counts across all partitions should be roughly equal. Differences in data volumes between keys often skew data slightly, but any significant (e.g., more than 5-10%) differences in volume should be a warning sign that alternate keys, or an alternate partitioning strategy, might be required.

Buffering

Buffering is intended to slow down input to match the consumption rate of the output. When the downstream operator reads very slowly, or not at all, for a length of time, upstream operators begin to slow down. This can cause a noticeable performance loss if the buffer's optimal behavior is something other than rate matching.

By default, each link has a 3 MB in-memory buffer. Once that buffer reaches half full, the operator begins to push back on the upstream operator's rate. Once the 3 MB buffer is filled, data is written to disk in 1 MB chunks.

In most cases, the easiest way to tune buffering is to eliminate the pushback and allow it to buffer the data to disk as necessary. Setting APT_BUFFER_FREE_RUN=N or setting **Buffer Free Run** in the Output page Advanced tab on a particular stage will do this. A buffer will read $N * \text{max_memory}$ (3 MB by default) bytes before beginning to push back on the upstream. If there is enough disk space to buffer large amounts of data, this will usually fix any egregious slowdown issues caused by the buffer operator.

If there is a significant amount of memory available on the machine, increasing the maximum in-memory buffer size is likely to be very useful if buffering is causing any disk I/O. Setting the APT_BUFFER_MAXIMUM_MEMORY environment variable or Maximum memory buffer size on the Output page Advanced tab on a particular stage will do this. It defaults to 3145728 (3 MB).

For systems where small to medium bursts of I/O are not desirable, the 1 MB write to disk size chunk size might be too small. The environment variable APT_BUFFER_DISK_WRITE_INCREMENT or **Disk write increment** on the Output page Advanced tab on a particular stage controls this and defaults to 1048576 (1 MB). This setting might not exceed $\text{max_memory} * 2/3$.

Finally, in a situation where a large, fixed buffer is needed within the flow, setting **Queue upper bound** on the Output page Advanced tab (no environment variable exists) can be set equal to max_memory to force a buffer of exactly max_memory bytes. Such a buffer will block an upstream operator (until data is read by the downstream operator) once its buffer has been filled, so this setting should be used with extreme caution. This setting is rarely, if ever, necessary to achieve good performance, but might be useful in an attempt to squeeze every last byte of performance out of the system where it is desirable to eliminate buffering to disk entirely. No environment variable is available for this flag, and therefore this can only be set at the individual stage level.

Platform specific tuning

HP-UX

HP-UX has a limitation when running in 32-bit mode, which limits memory mapped I/O to 2 GB per machine. This can be an issue when dealing with large lookups. The Memory Windows® options can provide a work around for this memory limitation. Product Support can provide this document on request.

AIX

If you are running WebSphere DataStage Enterprise Edition on an RS/6000® SP™ or a network of workstations, verify your setting of the network parameter `thewall`.

Disk space requirements of post-release 7.0.1 data sets

Some parallel data sets generated with WebSphere DataStage 7.0.1 and later releases require more disk space when the columns are of type VarChar when compared to 7.0. This is due to changes added for performance improvements for bounded length VarChars in 7.0.1.

The preferred solution is to use unbounded length VarChars (don't set any length) for columns where the maximum length is rarely used. Alternatively, you can set the environment variable, `APT_OLD_BOUNDDED_LENGTH`, but this is not recommended, as it leads to performance degradation.

Chapter 4. Link buffering

These topics contain an in-depth description of when and how WebSphereDataStage buffers data within a job, and how you can change the automatic settings if required.

WebSphere DataStage automatically performs buffering on the links of certain stages. This is primarily intended to prevent deadlock situations arising (where one stage is unable to read its input because a previous stage in the job is blocked from writing to its output).

Deadlock situations can occur where you have a fork-join in your job. This is where a stage has two output links whose data paths are joined together later in the job. The situation can arise where all the stages in the flow are waiting for each other to read or write, so none of them can proceed. No error or warning message is output for deadlock; your job will be in a state where it will wait forever for an input.

WebSphere DataStage automatically inserts buffering into job flows containing fork-joins where deadlock situations might arise. In most circumstances you should not need to alter the default buffering implemented by WebSphere DataStage. However you might want to insert buffers in other places in your flow (to smooth and improve performance) or you might want to explicitly control the buffers inserted to avoid deadlocks. WebSphere DataStage allows you to do this, but use caution when altering the default buffer settings.

Buffering assumptions

This section describes buffering in more detail, and in particular the design assumptions underlying its default behavior.

Buffering in WebSphere DataStage is designed around the following assumptions:

- Buffering is primarily intended to remove the potential for deadlock in flows with fork-join structure.
- Throughput is preferable to overhead. The goal of the WebSphere DataStage buffering mechanism is to keep the flow moving with as little memory and disk usage as possible. Ideally, data should simply stream through the data flow and rarely land to disk. Upstream operators should tend to wait for downstream operators to consume their input before producing new data records.
- Stages in general are designed so that on each link between stages data is being read and written whenever possible. While buffering is designed to tolerate occasional backlog on specific links due to one operator getting ahead of another, it is assumed that operators are at least occasionally attempting to read and write data on each link.

Buffering is implemented by the automatic insertion of a hidden buffer operator on links between stages. The buffer operator attempts to match the rates of its input and output. When no data is being read from the buffer operator by the downstream stage, the buffer operator tries to throttle back incoming data from the upstream stage to avoid letting the buffer grow so large that it must be written out to disk.

The goal is to avoid situations where data will have to be moved to and from disk needlessly, especially in situations where the consumer cannot process data at the same rate as the producer (for example, due to a more complex calculation).

Because the buffer operator wants to keep the flow moving with low overhead, it is assumed in general that it is better to cause the producing stage to wait before writing new records, rather than allow the buffer operator to consume resources.

Controlling buffering

WebSphere DataStage offers two ways of controlling the operation of buffering: you can use environment variables to control buffering on all links of all stages in all jobs, or you can make individual settings on the links of particular stages via the stage editors.

Buffering policy

You can set this via the APT_BUFFERING_POLICY environment variable, or via the **Buffering mode** field on the Inputs or Outputs page Advanced tab for individual stage editors.

The environment variable has the following possible values:

- AUTOMATIC_BUFFERING. Buffer a data set only if necessary to prevent a dataflow deadlock. This setting is the default if you do not define the environment variable.
- FORCE_BUFFERING. Unconditionally buffer all links.
- NO_BUFFERING. Do not buffer links. This setting can cause deadlock if used inappropriately.

The possible settings for the **Buffering mode** field are:

- (Default). This will take whatever the default settings are as specified by the environment variables (this will be Auto buffer unless you have explicitly changed the value of the APT_BUFFERING_POLICY environment variable).
- Auto buffer. Buffer data only if necessary to prevent a dataflow deadlock situation.
- Buffer. This will unconditionally buffer all data output from/input to this stage.
- No buffer. Do not buffer data under any circumstances. This could potentially lead to deadlock situations if not used carefully.

Overriding default buffering behavior

Since the default value of APT_BUFFERING_POLICY is AUTOMATIC_BUFFERING, the default action of WebSphere DataStage is to buffer a link only if required to avoid deadlock. You can, however, override the default buffering operation in your job.

For example, some operators read an entire input data set before outputting a single record. The Sort stage is an example of this. Before a sort operator can output a single record, it must read all input to determine the first output record. Therefore, these operators internally buffer the entire output data set, eliminating the need of the default buffering mechanism. For this reason, WebSphere DataStage never inserts a buffer on the output of a sort.

You might also develop a customized stage that does not require its output to be buffered, or you might want to change the size parameters of the WebSphere DataStage buffering mechanism. In this case, you can set the various buffering parameters. These can be set via environment variables or via the Advanced tab on the Inputs or Outputs page for individual stage editors. What you set in the Outputs page Advanced tab will automatically appear in the Inputs page Advanced tab of the stage at the other end of the link (and vice versa)

The available environment variables are as follows:

- APT_BUFFER_MAXIMUM_MEMORY. Specifies the maximum amount of virtual memory, in bytes, used per buffer. The default size is 3145728 (3 MB). If your step requires 10 buffers, each processing node would use a maximum of 30 MB of virtual memory for buffering. If WebSphere DataStage has to buffer more data than Maximum memory buffer size, the data is written to disk.
- APT_BUFFER_DISK_WRITE_INCREMENT. Sets the size, in bytes, of blocks of data being moved to/from disk by the buffering operator. The default is 1048576 (1 MByte.) Adjusting this value trades amount of disk access against throughput for small amounts of data. Increasing the block size reduces

disk access, but might decrease performance when data is being read/written in smaller units. Decreasing the block size increases throughput, but might increase the amount of disk access.

- **APT_BUFFER_FREE_RUN**. Specifies how much of the available in-memory buffer to consume before the buffer offers resistance to any new data being written to it, as a percentage of Maximum memory buffer size. When the amount of buffered data is less than the Buffer free run percentage, input data is accepted immediately by the buffer. After that point, the buffer does not immediately accept incoming data; it offers resistance to the incoming data by first trying to output data already in the buffer before accepting any new input. In this way, the buffering mechanism avoids buffering excessive amounts of data and can also avoid unnecessary disk I/O. The default percentage is 0.5 (50% of Maximum memory buffer size or by default 1.5 MB). You must set Buffer free run greater than 0.0. Typical values are between 0.0 and 1.0. You can set Buffer free run to a value greater than 1.0. In this case, the buffer continues to store data up to the indicated multiple of Maximum memory buffer size before writing data to disk.

The available settings in the Input or Outputs pageAdvanced tab of stage editors are:

- Maximum memory buffer size (bytes). Specifies the maximum amount of virtual memory, in bytes, used per buffer. The default size is 3145728 (3 MB).
- Buffer free run (percent). Specifies how much of the available in-memory buffer to consume before the buffer resists. This is expressed as a percentage of Maximum memory buffer size. When the amount of data in the buffer is less than this value, new data is accepted automatically. When the data exceeds it, the buffer first tries to write some of the data it contains before accepting more. The default value is 50% of the Maximum memory buffer size. You can set it to greater than 100%, in which case the buffer continues to store data up to the indicated multiple of Maximum memory buffer size before writing to disk.
- Queue upper bound size (bytes). Specifies the maximum amount of data buffered at any time using both memory and disk. The default value is zero, meaning that the buffer size is limited only by the available disk space as specified in the configuration file (resource scratchdisk). If you set Queue upper bound size (bytes) to a non-zero value, the amount of data stored in the buffer will not exceed this value (in bytes) plus one block (where the data stored in a block cannot exceed 32 KB).

If you set Queue upper bound size to a value equal to or slightly less than Maximum memory buffer size, and set Buffer free run to 1.0, you will create a finite capacity buffer that will not write to disk. However, the size of the buffer is limited by the virtual memory of your system and you can create deadlock if the buffer becomes full.

(Note that there is no environment variable for Queue upper bound size).

- Disk write increment (bytes). Sets the size, in bytes, of blocks of data being moved to/from disk by the buffering operator. The default is 1048576 (1 MB). Adjusting this value trades amount of disk access against throughput for small amounts of data. Increasing the block size reduces disk access, but might decrease performance when data is being read/written in smaller units. Decreasing the block size increases throughput, but might increase the amount of disk access.

Operators with special buffering requirements

If you have built a custom stage that is designed to not consume one of its inputs, for example to buffer all records before proceeding, the default behavior of the buffer operator can end up being a performance bottleneck, slowing down the job. This section describes how to fix this problem.

Although the buffer operator is not designed for buffering an entire data set as output by a stage, it is capable of doing so assuming sufficient memory or disk space is available to buffer the data. To achieve this you need to adjust the settings described above appropriately, based on your job. You might be able to solve your problem by modifying one buffering property, the Buffer free run setting. This controls the amount of memory/disk space that the buffer operator is allowed to consume before it begins to push back on the upstream operator.

The default setting for Buffer free run is 0.5 for the environment variable, (50% for **Buffer free run** on the Advanced tab), which means that half of the internal memory buffer can be consumed before pushback occurs. This biases the buffer operator to avoid allowing buffered data to be written to disk.

If your stage needs to buffer large data sets, we recommend that you initially set Buffer free run to a very large value such as 1000, and then adjust according to the needs of your application. This will allow the buffer operator to freely use both memory *and* disk space in order to accept incoming data without pushback.

We recommend that you set the Buffer free run property only for those links between stages that require a non-default value; this means altering the setting on the Inputs page or Outputs page Advanced tab of the stage editors, not the environment variable.

Chapter 5. Specifying your own parallel stages

In addition to the wide range of parallel stage types available, WebSphere DataStage allows you to define your own stage types, which you can then use in parallel jobs.

There are three different types of stage that you can define:

- Custom. This allows knowledgeable Orchestrate® users to specify an Orchestrate operator as a WebSphere DataStage stage. This is then available to use in WebSphere DataStage Parallel jobs.
- Build. This allows you to design and build your own operator as a stage to be included in WebSphere DataStage Parallel Jobs.
- Wrapped. This allows you to specify a UNIX command to be executed by a WebSphere DataStage stage. You define a wrapper file that in turn defines arguments for the UNIX command and inputs and outputs.

WebSphere DataStage Designer provides an interface that allows you to define a new WebSphere DataStage Parallel job stage of any of these types. This interface is also available from the repository tree of the WebSphere DataStage Designer. This topic describes how to use this interface.

Defining custom stages

To define a custom stage type:

1. Do one of:
 - a. Choose **File** → **New** from the Designer menu. The New dialog box appears.
 - b. Open the **Stage Type** folder and select the **Parallel Custom Stage Type** icon.
 - c. Click **OK**. The Stage Type dialog box appears, with the General page on top.
Or:
 - d. Select a folder in the repository tree.
 - e. Choose **New** → **Other Parallel Stage Custom** from the shortcut menu. The Stage Type dialog box appears, with the General page on top.
2. Fill in the fields on the General page as follows:
 - **Stage type name**. This is the name that the stage will be known by to WebSphere DataStage. Avoid using the same name as existing stages.
 - **Parallel Stage type**. This indicates the type of new Parallel job stage you are defining (Custom, Build, or Wrapped). You cannot change this setting.
 - **Execution Mode**. Choose the execution mode. This is the mode that will appear in the Advanced tab on the stage editor. You can override this mode for individual instances of the stage as required, unless you select Parallel only or Sequential only. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the execution mode.
 - **Mapping**. Choose whether the stage has a Mapping tab or not. A Mapping tab enables the user of the stage to specify how output columns are derived from the data produced by the stage. Choose **None** to specify that output mapping is not performed, choose **Default** to accept the default setting that WebSphere DataStage uses.
 - **Preserve Partitioning**. Choose the default setting of the Preserve Partitioning flag. This is the setting that will appear in the Advanced tab on the stage editor. You can override this setting for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the preserve partitioning flag.

- **Partitioning.** Choose the default partitioning method for the stage. This is the method that will appear in the Inputs page Partitioning tab of the stage editor. You can override this method for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the partitioning methods.
 - **Collecting.** Choose the default collection method for the stage. This is the method that will appear in the Inputs page Partitioning tab of the stage editor. You can override this method for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the collection methods.
 - **Operator.** Enter the name of the Orchestrate operator that you want the stage to invoke.
 - **Short Description.** Optionally enter a short description of the stage.
 - **Long Description.** Optionally enter a long description of the stage.
3. Go to the Links page and specify information about the links allowed to and from the stage you are defining.

Use this to specify the minimum and maximum number of input and output links that your custom stage can have, and to enable the ViewData feature for target data (you cannot enable target ViewData if your stage has any output links). When the stage is used in a job design, a **ViewData** button appears on the Input page, which allows you to view the data on the actual data target (provided some has been written there).

In order to use the target ViewData feature, you have to specify an Orchestrate operator to read the data back from the target. This will usually be different to the operator that the stage has used to write the data (that is, the operator defined in the **Operator** field of the General page). Specify the reading operator and associated arguments in the **Operator** and **Options** fields.

If you enable target ViewData, a further field appears in the Properties grid, called ViewData.

4. Go to the Creator page and optionally specify information about the stage you are creating. We recommend that you assign a version number to the stage so you can keep track of any subsequent changes.

You can specify that the actual stage will use a custom GUI by entering the ProgID for a custom GUI in the **Custom GUI Prog ID** field.

You can also specify that the stage has its own icon. You need to supply a 16 x 16 bit bitmap and a 32 x 32 bit bitmap to be displayed in various places in the WebSphere DataStage user interface. Click the 16 x 16 Bitmap button and browse for the smaller bitmap file. Click the 32 x 32 Bitmap button and browse for the large bitmap file. Note that bitmaps with 32-bit color are not supported. Click the **Reset Bitmap Info** button to revert to using the default WebSphere DataStage icon for this stage.

5. Go to the Properties page. This allows you to specify the options that the Orchestrate operator requires as properties that appear in the Stage Properties tab. For custom stages the Properties tab always appears under the Stage page.
6. Fill in the fields as follows:

- **Property name.** The name of the property.
- **Data type.** The data type of the property. Choose from:

Boolean

Float

Integer

String

Pathname

List

Input Column

Output Column

If you choose **Input Column** or **Output Column**, when the stage is included in a job a drop-down list will offer a choice of the defined input or output columns.

If you choose **list** you should open the Extended Properties dialog box from the grid shortcut menu to specify what appears in the list.

- **Prompt.** The name of the property that will be displayed on the Properties tab of the stage editor.
- **Default Value.** The value the option will take if no other is specified.
- **Required.** Set this to True if the property is mandatory.
- **Repeats.** Set this true if the property repeats (that is, you can have multiple instances of it).
- **Use Quoting.** Specify whether the property will have quotes added when it is passed to the Orchestrate operator.
- **Conversion.** Specifies the type of property as follows:
 - Name.** The name of the property will be passed to the operator as the option value. This will normally be a hidden property, that is, not visible in the stage editor.
 - Name Value.** The name of the property will be passed to the operator as the option name, and any value specified in the stage editor is passed as the value.
 - Value.** The value for the property specified in the stage editor is passed to the operator as the option name. Typically used to group operator options that are mutually exclusive.
 - Value only.** The value for the property specified in the stage editor is passed as it is.

Input Schema. Specifies that the property will contain a schema string whose contents are populated from the Input page **Columns** tab.

Output Schema. Specifies that the property will contain a schema string whose contents are populated from the Output page **Columns** tab.

None. This allows the creation of properties that do not generate any OSH, but can be used for conditions on other properties (for example, for use in a situation where you have mutually exclusive properties, but at least one of them must be specified).

- Schema properties require format options. Select this check box to specify that the stage being specified will have a Format tab.

If you have enabled target ViewData on the Links page, the following property is also displayed:

- **ViewData.** Select Yes to indicate that the value of this property should be used when viewing data. For example, if this property specifies a file to write to when the stage is used in a job design, the value of this property will be used to read the data back if ViewData is used in the stage.

If you select a conversion type of Input Schema or Output Schema, you should note the following:

- **Data Type** is set to String.
- **Required** is set to Yes.
- The property is marked as hidden and will not appear on the Properties page when the custom stage is used in a job design.

If your stage can have multiple input or output links there would be a Input Schema property or Output Schema property per-link.

When the stage is used in a job design, the property will contain the following OSH for each input or output link:

`-property_name record {format_properties} (column_definition {format_properties}; ...)`

Where:

- *property_name* is the name of the property (usually 'schema')
- *format_properties* are formatting information supplied on the Format page (if the stage has one).
- there is one *column_definition* for each column defined in the Columns tab for that link. The *format_props* in this case refers to per-column format information specified in the Edit Column Meta Data dialog box.

Schema properties are mutually exclusive with schema file properties. If your custom stage supports both, you should use the Extended Properties dialog box to specify a condition of "schemafile= " for the schema property. The schema property is then only valid provided the schema file property is blank (or does not exist).

7. If you want to specify a list property, or otherwise control how properties are handled by your stage, choose **Extended Properties** from the Properties grid shortcut menu to open the Extended Properties dialog box.

The settings you use depend on the type of property you are specifying:

- Specify a category to have the property appear under this category in the stage editor. By default all properties appear in the Options category.
 - Specify that the property will be hidden and not appear in the stage editor. This is primarily intended to support the case where the underlying operator needs to know the JobName. This can be passed using a mandatory String property with a default value that uses a DS Macro. However, to prevent the user from changing the value, the property needs to be hidden.
 - If you are specifying a List category, specify the possible values for list members in the List Value field.
 - If the property is to be a dependent of another property, select the parent property in the Parents field.
 - Specify an expression in the Template field to have the actual value of the property generated at compile time. It is usually based on values in other properties and columns.
 - Specify an expression in the Conditions field to indicate that the property is only valid if the conditions are met. The specification of this property is a bar '|' separated list of conditions that are AND'ed together. For example, if the specification was a=b | c!=d, then this property would only be valid (and therefore only available in the GUI) when property a is equal to b, and property c is not equal to d.
8. If your custom stage will create columns, go to the Mapping Additions page. It contains a grid that allows for the specification of columns created by the stage. You can also specify that column details are filled in from properties supplied when the stage is used in a job design, allowing for dynamic specification of columns.

The grid contains the following fields:

- **Column name.** The name of the column created by the stage. You can specify the name of a property you specified on the Property page of the dialog box to dynamically allocate the column name. Specify this in the form #property_name#, the created column will then take the value of this property, as specified at design time, as the name of the created column.
- **Parallel type.** The type of the column (this is the underlying data type, not the SQL data type). Again you can specify the name of a property you specified on the Property page of the dialog box to dynamically allocate the column type. Specify this in the form #property_name#, the created column will then take the value of this property, as specified at design time, as the type of the created column. (Note that you cannot use a repeatable property to dynamically allocate a column type in this way.)
- **Nullable.** Choose **Yes** or **No** to indicate whether the created column can contain a null.
- **Conditions.** Allows you to enter an expression specifying the conditions under which the column will be created. This could, for example, depend on the setting of one of the properties specified in the Property page.

You can propagate the values of the Conditions fields to other columns if required. Do this by selecting the columns you want to propagate to, then right-clicking in the source **Conditions** field and choosing **Propagate** from the shortcut menu. A dialog box asks you to confirm that you want to propagate the conditions to all columns.

9. Click **OK** when you are happy with your custom stage definition. The Save As dialog box appears.
10. Select the folder in the repository tree where you want to store the stage type and click **OK**.

Defining custom stages

To define a custom stage type:

1. Do one of:
 - a. Choose **File** → **New** from the Designer menu. The New dialog box appears.
 - b. Open the **Stage Type** folder and select the **Parallel Custom Stage Type** icon.
 - c. Click **OK**. The Stage Type dialog box appears, with the General page on top.
Or:
 - d. Select a folder in the repository tree.
 - e. Choose **New** → **Other Parallel Stage Custom** from the shortcut menu. The Stage Type dialog box appears, with the General page on top.
2. Fill in the fields on the General page as follows:
 - **Stage type name.** This is the name that the stage will be known by to WebSphere DataStage. Avoid using the same name as existing stages.
 - **Parallel Stage type.** This indicates the type of new Parallel job stage you are defining (Custom, Build, or Wrapped). You cannot change this setting.
 - **Execution Mode.** Choose the execution mode. This is the mode that will appear in the Advanced tab on the stage editor. You can override this mode for individual instances of the stage as required, unless you select Parallel only or Sequential only. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the execution mode.
 - **Mapping.** Choose whether the stage has a Mapping tab or not. A Mapping tab enables the user of the stage to specify how output columns are derived from the data produced by the stage. Choose **None** to specify that output mapping is not performed, choose **Default** to accept the default setting that WebSphere DataStage uses.
 - **Preserve Partitioning.** Choose the default setting of the Preserve Partitioning flag. This is the setting that will appear in the Advanced tab on the stage editor. You can override this setting for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the preserve partitioning flag.
 - **Partitioning.** Choose the default partitioning method for the stage. This is the method that will appear in the Inputs page Partitioning tab of the stage editor. You can override this method for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the partitioning methods.
 - **Collecting.** Choose the default collection method for the stage. This is the method that will appear in the Inputs page Partitioning tab of the stage editor. You can override this method for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the collection methods.
 - **Operator.** Enter the name of the Orchestrate operator that you want the stage to invoke.
 - **Short Description.** Optionally enter a short description of the stage.
 - **Long Description.** Optionally enter a long description of the stage.
3. Go to the Links page and specify information about the links allowed to and from the stage you are defining.

Use this to specify the minimum and maximum number of input and output links that your custom stage can have, and to enable the ViewData feature for target data (you cannot enable target ViewData if your stage has any output links). When the stage is used in a job design, a **ViewData** button appears on the Input page, which allows you to view the data on the actual data target (provided some has been written there).

In order to use the target ViewData feature, you have to specify an Orchestrate operator to read the data back from the target. This will usually be different to the operator that the stage has used to write the data (that is, the operator defined in the **Operator** field of the General page). Specify the reading operator and associated arguments in the **Operator** and **Options** fields.

- If you enable target ViewData, a further field appears in the Properties grid, called ViewData.
4. Go to the Creator page and optionally specify information about the stage you are creating. We recommend that you assign a version number to the stage so you can keep track of any subsequent changes.
- You can specify that the actual stage will use a custom GUI by entering the ProgID for a custom GUI in the **Custom GUI Prog ID** field.
- You can also specify that the stage has its own icon. You need to supply a 16 x 16 bit bitmap and a 32 x 32 bit bitmap to be displayed in various places in the WebSphere DataStage user interface. Click the 16 x 16 Bitmap button and browse for the smaller bitmap file. Click the 32 x 32 Bitmap button and browse for the large bitmap file. Note that bitmaps with 32-bit color are not supported. Click the **Reset Bitmap Info** button to revert to using the default WebSphere DataStage icon for this stage.
5. Go to the Properties page. This allows you to specify the options that the Orchestrate operator requires as properties that appear in the Stage Properties tab. For custom stages the Properties tab always appears under the Stage page.
 6. Fill in the fields as follows:

- **Property name.** The name of the property.
- **Data type.** The data type of the property. Choose from:

Boolean

Float

Integer

String

Pathname

List

Input Column

Output Column

If you choose **Input Column** or **Output Column**, when the stage is included in a job a drop-down list will offer a choice of the defined input or output columns.

If you choose **List** you should open the Extended Properties dialog box from the grid shortcut menu to specify what appears in the list.

- **Prompt.** The name of the property that will be displayed on the Properties tab of the stage editor.
- **Default Value.** The value the option will take if no other is specified.

- **Required.** Set this to True if the property is mandatory.

- **Repeats.** Set this true if the property repeats (that is, you can have multiple instances of it).

- **Use Quoting.** Specify whether the property will have quotes added when it is passed to the Orchestrate operator.

- **Conversion.** Specifies the type of property as follows:

-Name. The name of the property will be passed to the operator as the option value. This will normally be a hidden property, that is, not visible in the stage editor.

-Name Value. The name of the property will be passed to the operator as the option name, and any value specified in the stage editor is passed as the value.

-Value. The value for the property specified in the stage editor is passed to the operator as the option name. Typically used to group operator options that are mutually exclusive.

Value only. The value for the property specified in the stage editor is passed as it is.

Input Schema. Specifies that the property will contain a schema string whose contents are populated from the Input page **Columns** tab.

Output Schema. Specifies that the property will contain a schema string whose contents are populated from the Output page **Columns** tab.

None. This allows the creation of properties that do not generate any osh, but can be used for conditions on other properties (for example, for use in a situation where you have mutually exclusive properties, but at least one of them must be specified).

- Schema properties require format options. Select this check box to specify that the stage being specified will have a Format tab.

If you have enabled target ViewData on the Links page, the following property is also displayed:

- **ViewData.** Select Yes to indicate that the value of this property should be used when viewing data. For example, if this property specifies a file to write to when the stage is used in a job design, the value of this property will be used to read the data back if ViewData is used in the stage.

If you select a conversion type of Input Schema or Output Schema, you should note the following:

- **Data Type** is set to String.
- **Required** is set to Yes.
- The property is marked as hidden and will not appear on the Properties page when the custom stage is used in a job design.

If your stage can have multiple input or output links there would be a Input Schema property or Output Schema property per-link.

When the stage is used in a job design, the property will contain the following OSH for each input or output link:

```
-property_name record {format_properties} ( column_definition {format_properties}; ...)
```

Where:

- *property_name* is the name of the property (usually `schema`)
- *format_properties* are formatting information supplied on the Format page (if the stage has one).
- there is one *column_definition* for each column defined in the Columns tab for that link. The *format_props* in this case refers to per-column format information specified in the Edit Column Meta Data dialog box.

Schema properties are mutually exclusive with schema file properties. If your custom stage supports both, you should use the Extended Properties dialog box to specify a condition of "schemafile= " for the schema property. The schema property is then only valid provided the schema file property is blank (or does not exist).

7. If you want to specify a list property, or otherwise control how properties are handled by your stage, choose **Extended Properties** from the Properties grid shortcut menu to open the Extended Properties dialog box.

The settings you use depend on the type of property you are specifying:

- Specify a category to have the property appear under this category in the stage editor. By default all properties appear in the Options category.
- Specify that the property will be hidden and not appear in the stage editor. This is primarily intended to support the case where the underlying operator needs to know the JobName. This can be passed using a mandatory String property with a default value that uses a DS Macro. However, to prevent the user from changing the value, the property needs to be hidden.
- If you are specifying a List category, specify the possible values for list members in the List Value field.
- If the property is to be a dependent of another property, select the parent property in the Parents field.
- Specify an expression in the Template field to have the actual value of the property generated at compile time. It is usually based on values in other properties and columns.
- Specify an expression in the Conditions field to indicate that the property is only valid if the conditions are met. The specification of this property is a bar ' | ' separated list of conditions that

are AND'ed together. For example, if the specification was a=b | c!=d, then this property would only be valid (and therefore only available in the GUI) when property a is equal to b, and property c is not equal to d.

8. If your custom stage will create columns, go to the Mapping Additions page. It contains a grid that allows for the specification of columns created by the stage. You can also specify that column details are filled in from properties supplied when the stage is used in a job design, allowing for dynamic specification of columns.

The grid contains the following fields:

- **Column name.** The name of the column created by the stage. You can specify the name of a property you specified on the Property page of the dialog box to dynamically allocate the column name. Specify this in the form #property_name#, the created column will then take the value of this property, as specified at design time, as the name of the created column.
- **Parallel type.** The type of the column (this is the underlying data type, not the SQL data type). Again you can specify the name of a property you specified on the Property page of the dialog box to dynamically allocate the column type. Specify this in the form #property_name#, the created column will then take the value of this property, as specified at design time, as the type of the created column. (Note that you cannot use a repeatable property to dynamically allocate a column type in this way.)
- **Nullable.** Choose **Yes** or **No** to indicate whether the created column can contain a null.
- **Conditions.** Allows you to enter an expression specifying the conditions under which the column will be created. This could, for example, depend on the setting of one of the properties specified in the Property page.

You can propagate the values of the Conditions fields to other columns if required. Do this by selecting the columns you want to propagate to, then right-clicking in the source **Conditions** field and choosing **Propagate** from the shortcut menu. A dialog box asks you to confirm that you want to propagate the conditions to all columns.

9. Click **OK** when you are happy with your custom stage definition. The Save As dialog box appears.
10. Select the folder in the repository tree where you want to store the stage type and click **OK**.

Defining build stages

You define a Build stage to enable you to provide a custom operator that can be executed from a parallel job stage. The stage will be available to all jobs in the project in which the stage was defined. You can make it available to other projects using the WebSphere DataStage Export facilities. The stage is automatically added to the job palette.

When defining a Build stage you provide the following information:

- Description of the data that will be input to the stage.
- Whether records are transferred from input to output. A transfer copies the input record to the output buffer. If you specify auto transfer, the operator transfers the input record to the output record immediately after execution of the per record code. The code can still access data in the output buffer until it is actually written.
- Any definitions and header file information that needs to be included.
- Code that is executed at the beginning of the stage (before any records are processed).
- Code that is executed at the end of the stage (after all records have been processed).
- Code that is executed every time the stage processes a record.
- Compilation and build details for actually building the stage.

Note that the custom operator that your build stage executes must have at least one input data set and one output data set.

The Code for the Build stage is specified in C++. There are a number of macros available to make the job of coding simpler (see "Build Stage Macros". There are also a number of header files available containing many useful functions, see Appendix A.

When you have specified the information, and request that the stage is generated, WebSphere DataStage generates a number of files and then compiles these to build an operator which the stage executes. The generated files include:

- Header files (ending in .h)
- Source files (ending in .c)
- Object files (ending in .so)

The following shows a build stage in diagrammatic form:

To define a Build stage:

1. Do one of:
 - a. Choose **File** → **New** from the Designer menu. The New dialog box appears.
 - b. Open the **Stage Type** folder and select the **Parallel Build Stage Type** icon.
 - c. Click **OK**. The Stage Type dialog box appears, with the General page on top.
Or:
 - d. Select a folder in the repository tree.
 - e. Choose **New** → **Other Parallel Stage Custom** from the shortcut menu. The Stage Type dialog box appears, with the General page on top.
2. Fill in the fields on the General page as follows:
 - **Stage type name.** This is the name that the stage will be known by to WebSphere DataStage. Avoid using the same name as existing stages.
 - **Class Name.** The name of the C++ class. By default this takes the name of the stage type.
 - **Parallel Stage type.** This indicates the type of new parallel job stage you are defining (Custom, Build, or Wrapped). You cannot change this setting.
 - **Execution mode.** Choose the default execution mode. This is the mode that will appear in the Advanced tab on the stage editor. You can override this mode for individual instances of the stage as required, unless you select **Parallel only** or **Sequential only**. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the execution mode.
 - **Preserve Partitioning.** This shows the default setting of the Preserve Partitioning flag, which you cannot change in a Build stage. This is the setting that will appear in the Advanced tab on the stage editor. You can override this setting for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the preserve partitioning flag.
 - **Partitioning.** This shows the default partitioning method, which you cannot change in a Build stage. This is the method that will appear in the Inputs Page Partitioning tab of the stage editor. You can override this method for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the partitioning methods.
 - **Collecting.** This shows the default collection method, which you cannot change in a Build stage. This is the method that will appear in the Inputs Page Partitioning tab of the stage editor. You can override this method for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the collection methods.
 - **Operator.** The name of the operator that your code is defining and which will be executed by the WebSphere DataStage stage. By default this takes the name of the stage type.
 - **Short Description.** Optionally enter a short description of the stage.
 - **Long Description.** Optionally enter a long description of the stage.

3. Go to the Creator page and optionally specify information about the stage you are creating. We recommend that you assign a release number to the stage so you can keep track of any subsequent changes.

You can specify that the actual stage will use a custom GUI by entering the ProgID for a custom GUI in the **Custom GUI Prog ID** field.

You can also specify that the stage has its own icon. You need to supply a 16 x 16 bit bitmap and a 32 x 32 bit bitmap to be displayed in various places in the WebSphere DataStage user interface. Click the 16 x 16 Bitmap button and browse for the smaller bitmap file. Click the 32 x 32 Bitmap button and browse for the large bitmap file. Note that bitmaps with 32-bit color are not supported. Click the **Reset Bitmap Info** button to revert to using the default WebSphere DataStage icon for this stage.

4. Go to the Properties page. This allows you to specify the options that the Build stage requires as properties that appear in the Stage Properties tab. For custom stages the Properties tab always appears under the Stage page.

Fill in the fields as follows:

- Property name. The name of the property. This will be passed to the operator you are defining as an option, prefixed with `-' and followed by the value selected in the Properties tab of the stage editor.
- Data type. The data type of the property. Choose from:

Boolean

Float

Integer

String

Pathname

List

Input Column

Output Column

If you choose Input Column or Output Column, when the stage is included in a job a drop-down list will offer a choice of the defined input or output columns.

If you choose list you should open the Extended Properties dialog box from the grid shortcut menu to specify what appears in the list.

- Prompt. The name of the property that will be displayed on the Properties tab of the stage editor.
- Default Value. The value the option will take if no other is specified.
- Required. Set this to True if the property is mandatory.
- Conversion. Specifies the type of property as follows:
 - Name. The name of the property will be passed to the operator as the option value. This will normally be a hidden property, that is, not visible in the stage editor.
 - Name Value. The name of the property will be passed to the operator as the option name, and any value specified in the stage editor is passed as the value.
 - Value. The value for the property specified in the stage editor is passed to the operator as the option name. Typically used to group operator options that are mutually exclusive.
 - Value only. The value for the property specified in the stage editor is passed as it is.

5. If you want to specify a list property, or otherwise control how properties are handled by your stage, choose **Extended Properties** from the Properties grid shortcut menu to open the Extended Properties dialog box.

The settings you use depend on the type of property you are specifying:

- Specify a category to have the property appear under this category in the stage editor. By default all properties appear in the Options category.
- If you are specifying a List category, specify the possible values for list members in the List Value field.

- If the property is to be a dependent of another property, select the parent property in the Parents field.
- Specify an expression in the Template field to have the actual value of the property generated at compile time. It is usually based on values in other properties and columns.
- Specify an expression in the Conditions field to indicate that the property is only valid if the conditions are met. The specification of this property is a bar ‘|’ separated list of conditions that are AND’ed together. For example, if the specification was $a=b \mid c=d$, then this property would only be valid (and therefore only available in the GUI) when property a is equal to b, and property c is not equal to d.

Click OK when you are happy with the extended properties.

6. Click on the Build page. The tabs here allow you to define the actual operation that the stage will perform.

The Interfaces tab enable you to specify details about inputs to and outputs from the stage, and about automatic transfer of records from input to output. You specify port details, a port being where a link connects to the stage. You need a port for each possible input link to the stage, and a port for each possible output link from the stage.

You provide the following information on the Input sub-tab:

- Port Name. Optional name for the port. The default names for the ports are in0, in1, in2 You can refer to them in the code using either the default name or the name you have specified.
- Alias. Where the port name contains non-ascii characters, you can give it an alias in this column (this is only available where NLS is enabled).
- AutoRead. This defaults to True which means the stage will automatically read records from the port. Otherwise you explicitly control read operations in the code.
- Table Name. Specify a table definition in the WebSphere DataStage Repository which describes the meta data for the port. You can browse for a table definition by choosing Select Table from the menu that appears when you click the browse button. You can also view the schema corresponding to this table definition by choosing View Schema from the same menu. You do not have to supply a Table Name. If any of the columns in your table definition have names that contain non-ascii characters, you should choose Column Aliases from the menu. The Build Column Aliases dialog box appears. This lists the columns that require an alias and let you specify one.
- RCP. Choose True if runtime column propagation is allowed for inputs to this port. Defaults to False. You do not need to set this if you are using the automatic transfer facility.

You provide the following information on the Output sub-tab:

- Port Name. Optional name for the port. The default names for the links are out0, out1, out2 You can refer to them in the code using either the default name or the name you have specified.
- Alias. Where the port name contains non-ascii characters, you can give it an alias in this column.
- AutoWrite. This defaults to True which means the stage will automatically write records to the port. Otherwise you explicitly control write operations in the code. Once records are written, the code can no longer access them.
- Table Name. Specify a table definition in the WebSphere DataStage Repository which describes the meta data for the port. You can browse for a table definition. You do not have to supply a Table Name. A shortcut menu accessed from the browse button offers a choice of Clear Table Name, Select Table, Create Table, View Schema, and Column Aliases. The use of these is as described for the Input sub-tab.
- RCP. Choose True if runtime column propagation is allowed for outputs from this port. Defaults to False. You do not need to set this if you are using the automatic transfer facility.

The Transfer sub-tab allows you to connect an input buffer to an output buffer such that records will be automatically transferred from input to output. You can also disable automatic transfer, in which case you have to explicitly transfer data in the code. Transferred data sits in an output buffer and can still be accessed and altered by the code until it is actually written to the port.

You provide the following information on the Transfer tab:

- Input. Select the input port to connect to the buffer from the drop-down list. If you have specified an alias, this will be displayed here.
- Output. Select the output port to transfer input records from the output buffer to from the drop-down list. If you have specified an alias, this will be displayed here.
- Auto Transfer. This defaults to False, which means that you have to include code which manages the transfer. Set to True to have the transfer carried out automatically.
- Separate. This is False by default, which means this transfer will be combined with other transfers to the same port. Set to True to specify that the transfer should be separate from other transfers.

The Logic tab is where you specify the actual code that the stage executes.

The Definitions sub-tab allows you to specify variables, include header files, and otherwise initialize the stage before processing any records.

The Pre-Loop sub-tab allows you to specify code which is executed at the beginning of the stage, before any records are processed.

The Per-Record sub-tab allows you to specify the code which is executed once for every record processed.

The Post-Loop sub-tab allows you to specify code that is executed after all the records have been processed.

You can type straight into these pages or cut and paste from another editor. The shortcut menu on the Pre-Loop, Per-Record, and Post-Loop pages gives access to the macros that are available for use in the code.

The Advanced tab allows you to specify details about how the stage is compiled and built. Fill in the page as follows:

- Compile and Link Flags. Allows you to specify flags that are passed to the C++ compiler.
- Verbose. Select this check box to specify that the compile and build is done in verbose mode.
- Debug. Select this check box to specify that the compile and build is done in debug mode. Otherwise, it is done in optimize mode.
- Suppress Compile. Select this check box to generate files without compiling, and without deleting the generated files. This option is useful for fault finding.
- Base File Name. The base filename for generated files. All generated files will have this name followed by the appropriate suffix. This defaults to the name specified under Operator on the General page.
- Source Directory. The directory where generated .c files are placed. This defaults to the buildop folder in the current project directory. You can also set it using the DS_OPERATOR_BUILDOP_DIR environment variable in the WebSphere DataStage Administrator (see *WebSphere DataStage Administrator Client Guide*).
- Header Directory. The directory where generated .h files are placed. This defaults to the buildop folder in the current project directory. You can also set it using the DS_OPERATOR_BUILDOP_DIR environment variable in the WebSphere DataStage Administrator (see *WebSphere DataStage Administrator Client Guide*).
- Object Directory. The directory where generated .so files are placed. This defaults to the buildop folder in the current project directory. You can also set it using the DS_OPERATOR_BUILDOP_DIR environment variable in the WebSphere DataStage Administrator (see *WebSphere DataStage Administrator Client Guide*).
- Wrapper directory. The directory where generated .op files are placed. This defaults to the buildop folder in the current project directory. You can also set it using the DS_OPERATOR_BUILDOP_DIR environment variable in the WebSphere DataStage Administrator (see *WebSphere DataStage Administrator Client Guide*).

7. When you have filled in the details in all the pages, click **Generate** to generate the stage. A window appears showing you the result of the build.

Build stage macros

There are a number of macros you can use when specifying Pre-Loop, Per-Record, and Post-Loop code. Insert a macro by selecting it from the short cut menu. They are grouped into the following categories:

- Informational
- Flow-control
- Input and output
- Transfer

Informational macros

Use these macros in your code to determine the number of inputs, outputs, and transfers as follows:

- `inputs()`. Returns the number of inputs to the stage.
- `outputs()`. Returns the number of outputs from the stage.
- `transfers()`. Returns the number of transfers in the stage.

Flow-control macros

Use these macros to override the default behavior of the Per-Record loop in your stage definition:

- `endLoop()`. Causes the operator to stop looping, following completion of the current loop and after writing any auto outputs for this loop.
- `nextLoop()` Causes the operator to immediately skip to the start of next loop, without writing any outputs.
- `failStep()` Causes the operator to return a failed status and terminate the job.

Input and output macros

These macros allow you to explicitly control the read and write and transfer of individual records.

Each of the macros takes an argument as follows:

- *input* is the index of the input (0 to *n*). If you have defined a name for the input port you can use this in place of the index in the form *portname.portid_*.
- *output* is the index of the output (0 to *n*). If you have defined a name for the output port you can use this in place of the index in the form *portname.portid_*.
- *index* is the index of the transfer (0 to *n*).

The following macros are available:

- `readRecord(input)`. Immediately reads the next record from *input*, if there is one. If there is no record, the next call to `inputDone()` will return true.
- `writeRecord(output)`. Immediately writes a record to *output*.
- `inputDone(input)`. Returns true if the last call to `readRecord()` for the specified input failed to read a new record, because the input has no more records.
- `holdRecord(input)`. Causes auto input to be suspended for the current record, so that the operator does not automatically read a new record at the start of the next loop. If auto is not set for the input, `holdRecord()` has no effect.
- `discardRecord(output)`. Causes auto output to be suspended for the current record, so that the operator does not output the record at the end of the current loop. If auto is not set for the output, `discardRecord()` has no effect.
- `discardTransfer(index)`. Causes auto transfer to be suspended, so that the operator does not perform the transfer at the end of the current loop. If auto is not set for the transfer, `discardTransfer()` has no effect.

Transfer Macros

These macros allow you to explicitly control the transfer of individual records.

Each of the macros takes an argument as follows:

- *input* is the index of the input (0 to *n*). If you have defined a name for the input port you can use this in place of the index in the form *portname.portid_*.
- *output* is the index of the output (0 to *n*). If you have defined a name for the output port you can use this in place of the index in the form *portname.portid_*.
- *index* is the index of the transfer (0 to *n*).

The following macros are available:

- `doTransfer(index)`. Performs the transfer specified by index.
- `doTransfersFrom(input)`. Performs all transfers from input.
- `doTransfersTo(output)`. Performs all transfers to output.
- `transferAndWriteRecord(output)`. Performs all transfers and writes a record for the specified output. Calling this macro is equivalent to calling the macros `doTransfersTo()` and `writeRecord()`.

How your code is executed

This section describes how the code that you define when specifying a Build stage executes when the stage is run in a WebSphere DataStage job.

The sequence is as follows:

1. Handles any definitions that you specified in the **Definitions** sub-tab when you entered the stage details.
2. Executes any code that was entered in the **Pre-Loop** sub-tab.
3. Loops repeatedly until either all inputs have run out of records, or the **Per-Record** code has explicitly invoked `endLoop()`. In the loop, performs the following steps:
 - a. Reads one record for each input, except where any of the following is true:
 - b. The input has no more records left.
 - c. The input has Auto Read set to false.
 - d. The `holdRecord()` macro was called for the input last time around the loop.
 - e. Executes the Per-Record code, which can explicitly read and write records, perform transfers, and invoke loop-control macros such as `endLoop()`.
 - f. Performs each specified transfer, except where any of the following is true:
 - g. The input of the transfer has no more records.
 - h. The transfer has Auto Transfer set to False.
 - i. The `discardTransfer()` macro was called for the transfer during the current loop iteration.
 - j. Writes one record for each output, except where any of the following is true:
 - k. The output has Auto Write set to false.
 - l. The `discardRecord()` macro was called for the output during the current loop iteration.
4. If you have specified code in the **Post-loop** sub-tab, executes it.
5. Returns a status, which is written to the WebSphere DataStage Job Log.

Inputs and outputs

The input and output ports that you defined for your Build stage are where input and output links attach to the stage. By default, links are connected to ports in the order they are connected to the stage, but where your stage allows multiple input or output links you can change the link order using the Link Order tab on the stage editor.

When you specify details about the input and output ports for your Build stage, you need to define the meta data for the ports. You do this by loading a table definition from the WebSphere DataStage Repository.

When you actually use your stage in a job, you have to specify meta data for the links that attach to these ports. For the job to run successfully the meta data specified for the port and that specified for the link should match. An exception to this is where you have runtime column propagation enabled for the job. In this case the input link meta data can be a super-set of the port meta data and the extra columns will be automatically propagated.

Using multiple inputs

Where you require your stage to handle multiple inputs, there are some special considerations. Your code needs to ensure the following:

- The stage only tries to access a column when there are records available. It should not try to access a column after all records have been read (use the `inputDone()` macro to check), and should not attempt to access a column unless either Auto Read is enabled on the link or an explicit read record has been performed.
- The reading of records is terminated immediately after all the required records have been read from it. In the case of a port with Auto Read disabled, the code must determine when all required records have been read and call the `endLoop()` macro.

In most cases you might keep Auto Read enabled when you are using multiple inputs, this minimizes the need for explicit control in your code. But there are circumstances when this is not appropriate. The following paragraphs describes some common scenarios:

Using auto read for all inputs

All ports have Auto Read enabled and so all record reads are handled automatically. You need to code for Per-record loop such that each time it accesses a column on any input it first uses the `inputDone()` macro to determine if there are any more records.

This method is fine if you want your stage to read a record from every link, every time round the loop.

Using inputs with auto read enabled for some and disabled for others

You define one (or possibly more) inputs as Auto Read, and the rest with Auto Read disabled. You code the stage in such a way as the processing of records from the Auto Read input drives the processing of the other inputs. Each time round the loop, your code should call `inputDone()` on the Auto Read input and call `exitLoop()` to complete the actions of the stage.

This method is fine where you process a record from the Auto Read input every time around the loop, and then process records from one or more of the other inputs depending on the results of processing the Auto Read record.

Using inputs with auto read disabled

Your code must explicitly perform all record reads. You should define Per-Loop code which calls `readRecord()` once for each input to start processing. Your Per-record code should call `inputDone()` for every input each time round the loop to determine if a record was read on the most recent `readRecord()`, and if it did, call `readRecord()` again for that input. When all inputs run out of records, the Per-Loop code should exit.

This method is intended where you want explicit control over how each input is treated.

Example Build Stage

This section shows you how to define a Build stage called Divide, which basically divides one number by another and writes the result and any remainder to an output link. The stage also checks whether you are trying to divide by zero and, if you are, sends the input record down a reject link.

To demonstrate the use of properties, the stage also lets you define a minimum divisor. If the number you are dividing by is smaller than the minimum divisor you specify when adding the stage to a job, then the record is also rejected.

The input to the stage is defined as auto read, while the two outputs have auto write disabled. The code has to explicitly write the data to one or other of the output links. In the case of a successful division the data written is the original record plus the result of the division and any remainder. In the case of a rejected record, only the original record is written.

The input record has two columns: dividend and divisor. Output 0 has four columns: dividend, divisor, result, and remainder. Output 1 (the reject link) has two columns: dividend and divisor.

If the divisor column of an input record contains zero or is less than the specified minimum divisor, the record is rejected, and the code uses the macro transferAndWriteRecord(1) to transfer the data to port 1 and write it. If the divisor is not zero, the code uses doTransfersTo(0) to transfer the input record to Output 0, assigns the division results to result and remainder and finally calls writeRecord(0) to write the record to output 0.

The following screen shots show how this stage is defined in WebSphere DataStage using the Stage Type dialog box:

1. First general details are supplied in the General tab.
2. Details about the stage's creation are supplied on the Creator page.
3. The optional property of the stage is defined in the Properties tab.
4. Details of the inputs and outputs is defined on the interfaces tab of the Build page.

Details about the single input to Divide are given on the **Input** sub-tab of the Interfaces tab. A table definition for the inputs link is available to be loaded from the WebSphere DataStage Repository

Details about the outputs are given on the **Output** sub-tab of the Interfaces tab.

When you use the stage in a job, make sure that you use table definitions compatible with the tables defined in the input and output sub-tabs.

Details about the transfers carried out by the stage are defined on the **Transfer** sub-tab of the Interfaces tab.

5. The code itself is defined on the Logic tab. In this case all the processing is done in the Per-Record loop and so is entered on the **Per-Record** sub-tab.
6. As this example uses all the compile and build defaults, all that remains is to click **Generate** to build the stage.

Defining wrapped stages

You define a Wrapped stage to enable you to specify a UNIX command to be executed by a WebSphere DataStage stage. You define a wrapper file that handles arguments for the UNIX command and inputs and outputs. The Designer provides an interface that helps you define the wrapper. The stage will be available to all jobs in the project in which the stage was defined. You can make it available to other projects using the Designer Export facilities. You can add the stage to your job palette using palette customization features in the Designer.

When defining a Wrapped stage you provide the following information:

- Details of the UNIX command that the stage will execute.
- Description of the data that will be input to the stage.
- Description of the data that will be output from the stage.
- Definition of the environment in which the command will execute.

The UNIX command that you wrap can be a built-in command, such as grep, a utility, such as SyncSort, or your own UNIX application. The only limitation is that the command must be ‘pipe-safe’ (to be pipe-safe a UNIX command reads its input sequentially, from beginning to end).

You need to define meta data for the data being input to and output from the stage. You also need to define the way in which the data will be input or output. UNIX commands can take their inputs from standard in, or another stream, a file, or from the output of another command via a pipe. Similarly data is output to standard out, or another stream, to a file, or to a pipe to be input to another command. You specify what the command expects.

WebSphere DataStage handles data being input to the Wrapped stage and will present it in the specified form. If you specify a command that expects input on standard in, or another stream, WebSphere DataStage will present the input data from the job's data flow as if it was on standard in. Similarly it will intercept data output on standard out, or another stream, and integrate it into the job's data flow.

You also specify the environment in which the UNIX command will be executed when you define the wrapped stage.

To define a Wrapped stage:

1. Do one of:
 - a. Choose **File** → **New** from the Designer menu. The New dialog box appears.
 - b. Open the **Stage Type** folder and select the **Parallel Wrapped Stage Type** icon.
 - c. Click **OK**. The Stage Type dialog box appears, with the General page on top.
Or:
 - d. Select a folder in the repository tree.
 - e. Choose **New** → **Other Parallel Stage Wrapped** from the shortcut menu. The Stage Type dialog box appears, with the General page on top.
2. Fill in the fields on the General page as follows:
 - **Stage type name.** This is the name that the stage will be known by to WebSphere DataStage. Avoid using the same name as existing stages or the name of the actual UNIX command you are wrapping.
 - **Category.** The category that the new stage will be stored in under the stage types branch. Type in or browse for an existing category or type in the name of a new one. The category also determines what group in the palette the stage will be added to. Choose an existing category to add to an existing group, or specify a new category to create a new palette group.
 - **Parallel Stage type.** This indicates the type of new Parallel job stage you are defining (Custom, Build, or Wrapped). You cannot change this setting.
 - **Wrapper Name.** The name of the wrapper file WebSphere DataStage will generate to call the command. By default this will take the same name as the Stage type name.
 - **Execution mode.** Choose the default execution mode. This is the mode that will appear in the Advanced tab on the stage editor. You can override this mode for individual instances of the stage as required, unless you select **Parallel only** or **Sequential only**. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the execution mode.
 - **Preserve Partitioning.** This shows the default setting of the Preserve Partitioning flag, which you cannot change in a Wrapped stage. This is the setting that will appear in the Advanced tab on the stage editor. You can override this setting for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the preserve partitioning flag.

- **Partitioning.** This shows the default partitioning method, which you cannot change in a Wrapped stage. This is the method that will appear in the Inputs Page Partitioning tab of the stage editor. You can override this method for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the partitioning methods.
- **Collecting.** This shows the default collection method, which you cannot change in a Wrapped stage. This is the method that will appear in the Inputs Page Partitioning tab of the stage editor. You can override this method for individual instances of the stage as required. See *WebSphere DataStage Parallel Job Developer Guide* for a description of the collection methods.
- **Command.** The name of the UNIX command to be wrapped, plus any required arguments. The arguments that you enter here are ones that do not change with different invocations of the command. Arguments that need to be specified when the Wrapped stage is included in a job are defined as properties for the stage.
- **Short Description.** Optionally enter a short description of the stage.
- **Long Description.** Optionally enter a long description of the stage.

3. Go to the Creator page and optionally specify information about the stage you are creating. We recommend that you assign a release number to the stage so you can keep track of any subsequent changes.

You can specify that the actual stage will use a custom GUI by entering the ProgID for a custom GUI in the **Custom GUI Prog ID** field.

You can also specify that the stage has its own icon. You need to supply a 16 x 16 bit bitmap and a 32 x 32 bit bitmap to be displayed in various places in the WebSphere DataStage user interface. Click the 16 x 16 Bitmap button and browse for the smaller bitmap file. Click the 32 x 32 Bitmap button and browse for the large bitmap file. Note that bitmaps with 32-bit color are not supported. Click the **Reset Bitmap Info** button to revert to using the default WebSphere DataStage icon for this stage.

4. Go to the Properties page. This allows you to specify the arguments that the UNIX command requires as properties that appear in the stage Properties tab. For wrapped stages the Properties tab always appears under the Stage page.

Fill in the fields as follows:

- Property name. The name of the property that will be displayed on the Properties tab of the stage editor.
- Data type. The data type of the property. Choose from:

Boolean

Float

Integer

String

Pathname

List

Input Column

Output Column

If you choose Input Column or Output Column, when the stage is included in a job a drop-down list will offer a choice of the defined input or output columns.

If you choose list you should open the Extended Properties dialog box from the grid shortcut menu to specify what appears in the list.

- Prompt. The name of the property that will be displayed on the Properties tab of the stage editor.
- Default Value. The value the option will take if no other is specified.
- Required. Set this to True if the property is mandatory.
- Repeats. Set this true if the property repeats (that is, you can have multiple instances of it).
- Conversion. Specifies the type of property as follows:

- Name. The name of the property will be passed to the command as the argument value. This will normally be a hidden property, that is, not visible in the stage editor.
 - Name Value. The name of the property will be passed to the command as the argument name, and any value specified in the stage editor is passed as the value.
 - Value. The value for the property specified in the stage editor is passed to the command as the argument name. Typically used to group operator options that are mutually exclusive.
 - Value only. The value for the property specified in the stage editor is passed as it is.
5. If you want to specify a list property, or otherwise control how properties are handled by your stage, choose **Extended Properties** from the Properties grid shortcut menu to open the Extended Properties dialog box.

The settings you use depend on the type of property you are specifying:

- Specify a category to have the property appear under this category in the stage editor. By default all properties appear in the Options category.
- If you are specifying a List category, specify the possible values for list members in the List Value field.
- If the property is to be a dependent of another property, select the parent property in the Parents field.
- Specify an expression in the Template field to have the actual value of the property generated at compile time. It is usually based on values in other properties and columns.
- Specify an expression in the Conditions field to indicate that the property is only valid if the conditions are met. The specification of this property is a bar '|' separated list of conditions that are AND'ed together. For example, if the specification was a=b | c!=d, then this property would only be valid (and therefore only available in the GUI) when property a is equal to b, and property c is not equal to d.

Click OK when you are happy with the extended properties.

6. **Go to the Wrapped page.** This allows you to specify information about the command to be executed by the stage and how it will be handled.

The Interfaces tab is used to describe the inputs to and outputs from the stage, specifying the interfaces that the stage will need to function.

Details about inputs to the stage are defined on the Inputs sub-tab:

- Link. The link number, this is assigned for you and is read-only. When you actually use your stage, links will be assigned in the order in which you add them. In the example, the first link will be taken as link 0, the second as link 1 and so on. You can reassign the links using the stage editor's Link Ordering tab on the General page.
- Table Name. The meta data for the link. You define this by loading a table definition from the Repository. Type in the name, or browse for a table definition. Alternatively, you can specify an argument to the UNIX command which specifies a table definition. In this case, when the wrapped stage is used in a job design, the designer will be prompted for an actual table definition to use.
- Stream. Here you can specify whether the UNIX command expects its input on standard in, or another stream, or whether it expects it in a file. Click on the browse button to open the Wrapped Stream dialog box.

In the case of a file, you should also specify whether the file to be read is given in a command line argument, or by an environment variable.

Details about outputs from the stage are defined on the Outputs sub-tab:

- Link. The link number, this is assigned for you and is read-only. When you actually use your stage, links will be assigned in the order in which you add them. In the example, the first link will be taken as link 0, the second as link 1 and so on. You can reassign the links using the stage editor's Link Ordering tab on the General page.
- Table Name. The meta data for the link. You define this by loading a table definition from the Repository. Type in the name, or browse for a table definition.

- Stream. Here you can specify whether the UNIX command will write its output to standard out, or another stream, or whether it outputs to a file. Click on the browse button to open the Wrapped Stream dialog box.

In the case of a file, you should also specify whether the file to be written is specified in a command line argument, or by an environment variable.

The Environment tab gives information about the environment in which the command will execute. Set the following on the Environment tab:

- All Exit Codes Successful. By default WebSphere DataStage treats an exit code of 0 as successful and all others as errors. Select this check box to specify that all exit codes should be treated as successful other than those specified in the Failure codes grid.
- Exit Codes. The use of this depends on the setting of the All Exits Codes Successful check box. If All Exits Codes Successful is not selected, enter the codes in the Success Codes grid which will be taken as indicating successful completion. All others will be taken as indicating failure. If All Exits Codes Successful is selected, enter the exit codes in the Failure Code grid which will be taken as indicating failure. All others will be taken as indicating success.
- Environment. Specify environment variables and settings that the UNIX command requires in order to run.

7. When you have filled in the details in all the pages, click **Generate** to generate the stage.

Example wrapped stage

This section shows you how to define a Wrapped stage called exhort which runs the UNIX sort command in parallel. The stage sorts data in two files and outputs the results to a file. The incoming data has two columns, order number and code. The sort command sorts the data on the second field, code. You can optionally specify that the sort is run in reverse order.

Wrapping the sort command in this way would be useful if you had a situation where you had a fixed sort operation that was likely to be needed in several jobs. Having it as an easily reusable stage would save having to configure a built-in sort stage every time you needed it.

When included in a job and run, the stage will effectively call the Sort command as follows:

```
sort -r -o outfile -k 2 infile1 infile2
```

The following screen shots show how this stage is defined in WebSphere DataStage using the Stage Type dialog box:

1. First general details are supplied in the General tab. The argument defining the second column as the key is included in the command because this does not vary:
2. The reverse order argument (-r) are included as properties because it is optional and might or might not be included when the stage is incorporated into a job.
3. The fact that the sort command expects two files as input is defined on the **Input** sub-tab on the Interfaces tab of the Wrapper page.
4. The fact that the sort command outputs to a file is defined on the **Output** sub-tab on the Interfaces tab of the Wrapper page.

Note: When you use the stage in a job, make sure that you use table definitions compatible with the tables defined in the input and output sub-tabs.

5. Because all exit codes other than 0 are treated as errors, and because there are no special environment requirements for this command, you do not need to alter anything on the Environment tab of the Wrapped page. All that remains is to click **Generate** to build the stage.

Chapter 6. Environment Variables

These topics list the environment variables that are available for affecting the set up and operation of parallel jobs.

There are many environment variables that affect the design and running of parallel jobs in WebSphere DataStage. Commonly used ones are exposed in the WebSphere DataStage Administrator client, and can be set or unset using the Administrator (see *WebSphere DataStage Administrator Client Guide*). There are additional environment variables, however. This topic describes all the environment variables that apply to parallel jobs. They can be set or unset as you would any other UNIX system variables, or you can add them to the User Defined section in the WebSphere DataStage Administrator environment variable tree.

The available environment variables are grouped according to function. They are summarized in the following table.

The final section in this topic gives some guidance to setting the environment variables.

Category	Environment Variable
----------	----------------------

Buffering

- APT_BUFFER_FREE_RUN
- APT_BUFFER_MAXIMUM_MEMORY
- APT_BUFFER_MAXIMUM_TIMEOUT
- APT_BUFFER_DISK_WRITE_INCREMENT
- APT_BUFFERING_POLICY
- APT_SHARED_MEMORY_BUFFERS

Building Custom Stages

- DS_OPERATOR_BUILDOP_DIR
- OSH_BUILDOP_CODE
- OSH_BUILDOP_HEADER
- OSH_BUILDOP_OBJECT
- OSH_BUILDOP_XLC_BIN
- OSH_CBUILDOP_XLC_BIN

Compiler

- APT_COMPILER
- APT_COMPILEOPT
- APT_LINKER
- APT_LINKOPT

DB2 Support

- APT_DB2INSTANCE_HOME
- APT_DB2READ_LOCK_TABLE
- APT_DBNAME
- APT_RDBMS_COMMIT_ROWS

DB2DBDFT

Debugging

APT_DEBUG_OPERATOR
APT_DEBUG_MODULE_NAMES
APT_DEBUG_PARTITION
APT_DEBUG_SIGNALS
APT_DEBUG_STEP
APT_DEBUG_SUBPROC
APT_EXECUTION_MODE
APT_PM_DBX
APT_PM_GDB
APT_PM_SHOW_PIDS
APT_PM_XLDB
APT_PM_XTERM
APT_SHOW_LIBLOAD

Decimal Support

APT_DECIMAL_INTERM_PRECISION
APT_DECIMAL_INTERM_SCALE
APT_DECIMAL_INTERM_ROUND_MODE

Disk I/O

APT_BUFFER_DISK_WRITE_INCREMENT
APT_CONSISTENT_BUFFERIO_SIZE
APT_EXPORT_FLUSH_COUNT
APT_IO_MAP/APT_IO_NOMAP and APT_BUFFERIO_MAP/APT_BUFFERIO_NOMAP
APT_PHYSICAL_DATASET_BLOCK_SIZE

General Job Administration

APT_CHECKPOINT_DIR
APT_CLOBBER_OUTPUT
APT_CONFIG_FILE
APT_DISABLE_COMBINATION
APT_EXECUTION_MODE
APT_ORCHHOME
APT_STARTUP_SCRIPT
APT_NO_STARTUP_SCRIPT
APT_STARTUP_STATUS
APT_THIN_SCORE

Job Monitoring

APT_MONITOR_SIZE
APT_MONITOR_TIME

APT_NO_JOBMON
APT_PERFORMANCE_DATA

Look Up support

APT_LUTCREATE_NO_MMAP

Miscellaneous

APT_COPY_TRANSFORM_OPERATOR
“APT_EBCDIC_VERSION” on page 65
APT_DATE_CENTURY_BREAK_YEAR
APT_IMPEXP_ALLOW_ZERO_LENGTH_FIXED_NULL
APT_IMPORT_REJECT_STRING_FIELD_OVERRUNS
APT_INSERT_COPY_BEFORE MODIFY
APT_OLD_BOUNDED_LENGTH
APT_OPERATOR_REGISTRY_PATH
APT_PM_NO_SHARED_MEMORY
APT_PM_NO_NAMED_PIPES
APT_PM_SOFT_KILL_WAIT
APT_PM_STARTUP_CONCURRENCY
APT_RECORD_COUNTS
APT_SAVE_SCORE
APT_SHOW_COMPONENT_CALLS
APT_STACK_TRACE
APT_WRITE_DS_VERSION
OSH_PRELOAD_LIBS

Network

APT_IO_MAXIMUM_OUTSTANDING
APT_IOMGR_CONNECT_ATTEMPTS
APT_PM_CONDUCTOR_HOSTNAME
APT_PM_NO_TCPIP
APT_PM_NODE_TIMEOUT
APT_PM_SHOWRSH
“APT_PM_STARTUP_PORT” on page 68
APT_PM_USE_RSH_LOCALLY

APT_RECVBUFSIZE
APT_RECVBUFSIZE

NLS APT_COLLATION_SEQUENCE
APT_COLLATION_STRENGTH
APT_ENGLISH_MESSAGES
APT_IMPEXP_CHARSET

APT_INPUT_CHARSET
APT_OS_CHARSET
APT_OUTPUT_CHARSET
APT_STRING_CHARSET

Oracle Support

APT_ORACLE_LOAD_DELIMITED
APT_ORACLE_LOAD_OPTIONS
APT_ORACLE_NO_OPS
APT_ORACLE_PRESERVE_BLANKS
APT_ORA_IGNORE_CONFIG_FILE_PARALLELISM
APT_ORA_WRITE_FILES
APT_ORAUPSERT_COMMIT_ROW_INTERVAL APT_ORAUPSERT_COMMIT_TIME_INTERVAL

Partitioning

APT_NO_PART_INSERTION
APT_PARTITION_COUNT
APT_PARTITION_NUMBER

Reading and Writing Files

APT_DELIMITED_READ_SIZE
APT_FILE_IMPORT_BUFFER_SIZE
APT_FILE_EXPORT_BUFFER_SIZE
APT_IMPORT_PATTERNUSESFILESET
APT_MAX_DELIMITED_READ_SIZE
APT_PREVIOUS_FINAL_DELIMITER_COMPATIBLE
APT_STRING_PADCHAR

Reporting

APT_DUMP_SCORE
APT_ERROR_CONFIGURATION
APT_MSG_FILELINE
APT_PM_PLAYER_MEMORY
APT_PM_PLAYER_TIMING
APT_RECORD_COUNTS
OSH_DUMP
OSH_ECHO
OSH_EXPLAIN
OSH_PRINT_SCHEMAS

SAS Support

APT_HASH_TO_SASHASH
APT_NO_SASOUT_INSERT
APT_NO_SAS_TRANSFORMS

APT_SAS_ACCEPT_ERROR
APT_SAS_CHARSET
APT_SAS_CHARSET_ABORT
APT_SAS_COMMAND
APT_SASINT_COMMAND
APT_SAS_DEBUG
APT_SAS_DEBUG_IO
APT_SAS_DEBUG_LEVEL
APT_SAS_DEBUG_VERBOSE
APT_SAS_NO_PSDS_USTRING
APT_SAS_S_ARGUMENT
APT_SAS_SCHEMASOURCE_DUMP
APT_SAS_SHOW_INFO
APT_SAS_TRUNCATION

Sorting

APT_NO_SORT_INSERTION
APT_SORT_INSERTION_CHECK_ONLY

Teradata Support

APT_TERA_64K_BUFFERS
APT_TERA_NO_ERR_CLEANUP
APT_TERA_NO_PERM_CHECKS
APT_TERA_NO_SQL_CONVERSION
APT_TERA_SYNC_DATABASE
APT_TERA_SYNC_USER

Transport Blocks

APT_AUTO_TRANSPORT_BLOCK_SIZE
APT_LATENCY_COEFFICIENT
APT_DEFAULT_TRANSPORT_BLOCK_SIZE
APT_MAX_TRANSPORT_BLOCK_SIZE/ APT_MIN_TRANSPORT_BLOCK_SIZE

Buffering

These environment variable are all concerned with the buffering WebSphere DataStage performs on stage links to avoid deadlock situations. These settings can also be made on the Inputs page or Outputs page Advanced tab of the parallel stage editors.

APT_BUFFER_FREE_RUN

This environment variable is available in the WebSphere DataStage Administrator, under the Parallel category. It specifies how much of the available in-memory buffer to consume before the buffer resists. This is expressed as a decimal representing the percentage of Maximum memory buffer size (for example,

0.5 is 50%). When the amount of data in the buffer is less than this value, new data is accepted automatically. When the data exceeds it, the buffer first tries to write some of the data it contains before accepting more.

The default value is 50% of the Maximum memory buffer size. You can set it to greater than 100%, in which case the buffer continues to store data up to the indicated multiple of Maximum memory buffer size before writing to disk.

APT_BUFFER_MAXIMUM_MEMORY

Sets the default value of Maximum memory buffer size. The default value is 3145728 (3 MB). Specifies the maximum amount of virtual memory, in bytes, used per buffer.

APT_BUFFER_MAXIMUM_TIMEOUT

WebSphere DataStage buffering is self tuning, which can theoretically lead to long delays between retries. This environment variable specified the maximum wait before a retry in seconds, and is by default set to 1.

APT_BUFFER_DISK_WRITE_INCREMENT

Sets the size, in bytes, of blocks of data being moved to/from disk by the buffering operator. The default is 1048576 (1 MB). Adjusting this value trades amount of disk access against throughput for small amounts of data. Increasing the block size reduces disk access, but might decrease performance when data is being read/written in smaller units. Decreasing the block size increases throughput, but might increase the amount of disk access.

APT_BUFFERING_POLICY

This environment variable is available in the WebSphere DataStage Administrator, under the Parallel category. Controls the buffering policy for all virtual data sets in all steps. The variable has the following settings:

- AUTOMATIC_BUFFERING (default). Buffer a data set only if necessary to prevent a data flow deadlock.
- FORCE_BUFFERING. Unconditionally buffer all virtual data sets. Note that this can slow down processing considerably.
- NO_BUFFERING. Do not buffer data sets. This setting can cause data flow deadlock if used inappropriately.

APT_SHARED_MEMORY_BUFFERS

Typically the number of shared memory buffers between two processes is fixed at 2. Setting this will increase the number used. The likely result of this is POSSIBLY both increased latency and increased performance. This setting can significantly increase memory use.

Building Custom Stages

These environment variables are concerned with the building of custom operators that form the basis of customized stages (as described in Specifying your own parallel stages,

DS_OPERATOR_BUILDOP_DIR

Identifies the directory in which generated buildops are created. By default this identifies a directory called buildop under the current project directory. If the directory is changed, the corresponding entry in APT_OPERATOR_REGISTRY_PATH needs to change to match the buildop folder.

OSH_BUILDOP_CODE

Identifies the directory into which buildop writes the generated .C file and build script. It defaults to the current working directory. The -C option of buildop overrides this setting.

OSH_BUILDOP_HEADER

Identifies the directory into which buildop writes the generated .h file. It defaults to the current working directory. The -H option of buildop overrides this setting.

OSH_BUILDOP_OBJECT

Identifies the directory into which buildop writes the dynamically loadable object file, whose extension is .so on Solaris, .sl on HP-UX, or .o on AIX. Defaults to the current working directory.

The -O option of buildop overrides this setting.

OSH_BUILDOP_XLC_BIN

AIX only. Identifies the directory specifying the location of the shared library creation command used by buildop.

On older AIX systems this defaults to /usr/lpp/xlc/bin/makeC++SharedLib_r for thread-safe compilation. On newer AIX systems it defaults to /usr/ibmcxx/bin/makeC++SharedLib_r. For non-thread-safe compilation, the default path is the same, but the name of the file is makeC++SharedLib.

OSH_CBUILDOP_XLC_BIN

AIX only. Identifies the directory specifying the location of the shared library creation command used by cbuildop. If this environment variable is not set, cbuilder checks the setting of OSH_BUILDOP_XLC_BIN for the path. On older AIX systems OSH_CBUILDOP_XLC_BIN defaults to /usr/lpp/xlc/bin/makeC++SharedLib_r for thread-safe compilation. On newer AIX systems it defaults to /usr/ibmcxx/bin/makeC++SharedLib_r. For non-threadsafe compilation, the default path is the same, but the name of the command is makeC++SharedLib.

Compiler

These environment variables specify details about the C++ compiler used by WebSphere DataStage in connection with parallel jobs.

APT_COMPILER

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Compiler** branch. Specifies the full path to the C++ compiler.

APT_COMPILEOPT

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Compiler** branch. Specifies extra options to be passed to the C++ compiler when it is invoked.

APT_LINKER

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Compiler** branch. Specifies the full path to the C++ linker.

APT_LINKOPT

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Compiler** branch. Specifies extra options to be passed to the C++ linker when it is invoked.

DB2 Support

These environment variables are concerned with setting up access to DB2 databases from WebSphere DataStage.

APT_DB2INSTANCE_HOME

Specifies the DB2 installation directory. This variable is set by WebSphere DataStage to values obtained from the DB2Server table, representing the currently selected DB2 server.

APT_DB2READ_LOCK_TABLE

If this variable is defined and the open option is not specified for the DB2 stage, WebSphere DataStage performs the following open command to lock the table:

```
lock table 'table_name' in share mode
```

APT_DBNAME

Specifies the name of the database if you choose to leave out the Database option for DB2 stages. If APT_DBNAME is not defined as well, DB2DBDFT is used to find the database name. These variables are set by WebSphere DataStage to values obtained from the DB2Server table, representing the currently selected DB2 server.

APT_RDBMS_COMMIT_ROWS

Specifies the number of records to insert into a data set between commits. The default value is 2048.

DB2DBDFT

For DB2 operators, you can set the name of the database by using the **-dbname** option or by setting APT_DBNAME. If you do not use either method, DB2DBDFT is used to find the database name. These variables are set by WebSphere DataStage to values obtained from the DB2Server table, representing the currently selected DB2 server.

Debugging

These environment variables are concerned with the debugging of WebSphere DataStage parallel jobs.

APT_DEBUG_OPERATOR

Specifies the operators on which to start debuggers. If not set, no debuggers are started. If set to an operator number (as determined from the output of APT_DUMP_SCORE), debuggers are started for that single operator. If set to -1, debuggers are started for all operators.

APT_DEBUG_MODULE_NAMES

This comprises a list of module names separated by white space that are the modules to debug, that is, where internal IF_DEBUG statements will be run. The subproc operator module (module name is "subproc") is one example of a module that uses this facility.

APT_DEBUG_PARTITION

Specifies the partitions on which to start debuggers. One instance, or partition, of an operator is run on each node running the operator. If set to a single number, debuggers are started on that partition; if not set or set to -1, debuggers are started on all partitions.

See the description of APT_DEBUG_OPERATOR for more information on using this environment variable.

For example, setting APT_DEBUG_STEP to 0, APT_DEBUG_OPERATOR to 1, and APT_DEBUG_PARTITION to -1 starts debuggers for every partition of the second operator in the first step.

APT_DEBUG_OPERATOR	APT_DEBUG_PARTITION	Effect
not set	any value	no debugging
-1	not set or -1	debug all partitions of all operators
-1	≥ 0	debug a specific partition of all operators
≥ 0	-1	debug all partitions of a specific operator
≥ 0	≥ 0	debug a specific partition of a specific operator

APT_DEBUG_SIGNALS

You can use the APT_DEBUG_SIGNALS environment variable to specify that signals such as SIGSEGV, SIGBUS, and so on, should cause a debugger to start. If any of these signals occurs within an APT_Operator::runLocally() function, a debugger such as dbx is invoked.

Note that the UNIX and WebSphere DataStage variables DEBUGGER, DISPLAY, and APT_PM_XTERM, specifying a debugger and how the output should be displayed, must be set correctly.

APT_DEBUG_STEP

Specifies the steps on which to start debuggers. If not set or if set to -1, debuggers are started on the processes specified by APT_DEBUG_OPERATOR and APT_DEBUG_PARTITION in all steps. If set to a step number, debuggers are started for processes in the specified step.

APT_DEBUG_SUBPROC

Display debug information about each subprocess operator.

APT_EXECUTION_MODE

This environment variable is available in the WebSphere DataStage Administrator under the Parallel branch. By default, the execution mode is parallel, with multiple processes. Set this variable to one of the following values to run an application in sequential execution mode:

- **ONE_PROCESS** one-process mode
- **MANY_PROCESS** many-process mode
- **NO_SERIALIZE** many-process mode, without serialization

In **ONE_PROCESS** mode:

- The application executes in a single UNIX process. You need only run a single debugger session and can set breakpoints anywhere in your code.
- Data is partitioned according to the number of nodes defined in the configuration file.
- Each operator is run as a subroutine and is called the number of times appropriate for the number of partitions on which it must operate.

In **MANY_PROCESS** mode the framework forks a new process for each instance of each operator and waits for it to complete rather than calling operators as subroutines.

In both cases, the step is run entirely on the Conductor node rather than spread across the configuration.

NO_SERIALIZE mode is similar to **MANY_PROCESS** mode, but the WebSphere DataStage persistence mechanism is not used to load and save objects. Turning off persistence might be useful for tracking errors in derived C++ classes.

APT_PM_DBX

Set this environment variable to the path of your dbx debugger, if a debugger is not already included in your path. This variable sets the location; it does not run the debugger.

APT_PM_GDB

Linux only. Set this environment variable to the path of your xldb debugger, if a debugger is not already included in your path. This variable sets the location; it does not run the debugger.

APT_PM_LADEBUG

Tru64 only. Set this environment variable to the path of your dbx debugger, if a debugger is not already included in your path. This variable sets the location; it does not run the debugger.

APT_PM_SHOW_PIDS

If this variable is set, players will output an informational message upon startup, displaying their process id.

APT_PM_XLDB

Set this environment variable to the path of your xlbd debugger, if a debugger is not already included in your path. This variable sets the location; it does not run the debugger.

APT_PM_XTERM

If WebSphere DataStage invokes dbx, the debugger starts in an xterm window; this means WebSphere DataStage must know where to find the xterm program. The default location is /usr/bin/X11/xterm. You can override this default by setting the APT_PM_XTERM environment variable to the appropriate path. APT_PM_XTERM is ignored if you are using xlbd.

APT_SHOW_LIBLOAD

If set, dumps a message to stdout every time a library is loaded. This can be useful for testing/verifying the right library is being loaded. Note that the message is output to stdout, NOT to the error log.

Decimal support

APT_DECIMAL_INTERM_PRECISION

Specifies the default maximum precision value for any decimal intermediate variables required in calculations. Default value is 38.

APT_DECIMAL_INTERM_SCALE

Specifies the default scale value for any decimal intermediate variables required in calculations. Default value is 10.

APT_DECIMAL_INTERM_ROUND_MODE

Specifies the default rounding mode for any decimal intermediate variables required in calculations. The default is round_inf.

Disk I/O

These environment variables are all concerned with when and how WebSphere DataStage parallel jobs write information to disk.

APT_BUFFER_DISK_WRITE_INCREMENT

For systems where small to medium bursts of I/O are not desirable, the default 1MB write to disk size chunk size might be too small. APT_BUFFER_DISK_WRITE_INCREMENT controls this and can be set larger than 1048576 (1 MB). The setting might not exceed *max_memory* * 2/3.

APT_CONSISTENT_BUFFERIO_SIZE

Some disk arrays have read ahead caches that are only effective when data is read repeatedly in like-sized chunks. Setting APT_CONSISTENT_BUFFERIO_SIZE=N will force stages to read data in chunks which are size N or a multiple of N.

APT_EXPORT_FLUSH_COUNT

Allows the export operator to flush data to disk more often than it typically does (data is explicitly flushed at the end of a job, although the OS might choose to do so more frequently). Set this variable to an integer which, in number of records, controls how often flushes should occur. Setting this value to a low number (such as 1) is useful for real time applications, but there is a small performance penalty associated with setting this to a low value.

APT_IO_MAP/APT_IO_NOMAP and APT_BUFFERIO_MAP/ APT_BUFFERIO_NOMAP

In many cases memory mapped I/O contributes to improved performance. In certain situations, however, such as a remote disk mounted via NFS, memory mapped I/O might cause significant performance problems. Setting the environment variables APT_IO_NOMAP and APT_BUFFERIO_NOMAP true will turn off this feature and sometimes affect performance. (AIX and HP-UX default to NOMAP. Setting APT_IO_MAP and APT_BUFFERIO_MAP true can be used to turn memory mapped I/O on for these platforms.)

APT_PHYSICAL_DATASET_BLOCK_SIZE

Specify the block size to use for reading and writing to a data set stage. The default is 128 KB.

General Job Administration

These environment variables are concerned with details about the running of WebSphere DataStage parallel jobs.

APT_CHECKPOINT_DIR

This environment variable is available in the WebSphere DataStage Administrator under the Parallel branch. By default, when running a job, WebSphere DataStage stores state information in the current working directory. Use APT_CHECKPOINT_DIR to specify another directory.

APT_CLOBBER_OUTPUT

This environment variable is available in the WebSphere DataStage Administrator under the Parallel branch. By default, if an output file or data set already exists, WebSphere DataStage issues an error and stops before overwriting it, notifying you of the name conflict. Setting this variable to any value permits WebSphere DataStage to overwrite existing files or data sets without a warning message.

APT_CONFIG_FILE

This environment variable is available in the WebSphere DataStage Administrator under the Parallel branch. Sets the path name of the configuration file. (You might want to include this as a job parameter, so that you can specify the configuration file at job run time).

APT_DISABLE_COMBINATION

This environment variable is available in the WebSphere DataStage Administrator under the Parallel branch. Globally disables operator combining. Operator combining is WebSphere DataStage's default behavior, in which two or more (in fact any number of) operators within a step are combined into one process where possible.

You might need to disable combining to facilitate debugging. Note that disabling combining generates more UNIX processes, and hence requires more system resources and memory. It also disables internal optimizations for job efficiency and run times.

APT_EXECUTION_MODE

This environment variable is available in the WebSphere DataStage Administrator under the Parallel branch. By default, the execution mode is parallel, with multiple processes. Set this variable to one of the following values to run an application in sequential execution mode:

- ONE_PROCESS one-process mode
- MANY_PROCESS many-process mode
- NO_SERIALIZE many-process mode, without serialization

In ONE_PROCESS mode:

- The application executes in a single UNIX process. You need only run a single debugger session and can set breakpoints anywhere in your code.
- Data is partitioned according to the number of nodes defined in the configuration file.
- Each operator is run as a subroutine and is called the number of times appropriate for the number of partitions on which it must operate.

In MANY_PROCESS mode the framework forks a new process for each instance of each operator and waits for it to complete rather than calling operators as subroutines.

In both cases, the step is run entirely on the Conductor node rather than spread across the configuration.

NO_SERIALIZE mode is similar to MANY_PROCESS mode, but the WebSphere DataStage persistence mechanism is not used to load and save objects. Turning off persistence might be useful for tracking errors in derived C++ classes.

APT_ORCHHOME

Must be set by all WebSphere DataStage users to point to the top-level directory of the WebSphere DataStage parallel engine installation.

APT_STARTUP_SCRIPT

As part of running an application, WebSphere DataStage creates a remote shell on all WebSphere DataStage processing nodes on which the job runs. By default, the remote shell is given the same environment as the shell from which WebSphere DataStage is invoked. However, you can write an optional *startup* shell script to modify the shell configuration of one or more processing nodes. If a startup script exists, WebSphere DataStage runs it on remote shells before running your application.

APT_STARTUP_SCRIPT specifies the script to be run. If it is not defined, WebSphere DataStage searches ./startup.apt, \$APT_ORCHHOME/etc/startup.apt and \$APT_ORCHHOME/etc/startup, in that order. APT_NO_STARTUP_SCRIPT disables running the startup script.

APT_NO_STARTUP_SCRIPT

Prevents WebSphere DataStage from executing a startup script. By default, this variable is not set, and WebSphere DataStage runs the startup script. If this variable is set, WebSphere DataStage ignores the startup script. This might be useful when debugging a startup script. See also APT_STARTUP_SCRIPT.

APT_STARTUP_STATUS

Set this to cause messages to be generated as parallel job startup moves from phase to phase. This can be useful as a diagnostic if parallel job startup is failing.

APT_THIN_SCORE

Setting this variable decreases the memory usage of steps with 100 operator instances or more by a noticeable amount. To use this optimization, set APT_THIN_SCORE=1 in your environment. There are no performance benefits in setting this variable unless you are running out of real memory at some point in your flow or the additional memory is useful for sorting or buffering. This variable does not affect any specific operators which consume large amounts of memory, but improves general parallel job memory handling.

Job Monitoring

These environment variables are concerned with the Job Monitor on WebSphere DataStage.

APT_MONITOR_SIZE

This environment variable is available in the WebSphere DataStage Administrator under the Parallel branch. Determines the minimum number of records the WebSphere DataStage Job Monitor reports. The default is 5000 records.

APT_MONITOR_TIME

This environment variable is available in the WebSphere DataStage Administrator under the Parallel branch. Determines the minimum time interval in seconds for generating monitor information at runtime. The default is 5 seconds. This variable takes precedence over APT_MONITOR_SIZE.

APT_NO_JOBMON

Turn off job monitoring entirely.

APT_PERFORMANCE_DATA

Set this variable to turn on performance data output generation. APT_PERFORMANCE_DATA can be either set with no value, or be set to a valid path which will be used as the default location for performance data output.

Look up support

APT_LUTCREATE_MMAP

This is only valid on TRU64 systems. Set this to force lookup tables to be created using memory mapped files. By default on TRU64 lookup table creation is done in memory created using malloc. This is for performance reasons. If, for some reason, malloced memory is not desirable, this variable can be used to switch over the memory mapped files.

APT_LUTCREATE_NO_MMAP

Set this to force lookup tables to be created using malloced memory. By default lookup table creation is done using memory mapped files. There might be situations, depending on the OS configuration or file system, where writing to memory mapped files causes poor performance. In these situations this variable can be set so that malloced memory is used, which should boost performance.

Miscellaneous

APT_COPY_TRANSFORM_OPERATOR

If set, distributes the shared object file of the sub-level transform operator and the shared object file of user-defined functions (not extern functions) via distribute-component in a non-NFS MPP.

APT_DATE_CENTURY_BREAK_YEAR

Four digit year which marks the century two-digit dates belong to. It is set to 1900 by default.

APT_EBCDIC_VERSION

Certain operators, including the import and export operators, support the “ebcdic” property specifying that field data is represented in the EBCDIC character set. The APT_EBCDIC_VERSION variable indicates the specific EBCDIC character set to use. Legal values are:

HP use the EBCDIC character set supported by HP terminals (this is the default setting, except on USS installations).

IBM Use the EBCDIC character set supported by IBM 3780 terminals

ATT Use the EBCDIC character set supported by AT&T terminals.

USS Use the IBM 1047 EBCDIC character set (this is the default setting on USS installations).

IBM037

Use the IBM 037 EBCDIC character set.

IBM500

Use the IBM 500 EBCDIC character set.

If the value of the variable is HP, IBM, ATT, or USS, then EBCDIC data is internally converted to/from 7-bit ASCII. If the value is IBM037 or IBM500, internal conversion is between EBCDIC and ISO-8859-1 (the 8-bit Latin-1 superset of ASCII, with accented character support).

APT_IMPEXP_ALLOW_ZERO_LENGTH_FIXED_NULL

When set, allows zero length null_field value with fixed length fields. This should be used with care as poorly formatted data will cause incorrect results. By default a zero length null_field value will cause an error.

APT_IMPORT_REJECT_STRING_FIELD_OVERRUNS

When set, WebSphere DataStage will reject any string or ustring fields read that go over their fixed size. By default these records are truncated.

APT_INSERT_COPY_BEFORE MODIFY

When defined, turns on automatic insertion of a copy operator before any modify operator (WARNING: if this variable is not set and the operator immediately preceding 'modify' in the data flow uses a modify adapter, the 'modify' operator will be removed from the data flow).

Only set this if you write your own custom operators AND use modify within those operators.

APT_OLD_BOUNDED_LENGTH

Some parallel datasets generated with WebSphere DataStage 7.0.1 and later releases require more disk space when the columns are of type VarChar when compared to 7.0. This is due to changes added for performance improvements for bounded length VarChars in 7.0.1.

Set APT_OLD_BOUNDED_LENGTH to any value to revert to pre-7.0.1 storage behavior when using bounded length varchars. Setting this variable can have adverse performance effects. The preferred and more performant solution is to use unbounded length VarChars (don't set any length) for columns where the maximum length is rarely used, rather than set this environment variable.

APT_OPERATOR_REGISTRY_PATH

Used to locate operator .apt files, which define what operators are available and which libraries they are found in.

APT_PM_NO_SHARED_MEMORY

By default, shared memory is used for local connections. If this variable is set, named pipes rather than shared memory are used for local connections. If both APT_PM_NO_NAMED_PIPES and APT_PM_NO_SHARED_MEMORY are set, then TCP sockets are used for local connections.

APT_PM_NO_NAMED_PIPES

Specifies not to use named pipes for local connections. Named pipes will still be used in other areas of WebSphere DataStage, including subprocs and setting up of the shared memory transport protocol in the process manager.

APT_PM_SOFT_KILL_WAIT

Delay between SIGINT and SIGKILL during abnormal job shutdown. Gives time for processes to run cleanups if they catch SIGINT. Defaults to ZERO.

APT_PM_STARTUP_CONCURRENCY

Setting this to a small integer determines the number of simultaneous section leader startups to be allowed. Setting this to 1 forces sequential startup. The default is defined by SOMAXCONN in sys/socket.h (currently 5 for Solaris, 10 for AIX).

APT_RECORD_COUNTS

Causes WebSphere DataStage to print, for each operator Player, the number of records consumed by getRecord() and produced by putRecord(). Abandoned input records are not necessarily accounted for. Buffer operators do not print this information.

APT_SAVE_SCORE

Sets the name and path of the file used by the performance monitor to hold temporary score data. The path must be visible from the host machine. The performance monitor creates this file, therefore it need not exist when you set this variable.

APT_SHOW_COMPONENT_CALLS

This forces WebSphere DataStage to display messages at job check time as to which user overloadable functions (such as checkConfig and describeOperator) are being called. This will not produce output at runtime and is not guaranteed to be a complete list of all user-overloadable functions being called, but an effort is made to keep this synchronized with any new virtual functions provided.

APT_STACK_TRACE

This variable controls the number of lines printed for stack traces. The values are:

- unset. 10 lines printed
- 0. infinite lines printed
- N. N lines printed
- none. no stack trace

The last setting can be used to disable stack traces entirely.

APT_WRITE_DS_VERSION

By default, WebSphere DataStage saves data sets in the Orchestrate Version 4.1 format. APT_WRITE_DS_VERSION lets you save data sets in formats compatible with previous versions of Orchestrate.

The values of APT_WRITE_DS_VERSION are:

- v3_0. Orchestrate Version 3.0
- v3. Orchestrate Version 3.1.x
- v4. Orchestrate Version 4.0
- v4_0_3. Orchestrate Version 4.0.3 and later versions up to but not including Version 4.1
- v4_1. Orchestrate Version 4.1 and later versions through and including Version 4.6

OSH_PRELOAD_LIBS

Specifies a colon-separated list of names of libraries to be loaded before any other processing. Libraries containing custom operators must be assigned to this variable or they must be registered. For example, in Korn shell syntax:

```
$ export OSH_PRELOAD_LIBS="orchlib1:orchlib2:mylib1"
```

Network

These environment variables are concerned with the operation of WebSphere DataStage parallel jobs over a network.

APT_IO_MAXIMUM_OUTSTANDING

Sets the amount of memory, in bytes, allocated to a WebSphere DataStage job on every physical node for network communications. The default value is 2097152 (2MB).

When you are executing many partitions on a single physical node, this number might need to be increased.

APT_IOMGR_CONNECT_ATTEMPTS

Sets the number of attempts for a TCP connect in case of a connection failure. This is necessary only for jobs with a high degree of parallelism in an MPP environment. The default value is 2 attempts (1 retry after an initial failure).

APT_PM_CONDUCTOR_HOSTNAME

The network name of the processing node from which you invoke a job should be included in the configuration file as either a node or a fastname. If the network name is not included in the configuration file, WebSphere DataStage users must set the environment variable APT_PM_CONDUCTOR_HOSTNAME to the name of the node invoking the WebSphere DataStage job.

APT_PM_NO_TCPIP

This turns off use of UNIX sockets to communicate between player processes at runtime. If the job is being run in an MPP (non-shared memory) environment, do not set this variable, as UNIX sockets are your only communications option.

APT_PM_NODE_TIMEOUT

This controls the number of seconds that the conductor will wait for a section leader to start up and load a score before deciding that something has failed. The default for starting a section leader process is 30. The default for loading a score is 120.

APT_PM_SHOWRSH

Displays a trace message for every call to RSH.

APT_PM_STARTUP_PORT

Use this environment variable to specify the port number from which the parallel engine will start looking for TCP/IP ports.

By default, WebSphere DataStage will start look at port 10000. If you know that ports in this range are used by another application, set APT_PM_STARTUP_PORT to start at a different level. You should check the /etc/services file for reserved ports.

APT_PM_USE_RSH_LOCALLY

If set, startup will use rsh even on the conductor node.

NLS Support

These environment variables are concerned with WebSphere DataStage's implementation of NLS.

Note: You should not change the settings of any of these environment variables other than APT_COLLATION_STRENGTH if NLS is enabled on your server.

APT_COLLATION_SEQUENCE

This variable is used to specify the global collation sequence to be used by sorts, compares, and so on. This value can be overridden at the stage level.

APT_COLLATION_STRENGTH

Set this to specify the defines the specifics of the collation algorithm. This can be used to ignore accents, punctuation or other details.

It is set to one of Identical, Primary, Secondary, Tertiary, or Quartenary. Setting it to Default unsets the environment variable.

http://oss.software.ibm.com/icu/userguide/Collate_Concepts.html

APT_ENGLISH_MESSAGES

If set to 1, outputs every message issued with its English equivalent.

APT_IMPEXP_CHARSET

Controls the character encoding of ustring data imported and exported to and from WebSphere DataStage, and the record and field properties applied to ustring fields. Its syntax is:

APT_IMPEXP_CHARSET icu_character_set

APT_INPUT_CHARSET

Controls the character encoding of data input to schema and configuration files. Its syntax is:

APT_INPUT_CHARSET icu_character_set

APT_OS_CHARSET

Controls the character encoding WebSphere DataStage uses for operating system data such as the names of created files and the parameters to system calls. Its syntax is:

APT_OS_CHARSET icu_character_set

APT_OUTPUT_CHARSET

Controls the character encoding of WebSphere DataStage output messages and operators like peek that use the error logging system to output data input to the osh parser. Its syntax is:

APT_OUTPUT_CHARSET icu_character_set

APT_STRING_CHARSET

Controls the character encoding WebSphere DataStage uses when performing automatic conversions between string and ustring fields. Its syntax is:

APT_STRING_CHARSET icu_character_set

Oracle Support

These environment variables are concerned with the interaction between WebSphere DataStage and Oracle databases.

APT_ORACLE_LOAD_DELIMITED

If this is defined, the orawrite operator creates delimited records when loading into Oracle sqldr. This method preserves leading and trailing blanks within string fields (VARCHARS in the database). The value of this variable is used as the delimiter. If this is defined without a value, the default delimiter is a comma. Note that you cannot load a string which has embedded double quotes if you use this.

APT_ORACLE_LOAD_OPTIONS

You can use the environment variable APT_ORACLE_LOAD_OPTIONS to control the options that are included in the Oracle load control file. You can load a table with indexes without using the Index Mode or Disable Constraints properties by setting the APT_ORACLE_LOAD_OPTIONS environment variable appropriately. You need to set the Direct option or the PARALLEL option to FALSE, for example:

```
APT_ORACLE_LOAD_OPTIONS='OPTIONS(DIRECT=FALSE,PARALLEL=TRUE)'
```

In this example the stage would still run in parallel, however, since DIRECT is set to FALSE, the conventional path mode rather than the direct path mode would be used.

If loading index organized tables (IOTs), you should not set both DIRECT and PARALLEL to true as direct parallel path load is not allowed for IOTs.

APT_ORACLE_NO_OPS

Set this if you do not have Oracle Parallel server installed on an AIX system. It disables the OPS checking mechanism on the Oracle Enterprise stage.

APT_ORACLE_PRESERVE_BLANKS

Set this to set the PRESERVE BLANKS option in the control file. This preserves leading and trailing spaces. When PRESERVE BLANKS is not set Oracle removes the spaces and considers fields with only spaces to be NULL values.

APT_ORA_IGNORE_CONFIG_FILE_PARALLELISM

By default WebSphere DataStage determines the number of processing nodes available for a parallel write to Oracle from the configuration file. Set APT_ORA_IGNORE_CONFIG_FILE_PARALLELISM to use the number of data files in the destination table's tablespace instead.

APT_ORA_WRITE_FILES

Set this to prevent the invocation of the Oracle loader when write mode is selected on an Oracle Enterprise destination stage. Instead, the sqldr commands are written to a file, the name of which is specified by this environment variable. The file can be invoked once the job has finished to run the loaders sequentially. This can be useful in tracking down export and pipe-safety issues related to the loader.

APT_ORAUPSERT_COMMIT_ROW_INTERVAL APT_ORAUPSERT_COMMIT_TIME_INTERVAL

These two environment variables work together to specify how often target rows are committed when using the Upsert method to write to Oracle.

Commits are made whenever the time interval period has passed or the row interval is reached, whichever comes first. By default, commits are made every 2 seconds or 5000 rows.

Partitioning

The following environment variables are concerned with how WebSphere DataStage automatically partitions data.

APT_NO_PART_INSERTION

WebSphere DataStage automatically inserts partition components in your application to optimize the performance of the stages in your job. Set this variable to prevent this automatic insertion.

APT_PARTITION_COUNT

Read only. WebSphere DataStage sets this environment variable to the number of partitions of a stage. The number is based both on information listed in the configuration file and on any constraints applied to the stage. The number of partitions is the degree of parallelism of a stage. For example, if a stage executes on two processing nodes, APT_PARTITION_COUNT is set to 2.

You can access the environment variable APT_PARTITION_COUNT to determine the number of partitions of the stage from within:

- an operator wrapper
- a shell script called from a wrapper
- getenv() in C++ code
- sysget() in the SAS language.

APT_PARTITION_NUMBER

Read only. On each partition, WebSphere DataStage sets this environment variable to the index number (0, 1, ...) of this partition within the stage. A subprocess can then examine this variable when determining which partition of an input file it should handle.

Reading and writing files

These environment variables are concerned with reading and writing files.

APT_DELIMITED_READ_SIZE

By default, the WebSphere DataStage will read ahead 500 bytes to get the next delimiter. For streaming inputs (socket, FIFO, and so on) this is sub-optimal, since the WebSphere DataStage might block (and not output any records). WebSphere DataStage, when reading a delimited record, will read this many bytes (minimum legal value for this is 2) instead of 500. If a delimiter is NOT available within N bytes, N will be incremented by a factor of 2 (when this environment variable is not set, this changes to 4).

APT_FILE_IMPORT_BUFFER_SIZE

The value in kilobytes of the buffer for reading in files. The default is 128 (that is, 128 KB). It can be set to values from 8 upward, but is clamped to a minimum value of 8. That is, if you set it to a value less than 8, then 8 is used. Tune this upward for long-latency files (typically from heavily loaded file servers).

APT_FILE_EXPORT_BUFFER_SIZE

The value in kilobytes of the buffer for writing to files. The default is 128 (that is, 128 KB). It can be set to values from 8 upward, but is clamped to a minimum value of 8. That is, if you set it to a value less than 8, then 8 is used. Tune this upward for long-latency files (typically from heavily loaded file servers).

APT_IMPORT_PATTERNUSESFILESET

When this is set, WebSphere DataStage will turn any file pattern into a fileset before processing the files. This allows the files to be processed in parallel as opposed to sequentially. By default file pattern will cat the files together to be used as the input.

APT_MAX_DELIMITED_READ_SIZE

By default, when reading, WebSphere DataStage will read ahead 500 bytes to get the next delimiter. If it is not found, WebSphere DataStage looks ahead $4 \times 500 = 2000$ (1500 more) bytes, and so on (4X) up to 100,000 bytes. This variable controls the upper bound which is by default 100,000 bytes. Note that this variable should be used instead of APT_DELIMITED_READ_SIZE when a larger than 500 bytes read-ahead is desired.

APT_PREVIOUS_FINAL_DELIMITER_COMPATIBLE

Set this to revert to the pre-release 7.5 behavior of the final delimiter whereby, when writing data, a space is inserted after every field in a record including the last one. (The new behavior is that a space is written after every field except the last one).

APT_STRING_PADCHAR

Overrides the pad character of 0x0 (ASCII null), used by default when WebSphere DataStage extends, or pads, a string field to a fixed length.

Reporting

These environment variables are concerned with various aspects of WebSphere DataStage jobs reporting their progress.

APT_DUMP_SCORE

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Reporting**. Configures WebSphere DataStage to print a report showing the operators, processes, and data sets in a running job.

APT_ERROR_CONFIGURATION

Controls the format of WebSphere DataStage output messages.

Note: Changing these settings can seriously interfere with WebSphere DataStage logging.

This variable's value is a comma-separated list of keywords (see table below). Each keyword enables a corresponding portion of the message. To disable that portion of the message, precede it with a "!".

Default formats of messages displayed by WebSphere DataStage include the keywords severity, moduleId, errorIndex, timestamp, opid, and message.

The following table lists keywords, the length (in characters) of the associated components in the message, and the keyword's meaning. The characters "##" precede all messages. The keyword lengthprefix appears in three locations in the table. This single keyword controls the display of all length prefixes.

Keyword	Length	Meaning
severity	1	Severity indication: F, E, W, or I.
vseverity	7	Verbose description of error severity (Fatal, Error, Warning, Information).
jobid	3	The job identifier of the job. This allows you to identify multiple jobrunning at once. The default job identifier is 0.
moduleId	4	The module identifier. For WebSphere DataStage-defined messages, this value is a four byte string beginning with T. For user-defined messages written to the error log, this string is USER. For all outputs from a subprocess, the string is USBP.
errorIndex	6	The index of the message specified at the time the message was written to the error log.
timestamp	13	The message time stamp. This component consists of the string "HH:MM:SS(SEQ)", at the time the message was written to the error log. Messages generated in the same second have ordered sequence numbers.
ipaddr	15	The IP address of the processing node generating the message. This 15-character string is in octet form, with individual octets zero filled, for example, 104.032.007.100.
lengthprefix	2	Length in bytes of the following field.
nodename	variable	The node name of the processing node generating the message.
lengthprefix	2	Length in bytes of the following field.

Keyword	Length	Meaning
opid	variable	The string <main_program> for error messages originating in your main program (outside of a step or within the APT_Operator::describeOperator() override). The string <node_nodename> representing system error messages originating on a node, where <i>nodename</i> is the name of the node. The operator originator identifier, represented by "ident, partition_number", for errors originating within a step. This component identifies the instance of the operator that generated the message. <i>ident</i> is the operator name (with the operator index in parenthesis if there is more than one instance of it). <i>partition_number</i> defines the partition number of the operator issuing the message.
lengthprefix	5	Length, in bytes, of the following field. Maximum length is 15 KB.
message	variable	Error text.
	1	Newline character

APT_MSG_FILELINE

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Reporting** branch. Set this to have WebSphere DataStage log extra internal information for parallel jobs.

APT_PM_PLAYER_MEMORY

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Reporting** branch. Setting this variable causes each player process to report the process heap memory allocation in the job log when returning.

APT_PM_PLAYER_TIMING

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Reporting** branch. Setting this variable causes each player process to report its call and return in the job log. The message with the return is annotated with CPU times for the player process.

APT_RECORD_COUNTS

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Reporting** branch. Causes WebSphere DataStage to print to the job log, for each operator player, the number of records input and output. Abandoned input records are not necessarily accounted for. Buffer operators do not print this information.

OSH_DUMP

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Reporting** branch. If set, it causes WebSphere DataStage to put a verbose description of a job in the job log before attempting to execute it.

OSH_ECHO

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Reporting** branch. If set, it causes WebSphere DataStage to echo its job specification to the job log after the shell has expanded all arguments.

OSH_EXPLAIN

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Reporting** branch. If set, it causes WebSphere DataStage to place a terse description of the job in the job log before attempting to run it.

OSH_PRINT_SCHEMAS

This environment variable is available in the WebSphere DataStage Administrator under the **Parallel → Reporting** branch. If set, it causes WebSphere DataStage to print the record schema of all data sets and the interface schema of all operators in the job log.

SAS Support

These environment variables are concerned with WebSphere DataStage interaction with SAS.

APT_HASH_TO_SASHASH

The WebSphere DataStage hash partitioner contains support for hashing SAS data. In addition, WebSphere DataStage provides the sashash partitioner which uses an alternative non-standard hashing algorithm. Setting the APT_HASH_TO_SASHASH environment variable causes all appropriate instances of hash to be replaced by sashash. If the APT_NO_SAS_TRANSFORMS environment variable is set, APT_HASH_TO_SASHASH has no affect.

APT_NO_SASOUT_INSERT

This variable selectively disables the sasout operator insertions. It maintains the other SAS-specific transformations.

APT_NO_SAS_TRANSFORMS

WebSphere DataStage automatically performs certain types of SAS-specific component transformations, such as inserting an sasout operator and substituting sasRoundRobin for RoundRobin. Setting the APT_NO_SAS_TRANSFORMS variable prevents WebSphere DataStage from making these transformations.

APT_SAS_ACCEPT_ERROR

When a SAS procedure causes SAS to exit with an error, this variable prevents the SAS-interface operator from terminating. The default behavior is for WebSphere DataStage to terminate the operator with an error.

APT_SAS_CHARSET

When the -sas_cs option of a SAS-interface operator is not set and a SAS-interface operator encounters a ustring, WebSphere DataStage interrogates this variable to determine what character set to use. If this variable is not set, but APT_SAS_CHARSET_ABORT is set, the operator will abort; otherwise the -impexp_charset option or the APT_IMPEXP_CHARSET environment variable is accessed. Its syntax is:

```
APT_SAS_CHARSET icu_character_set | SAS_DBCSLANG
```

APT_SAS_CHARSET_ABORT

Causes a SAS-interface operator to abort if WebSphere DataStage encounters a ustring in the schema and neither the -sas_cs option nor the APT_SAS_CHARSET environment variable is set.

APT_SAS_COMMAND

Overrides the \$PATH directory for SAS with an absolute path to the basic SAS executable. An example path is:

```
/usr/local/sas/sas8.2/sas
```

APT_SASINT_COMMAND

Overrides the \$PATH directory for SAS with an absolute path to the International SAS executable. An example path is:

```
/usr/local/sas/sas8.2int/dbcs/sas
```

APT_SAS_DEBUG

Set this to set debug in the SAS process coupled to the SAS stage. Messages appear in the SAS log, which might then be copied into the WebSphere DataStage log. Use APT_SAS_DEBUG=1, APT_SAS_DEBUG_IO=1, and APT_SAS_DEBUG_VERBOSE=1 to get all debug messages.

APT_SAS_DEBUG_IO

Set this to set input/output debug in the SAS process coupled to the SAS stage. Messages appear in the SAS log, which might then be copied into the WebSphere DataStage log.

APT_SAS_DEBUG_LEVEL

Its syntax is:

```
APT_SAS_DEBUG_LEVEL=[0-3]
```

Specifies the level of debugging messages to output from the SAS driver. The values of 1, 2, and 3 duplicate the output for the -debug option of the SAS operator:

no, yes, and verbose.

APT_SAS_DEBUG_VERBOSE

Set this to set verbose debug in the SAS process coupled to the SAS stage. Messages appear in the SAS log, which might then be copied into the WebSphere DataStage log.

APT_SAS_NO_PSDS_USTRING

Set this to prevent WebSphere DataStage from automatically converting SAS char types to ustrings in an SAS parallel data set.

APT_SAS_S_ARGUMENT

By default, WebSphere DataStage executes SAS with -s 0. When this variable is set, its value is used instead of 0. Consult the SAS documentation for details.

APT_SAS_SCHEMA_SOURCE_DUMP

When using SAS Schema Source, causes the command line to be written to the log when executing SAS. You use it to inspect the data contained in a -schemaSource. Set this if you are getting an error when specifying the SAS data set containing the schema source.

APT_SAS_SHOW_INFO

Displays the standard SAS output from an import or export transaction. The SAS output is normally deleted since a transaction is usually successful.

APT_SAS_TRUNCATION

Its syntax is:

```
APT_SAS_TRUNCATION ABORT | NULL | TRUNCATE
```

Because a ustring of n characters does not fit into n characters of a SAS char value, the ustring value must be truncated beyond the space pad characters and \0.

The sasin and sas operators use this variable to determine how to truncate a ustring value to fit into a SAS char field. TRUNCATE, which is the default, causes the ustring to be truncated; ABORT causes the operator to abort; and NULL exports a null field. For NULL and TRUNCATE, the first five occurrences for each column cause an information message to be issued to the log.

Sorting

The following environment variables are concerned with how WebSphere DataStage automatically sorts data.

APT_NO_SORT_INSERTION

WebSphere DataStage automatically inserts sort components in your job to optimize the performance of the operators in your data flow. Set this variable to prevent this automatic insertion.

APT_SORT_INSERTION_CHECK_ONLY

When sorts are inserted automatically by WebSphere DataStage, if this is set, the sorts will just check that the order is correct, they won't actually sort. This is a better alternative to shutting partitioning and sorting off insertion off using APT_NO_PART_INSERTION and APT_NO_SORT_INSERTION.

Sybase support

These environment variables are concerned with setting up access to Sybase databases from WebSphere DataStage.

APT_SYBASE_NULL_AS_EMPTY

Set APT_SYBASE_NULL_AS_EMPTY to 1 to extract null values as empty, and to load null values as " " when reading or writing an IQ database.

APT_SYBASE_PRESERVE_BLANKS

Set APT_SYBASE_PRESERVE_BLANKS to preserve trailing blanks while writing to an IQ database.

Teradata Support

The following environment variables are concerned with WebSphere DataStage interaction with Teradata databases.

APT_TERA_64K_BUFFERS

WebSphere DataStage assumes that the terawrite operator writes to buffers whose maximum size is 32 KB. Enable the use of 64 KB buffers by setting this variable. The default is that it is not set.

APT_TERA_NO_ERR_CLEANUP

Setting this variable prevents removal of error tables and the partially written target table of a terawrite operation that has not successfully completed. Set this variable for diagnostic purposes only. In some cases, setting this variable forces completion of an unsuccessful write operation.

APT_TERA_NO_SQL_CONVERSION

Set this to prevent the SQL statements you are generating from being converted to the character set specified for your stage (character sets can be specified at project, job, or stage level). The SQL statements are converted to LATIN1 instead.

APT_TERA_NO_PERM_CHECKS

Set this to bypass permission checking on the several system tables that need to be readable for the load process. This can speed up the start time of the load process slightly.

APT_TERA_SYNC_DATABASE

Specifies the database used for the terasync table. By default, the database used for the terasync table is specified as part of APT_TERA_SYNC_USER. If you want the database to be different, set this variable. You must then give APT_TERA_SYNC_USER read and write permission for this database.

APT_TERA_SYNC_PASSWORD

Specifies the password for the user identified by APT_TERA_SYNC_USER.

APT_TERA_SYNC_USER

Specifies the user that creates and writes to the terasync table.

Transport Blocks

The following environment variables are all concerned with the block size used for the internal transfer of data as jobs run. Some of the settings only apply to fixed length records. The following variables are used only for fixed-length records.:

- APT_MIN_TRANSPORT_BLOCK_SIZE
- APT_MAX_TRANSPORT_BLOCK_SIZE
- APT_DEFAULT_TRANSPORT_BLOCK_SIZE
- APT_LATENCY_COEFFICIENT
- APT_AUTO_TRANSPORT_BLOCK_SIZE

APT_AUTO_TRANSPORT_BLOCK_SIZE

This environment variable is available in the WebSphere DataStage Administrator, under the Parallel category. When set, Orchestrate calculates the block size for transferring data internally as jobs run. It uses this algorithm:

```
if (recordSize * APT_LATENCY_COEFFICIENT
< APT_MIN_TRANSPORT_BLOCK_SIZE)
blockSize = minAllowedBlockSize
else if (recordSize * APT_LATENCY_COEFFICIENT
> APT_MAX_TRANSPORT_BLOCK_SIZE)
blockSize = maxAllowedBlockSize
else blockSize = recordSize * APT_LATENCY_COEFFICIENT
```

APT_LATENCY_COEFFICIENT

Specifies the number of writes to a block which transfers data between players. This variable allows you to control the latency of data flow through a step. The default value is 5. Specify a value of 0 to have a record transported immediately. This is only used for fixed length records.

Note: Many operators have a built-in latency and are not affected by this variable.

APT_DEFAULT_TRANSPORT_BLOCK_SIZE

Specify the default block size for transferring data between players. It defaults to 131072 (128 KB).

APT_MAX_TRANSPORT_BLOCK_SIZE/ APT_MIN_TRANSPORT_BLOCK_SIZE

Specify the minimum and maximum allowable block size for transferring data between players.

APT_MIN_TRANSPORT_BLOCK_SIZE cannot be less than 8192 which is its default value.

APT_MAX_TRANSPORT_BLOCK_SIZE cannot be greater than 1048576 which is its default value. These variables are only meaningful when used in combination with APT_LATENCY_COEFFICIENT and APT_AUTO_TRANSPORT_BLOCK_SIZE.

Guide to setting environment variables

This section gives some guide as to which environment variables should be set in what circumstances.

Environment variable settings for all jobs

We recommend that you set the following environment variables for all jobs:

- APT_CONFIG_FILE
- APT_DUMP_SCORE
- APT_RECORD_COUNTS

Optional environment variable settings

We recommend setting the following environment variables as needed on a per-job basis. These variables can be used to turn the performance of a particular job flow, to assist in debugging, and to change the default behavior of specific parallel job stages.

Performance tuning

- APT_BUFFER_MAXIMUM_MEMORY
- APT_BUFFER_FREE_RUN
- TMPDIR. This defaults to /tmp. It is used for miscellaneous internal temporary data, including FIFO queues and Transformer temporary storage. As a minor optimization, it can be better to ensure that it is set to a file system separate to the WebSphere DataStage install directory.

Job flow debugging

- OSH_PRINT_SCHEMAS
- APT_DISABLE_COMBINATION
- APT_PM_PLAYER_TIMING
- APT_PM_PLAYER_MEMORY
- APT_BUFFERING_POLICY

Job flow design

- APT_STRING_PADCHAR

Chapter 7. Operators

The parallel job stages are built on operators. These topics describe those operators and is intended for knowledgeable Orchestrate users.

The first section describes how WebSphere DataStage stages map to operators. Subsequent sections are an alphabetical listing and description of operators. Some operators are part of a library of related operators, and each of these has its own topic as follows:

- The Import/Export Library
- The Partitioning Library
- The Collection Library
- The Restructure Library
- The Sorting Library
- The Join Library
- The ODBC Interface Library
- The SAS Interface Library
- The Oracle Interface Library
- The DB2 Interface Library
- The Informix Interface Library
- The Sybase Interface Library
- The SQL Server Interface Library
- The iWay Interface Library

In these descriptions, the term WebSphere DataStage refers to the parallel engine that WebSphere DataStage uses to execute the operators.

Stage to Operator Mapping

There is not a one to one mapping between WebSphere DataStage stages and operators. Some stages are based on a range of related operators and which one is used depends on the setting of the stage's properties. All of the stages can include common operators such as partition and sort depending on how they are used in a job. Table 7 shows the mapping between WebSphere DataStage stages and operators. Where a stage uses an operator with a particular option set, this option is also given. The WebSphere DataStage stages are listed by palette category in the same order in which they are described in the *WebSphere DataStage Parallel Job Developer Guide*.

Table 7. Stage to Operator Mapping

WebSphere DataStage Stage	Operator	Options (where applicable)	Comment
File Set	-	-	Represents a permanent data set
Sequential File	Import Operator	- file -filepattern	
	Export Operator	-file	
File Set	Import Operator	-fileset	
	Export Operator	-fileset	
Lookup File Set	Lookup Operator	-createOnly	

Table 7. Stage to Operator Mapping (continued)

WebSphere DataStage Stage	Operator	Options (where applicable)	Comment
External Source	Import Operator	-source -sourcelist	
External Target	Export Operator	-destination -destinationlist	
Complex Flat File	Import Operator		
Transformer	Transform Operator		
BASIC Transformer			Represents server job transformer stage (gives access to BASIC transforms)
Aggregator	Group Operator		
Join	fullouterjoin Operator innerjoin Operator leftouterjoin Operator rightouterjoin Operator		
Merge	Merge Operator		
Lookup	Lookup Operator		
	The oralookup Operator		for direct lookup in Oracle table ('sparse' mode)
	The db2lookup Operator		for direct lookup in DB2 table ('sparse' mode)
	The sybaselookup Operator		for direct lookup in table accessed via iWay ('sparse' mode)
	The sybaselookup Operator		for direct lookup in Sybase table ('sparse' mode)
	The sqlsvrlookup Operator		
Funnel	Funnel Operators		
	Sortfunnel Operator		
	Sequence Operator		
Sort	The psort Operator		
	The tsort Operator		
Remove Duplicates	Remdup Operator		
Compress	Pcompress Operator	-compress	
Expand	Pcompress Operator	-expand	
Copy	Generator Operator		
Modify	Modify Operator		
Filter	Filter Operator		
External Filter	-	-	Any executable command line that acts as a filter
Change Capture	Changecapture Operator		
Change Apply	Changeapply Operator		
Difference	Diff Operator		

Table 7. Stage to Operator Mapping (continued)

WebSphere DataStage Stage	Operator	Options (where applicable)	Comment
Compare	Compare Operator		
Encode	Encode Operator	-encode	
Decode	Encode Operator	-decode	
Switch	Switch Operator		
Generic	-	-	Any operator
Surrogate Key	Surrogate key operator		
Column Import	The field_import Operator		
Column Export	The field_export Operator		
Make Subrecord	The makesubrec Operator		
Split Subrecord	The splitsubrec Operator		
Combine records	The aggtorec Operator		
Promote subrecord	The makesubrec Operator		
Make vector	The makevect Operator		
Split vector	The splitvect Operator		
Head	Head Operator		
Tail	Tail Operator		
Sample	Sample Operator		
Peek	Peek Operator		
Row Generator	Generator Operator		
Column generator	Generator Operator		
Write Range Map	Writerangemap Operator		
SAS Parallel Data Set	-	-	Represents Orchestrate parallel SAS data set.
SAS	The sas Operator		
	The sasout Operator		
	The sasin Operator		
	The sascontents Operator		
DB2/UDB Enterprise	The db2read Operator		
	The db2write and db2load Operators		
	The db2upsert Operator		
	The db2lookup Operator		For in-memory ('normal') lookups (
Oracle Enterprise	The oraread Operator		
	The oraupsert Operator		
	The orawrite Operator		
	The oralookup Operator		For in-memory ('normal') lookups
Informix Enterprise	The hplread operator		
	The hplwrite Operator		

Table 7. Stage to Operator Mapping (continued)

WebSphere DataStage Stage	Operator	Options (where applicable)	Comment
	The infxread Operator		
	The infxwrite Operator		
	The xpsread Operator		
	The xpswrite Operator		
Teradata	Teraread Operator		
	Terawrite Operator		
Sybase	The sybasereade Operator		
	The sybasewrite Operator		
	The sybaseupsert Operator		
	The sybaselookup Operator		For in-memory ('normal') lookups
SQL Server	The sqlsrvrread Operator		
	The sqlsrvrwrite Operator		
	The sqlsrvrupsert Operator		
	The sybaselookup Operator		For in-memory ('normal') lookups
iWay	The iwayread Operator		
	The iwaylookup Operator		

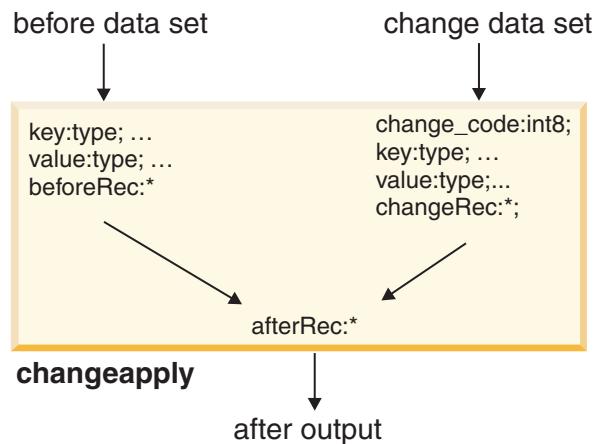
Changeapply operator

The changeapply operator takes the change data set output from the changecapture operator and applies the encoded change operations to a before data set to compute an after data set. If the before data set is identical to the before data set that was input to the changecapture operator, then the output after data set for changeapply is identical to the after data set that was input to the changecapture operator. That is:

```
change := changecapture(before, after)
after   := changeapply(before, change)
```

You use the companion operator changecapture to provide a data set that contains the changes in the before and after data sets.

Data flow diagram



changeapply: properties

Table 8. changeapply Operator Properties

Property	Value
Number of input data sets	2
Number of output data sets	1
Input interface schema	beforeRec:*, changeRec:*
Output interface schema	afterRec:*
Transfer behavior	changeRec:*->afterRec:*, dropping the change_code field; beforeRec:*->afterRec:* with type conversions
Input partitioning style	keys in same partition
Output partitioning style	keys in same partition
Preserve-partitioning flag in output data set	propagated
Composite operator	no

The before input to changeapply must have the same fields as the before input that was input to changecapture, and an automatic conversion must exist between the types of corresponding fields. In addition, results are only guaranteed if the contents of the before input to changeapply are identical (in value and record order in each partition) to the before input that was fed to changecapture, and if the keys are unique.

The change input to changeapply must have been output from changecapture without modification. Because preserve-partitioning is set on the change output of changecapture (under normal circumstances you should not override this), the changeapply operator has the same number of partitions as the changecapture operator. Additionally, both inputs of changeapply are designated as same partitioning by the operator logic.

The changeapply operator performs these actions for each change record:

- If the before keys come before the change keys in the specified sort order, the before record is consumed and transferred to the output; no change record is consumed. This is a copy.
- If the before keys are equal to the change keys, the behavior depends on the code in the change_code field of the change record:
 - Insert: The change record is consumed and transferred to the output; no before record is consumed.

If key fields are not unique, and there is more than one consecutive insert with the same key, then changeapply applies all the consecutive inserts before existing records. This record order might be different from the after data set given to changecapture.

- Delete: The value fields of the before and change records are compared. If they are not the same, the before record is consumed and transferred to the output; no change record is consumed (copy). If the value fields are the same or if ignoreDeleteValues is specified, the change and before records are both consumed; no record is transferred to the output.

If key fields are not unique, the value fields ensure that the correct record is deleted. If more than one record with the same keys have matching value fields, the first encountered is deleted. This might cause different record ordering than in the after data set given to the changecapture operator.

- Edit: The change record is consumed and transferred to the output; the before record is just consumed.

If key fields are not unique, then the first before record encountered with matching keys is edited. This might be a different record from the one that was edited in the after data set given to the changecapture operator, unless the -keepCopy option was used.

- Copy: The change record is consumed. The before record is consumed and transferred to the output.
- If the before keys come after the change keys, behavior also depends on the change_code field.
 - Insert: The change record is consumed and transferred to the output; no before record is consumed (the same as when the keys are equal).
 - Delete: A warning is issued and the change record is consumed; no record is transferred to the output; no before record is consumed.
 - Edit or Copy: A warning is issued and the change record is consumed and transferred to the output; no before record is consumed. This is an insert.
- If the before input of changeapply is identical to the before input of changecapture and either keys are unique or copy records are used, then the output of changeapply is identical to the after input of changecapture. However, if the before input of changeapply is not the same (different record contents or ordering), or keys are not unique and copy records are not used, this fact is not detected and the rules described above are applied anyway, producing a result that might or might not be useful.

Schemas

The changeapply output data set has the same schema as the change data set, with the change_code field removed.

The before interface schema is:

```
record (key:type; ... value:type; ... beforeRec:*)
```

The change interface schema is:

```
record (change_code:int8; key:type; ... value:type; ... changeRec:*)
```

The after interface schema is:

```
record (afterRec:*)
```

Transfer behavior

The change to after transfer uses an internal transfer adapter to drop the change_code field from the transfer. This transfer is declared first, so the schema of the change data set determines the schema of the after data set.

Key comparison fields

An internal, generic comparison function compares key fields. An internal, generic equality function compares non-key fields. You adjust the comparison with parameters and equality functions for individual fields using the -param suboption of the -key, -allkeys, -allvalues, and -value options.

Changeapply: syntax and options

You must specify at least one -key field or specify the -allkeys option. Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
changeapply
-key input_field_name [-cs | ci] [-asc | -desc] [-nulls first | last] [param params]
[-key inpt_field_name [-cs | ci] [-asc | -desc] -nulls first | last][param params ...]
| -allkeys [-cs | ci] [-asc | -desc] [-nulls first | last][param params]
[-allvalues [-cs | ci] [-param params]]
[-codeField field_name]
[-copyCode n]
[-collation_sequence locale | collation_file_pathname | OFF]
[-deleteCode n ]
[-doStats]
[-dropkey input_field_name ...]
[-dropvalue input_field_name ...]
[-editCode n]
[-ignoreDeleteValues]
[-insertCode n]
[-value inpt_field_name [-ci | -cs] [param params] ...]
```

Note: The -checkSort option has been deprecated. By default, partitioner and sort components are now inserted automatically.

Table 9. Changeapply options

Option	Use
-key	<p>-key <i>input_field_name</i> [-cs ci] [-asc -desc] [-nulls first last] [-param <i>params</i>] [-key <i>input_field_name</i> [-cs ci] [-asc -desc] [-nulls first last] [-param <i>params</i>] ...]</p> <p>Specify one or more key fields.</p> <p>You must specify at least one key for this option or specify the -allkeys option. These options are mutually exclusive. You cannot use a vector, subrecord, or tagged aggregate field as a value key.</p> <p>The -ci suboption specifies that the comparison of value keys is case insensitive. The -cs suboption specifies a case-sensitive comparison, which is the default.</p> <p>-asc and -desc specify ascending or descending sort order.</p> <p>-nulls first last specifies the position of nulls.</p> <p>The -params suboption allows you to specify extra parameters for a key. Specify parameters using pr <i>property</i> = <i>value</i> pairs separated by commas.</p>

Table 9. Changeapply options (continued)

Option	Use
-allkeys	<p>-allkeys [-cs ci] [-asc -desc] [-nulls first last] [-param <i>params</i>]</p> <p>Specify that all fields not explicitly declared are key fields. The suboptions are the same as the suboptions described for the -key option above.</p> <p>You must specify either the -allkeys option or the -key option. They are mutually exclusive.</p>
-allvalues	<p>-allvalues [-cs ci] [-param <i>params</i>]</p> <p>Specify that all fields not otherwise explicitly declared are value fields.</p> <p>The -ci suboption specifies that the comparison of value keys is case insensitive. The -cs suboption specifies a case-sensitive comparison, which is the default.</p> <p>The -param suboption allows you to specify extra parameters for a key. Specify parameters using property=value pairs separated by commas.</p> <p>The -allvalues option is mutually exclusive with the -value and -allkeys options.</p>
-codeField	<p>-codeField <i>field_name</i></p> <p>The name of the change code field. The default is change_code. This should match the field name used in changecapture.</p>
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> • Specify a predefined IBM ICU <i>locale</i>. • Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i>. • Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>
-copyCode	<p>-copyCode <i>n</i></p> <p>Specifies the value for the change_code field in the change record for the copy result. The <i>n</i> value is an int8. The default value is 0.</p> <p>A copy record means that the <i>before</i> record should be copied to the output without modification.</p>

Table 9. Changeapply options (continued)

Option	Use
-deleteCode	<p><code>-deleteCode <i>n</i></code></p> <p>Specifies the value for the change_code field in the change record for the delete result. The <i>n</i> value is an int8. The default value is 2.</p> <p>A delete record means that a record in the <i>before</i> data set must be deleted to produce the <i>after</i> data set.</p>
-doStats	<p><code>-doStats</code></p> <p>Configures the operator to display result information containing the number of input records and the number of copy, delete, edit, and insert records.</p>
-dropkey	<p><code>-dropkey <i>input_field_name</i></code></p> <p>Optionally specify that the field is not a key field. If you specify this option, you must also specify the -allkeys option.</p> <p>There can be any number of occurrences of this option.</p>
-dropvalue	<p><code>-dropvalue <i>input_field_name</i></code></p> <p>Optionally specify that the field is not a value field. If you specify this option, you must also specify the -allvalues option.</p> <p>There can be any number of occurrences of this option.</p>
-editCode	<p><code>-editCode <i>n</i></code></p> <p>Specifies the value for the change_code field in the change record for the edit result. The <i>n</i> value is an int8. The default value is 3.</p> <p>An edit record means that the value fields in the <i>before</i> data set must be edited to produce the <i>after</i> data set.</p>
-ignoreDeleteValues	<p><code>-ignoreDeleteValues</code></p> <p>Do not check value fields on deletes. Normally, changeapply compares the value fields of delete change records to those in the before record to ensure that it is deleting the correct record. The -ignoreDeleteValues option turns off this behavior.</p>
-insertCode	<p><code>-insertCode <i>n</i></code></p> <p>Specifies the value for the change_code field in the output record for the insert result. The <i>n</i> value is an int8. The default value is 1.</p> <p>An insert means that a record must be inserted into the <i>before</i> data set to reproduce the <i>after</i> data set.</p>

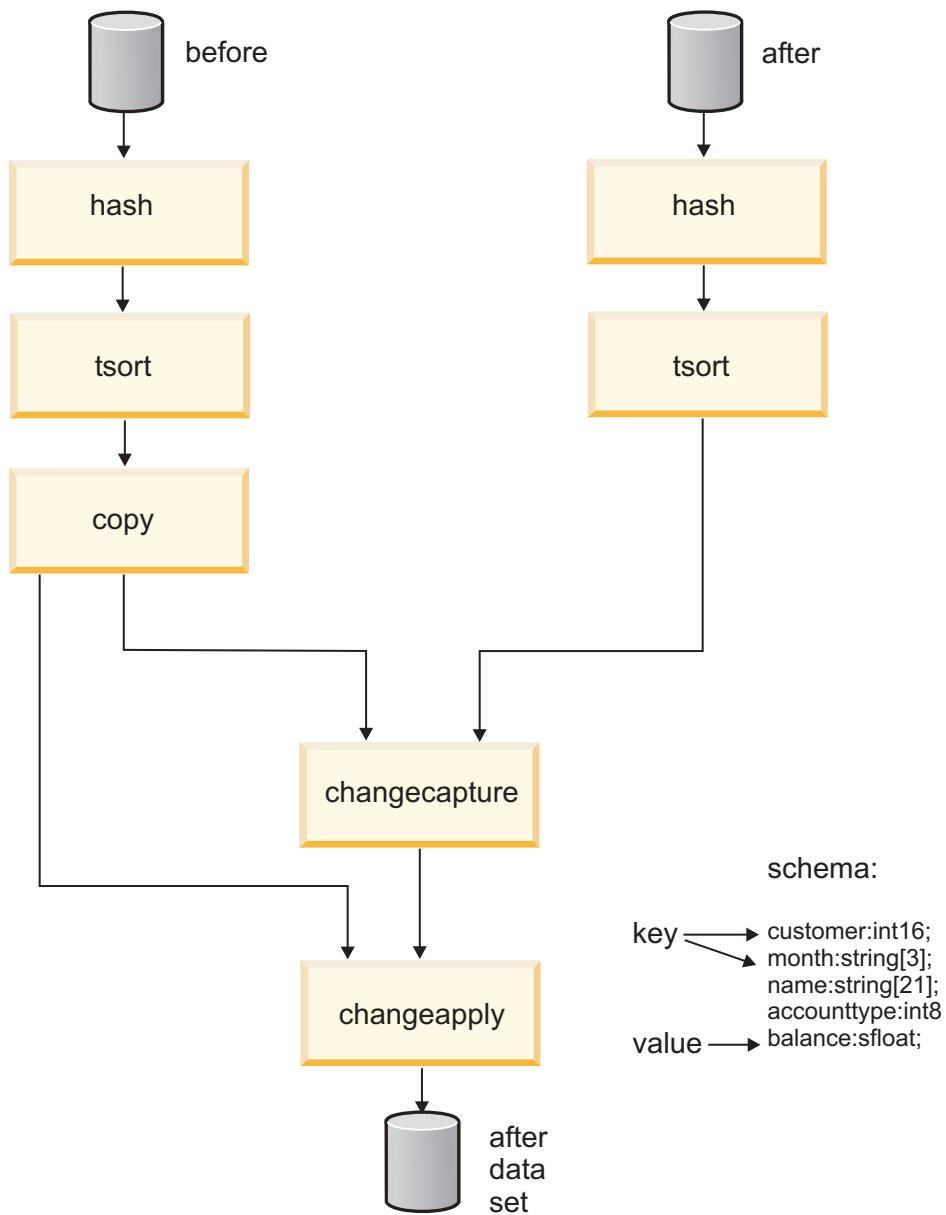
Table 9. Changeapply options (continued)

Option	Use
-value	<p>-value <i>field</i> [-ci] [-cs] [param <i>params</i>]</p> <p>Optionally specifies the name of a value field. The -value option might be repeated if there are multiple value fields.</p> <p>The value fields are modified by edit records, and can be used to ensure that the correct record is deleted when keys are not unique.</p> <p>Note that you cannot use a vector, subrecord, or tagged aggregate field as a value key.</p> <p>The -ci suboption specifies that the comparison of values is case insensitive. The -cs suboption specifies a case-sensitive comparison, which is the default.</p> <p>The -params suboption allows you to specify extra parameters for a key. Specify parameters using <i>property=value</i> pairs separated by commas.</p> <p>The -value and -allvalues options are mutually exclusive.</p>

Example

This example assumes that the input data set records contain customer, month, and balance fields. The operation examines the customer and month fields of each input record for differences. By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the changeapply operator and other operators.

Here is the data flow diagram for the example:



Here is the osh command:

```

$ osh "hash -key month -key customer < beforeRaw.ds |
      tsort -key month -key customer | copy > before_capture.v >
      before_apply.v;
      hash -key month -key customer < afterRaw.ds |
      tsort -key month -key customer > after.v;
      changecapture -key month -key customer -value balance <
      before_capture.v < after.v > change.v;
      changeapply -key month -key customer -value balance <
      before_apply.v < change.v > after.ds"
  
```

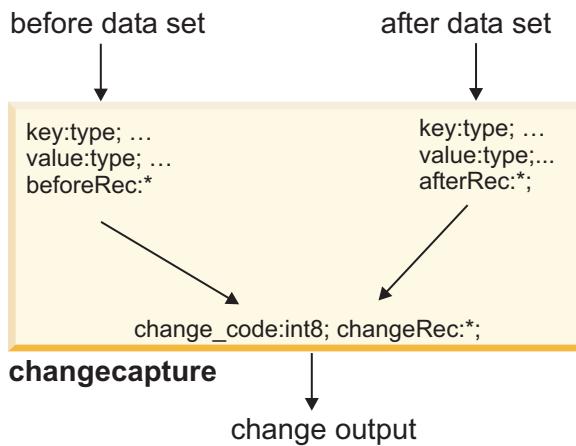
Changecapture operator

The changecapture operator takes two input data sets, denoted before and after, and outputs a single data set whose records represent the changes made to the before data set to obtain the after data set. The operator produces a change data set, whose schema is transferred from the schema of the after data set with the addition of one field: a change code with values encoding the four actions: insert, delete, copy, and edit. The preserve-partitioning flag is set on the change data set.

You can use the companion operator changeapply to combine the changes from the changecapture operator with the original before data set to reproduce the after data set.

The changecapture operator is very similar to the diff operator described in "Diff Operator".

Data flow diagram



Key and value fields

Records from the two input data sets are compared using key and value fields which must be top-level non-vector fields and can be nullable. Using the `-param` suboption of the `-key`, `-allkeys`, `-allvalues`, and `-value` options, you can provide comparison arguments to guide the manner in which key and value fields are compared. In the case of equal key fields, the value fields are compared to distinguish between the copy and edit cases.

Transfer behavior

In the insert and edit cases, the after input is transferred to output. In the delete case, an internal transfer adapter transfers the before keys and values to output. In the copy case, the after input is optionally transferred to output.

Because an internal transfer adapter is used, no user transfer or view adapter can be used with the changecapture operator.

Determining differences

The changecapture output data set has the same schema as the after data set, with the addition of a `change_code` field. The contents of the output depend on whether the after record represents an insert, delete, edit, or copy to the before data set:

- Insert: a record exists in the after data set but not the before data set as indicated by the sorted key fields. The after record is consumed and transferred to the output. No before record is consumed.

If key fields are not unique, changecapture might fail to identify an inserted record with the same key fields as an existing record. Such an insert might be represented as a series of edits, followed by an insert of an existing record. This has consequences for changeapply.

- Delete: a record exists in the before data set but not the after data set as indicated by the sorted key fields. The before record is consumed and the key and value fields are transferred to the output; no after record is consumed.

If key fields are not unique, changecapture might fail to identify a deleted record if another record with the same keys exists. Such a delete might be represented as a series of edits, followed by a delete of a different record. This has consequences for changeapply.

- Edit: a record exists in both the before and after data sets as indicated by the sorted key fields, but the before and after records differ in one or more value fields. The before record is consumed and discarded; the after record is consumed and transferred to the output.

If key fields are not unique, or sort order within a key is not maintained between the before and after data sets, spurious edit records might be generated for those records whose sort order has changed. This has consequences for changeapply

- Copy: a record exists in both the before and after data sets as indicated by the sorted key fields, and furthermore the before and after records are identical in value fields as well. The before record is consumed and discarded; the after record is consumed and optionally transferred to the output. If no after record is transferred, no output is generated for the record; this is the default.

The operator produces a change data set, whose schema is transferred from the schema of the after data set, with the addition of one field: a change code with values encoding insert, delete, copy, and edit. The preserve-partitioning flag is set on the change data set.

Changecapture: syntax and options

```
changecapture
-key input_field_name [-cs | ci] [-asc | -desc][-nulls first | last][-param params]
[-key input_field_name [-cs | ci] [-asc | -desc][-nulls first | last] [-param params ...]
[-allkeys [-cs | ci] [-asc | -desc] [-nulls first | last][-param params ...]
[-allvalues [-cs | ci] [-param params]]
[-codeField field_name]
[-copyCode n]
[-collation_sequence locale | collation_file_pathname | OFF]
[-deleteCode n]
[-doStats]
[-dropkey input_field_name ...]
[-dropvalue input_field_name ...]
[-editCode n]
[-insertCode n]
[-keepCopy | -dropCopy]
[-keepDelete | -dropDelete]
[-keepEdit | -dropEdit]
[-keepInsert | -dropInsert]
[-value input_field_name [-ci | -cs] [-param params] ...]
```

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

You must specify either one or more -key fields or the -allkeys option. You can parameterize each key field's comparison operation and specify the expected sort order (the default is ascending).

Note: The -checkSort option has been deprecated. By default, partitioner and sort components are now inserted automatically.

Table 10. Changecapture options

Option	Use
<p>-key</p>	<p>-key <i>input_field_name</i> [-cs ci] [-asc -desc] [-nulls first last] [-param <i>params</i>] [-key <i>input_field_name</i> [-cs ci] [-asc -desc] [-nulls first last] [-param <i>params</i>] ...]</p> <p>Specify one or more key fields.</p> <p>You must specify either the -allkeys option or at least one key for the -key option. These options are mutually exclusive. You cannot use a vector, subrecord, or tagged aggregate field as a value key.</p> <p>The -ci option specifies that the comparison of value keys is case insensitive. The -cs option specifies a case-sensitive comparison, which is the default.</p> <p>-asc and -desc specify ascending or descending sort order.</p> <p>-nulls first last specifies the position of nulls.</p> <p>The -param suboption allows you to specify extra parameters for a key. Specify parameters using <i>property=value</i> pairs separated by commas.</p>
<p>-allkeys</p>	<p>-allkeys [-cs ci] [-asc -desc] [-nulls first last] [-param <i>params</i>]</p> <p>Specify that all fields not explicitly declared are key fields. The suboptions are the same as the suboptions described for the -key option above.</p> <p>You must specify either the -allkeys option or the -key option. They are mutually exclusive.</p>
<p>-allvalues</p>	<p>-allvalues [-cs ci] [-param <i>params</i>]</p> <p>Specify that all fields not otherwise explicitly declared are value fields.</p> <p>The -ci option specifies that the comparison of value keys is case insensitive. The -cs option specifies a case-sensitive comparison, which is the default.</p> <p>The -param option allows you to specify extra parameters for a key. Specify parameters using <i>property=value</i> pairs separated by commas.</p> <p>The -allvalues option is mutually exclusive with the -value and -allkeys options. You must specify the -allvalues option when you supply the -dropkey option.</p>
<p>-codeField</p>	<p>-codeField <i>field_name</i></p> <p>Optionally specify the name of the change code field. The default is change_code.</p>

Table 10. Changecapture options (continued)

Option	Use
<code>-collation_sequence</code>	<p><code>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</code></p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> Specify a predefined IBM ICU <i>locale</i> Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>
<code>-copyCode</code>	<p><code>-copyCode <i>n</i></code></p> <p> Optionally specify the value of the <i>change_code</i> field in the output record for the copy result. The <i>n</i> value is an int8. The default value is 0. A copy result means that all keys and all values in the <i>before</i> data set are equal to those in the <i>after</i> data set.</p>
<code>-deleteCode</code>	<p><code>-deleteCode <i>n</i></code></p> <p> Optionally specify the value for the <i>change_code</i> field in the output record for the delete result. The <i>n</i> value is an int8. The default value is 2.</p> <p>A delete result means that a record exists in the before data set but not in the after data set as defined by the key fields.</p>
<code>-doStats</code>	<p><code>-doStats</code></p> <p> Optionally configure the operator to display result information containing the number of input records and the number of copy, delete, edit, and insert records.</p>
<code>-dropkey</code>	<p><code>-dropkey <i>input_field_name</i></code></p> <p> Optionally specify that the field is not a key field. If you specify this option, you must also specify the <code>-allkeys</code> option.</p> <p>There can be any number of occurrences of this option.</p>
<code>-dropvalue</code>	<p><code>-dropvalue <i>input_field_name</i></code></p> <p> Optionally specify that the field is not a value field. If you specify this option, you must also specify the <code>-allvalues</code> option.</p> <p>There can be any number of occurrences of this option.</p>

Table 10. Changecapture options (continued)

Option	Use
-editCode	<p>-editCode <i>n</i></p> <p>Optionally specify the value for the change_code field in the output record for the edit result. The <i>n</i> value is an int8. The default value is 3.</p> <p>An edit result means all key fields are equal but one or more value fields are different.</p>
-insertCode	<p>-insertCode <i>n</i></p> <p>Optionally specify the value for the change_code field in the output record for the insert result. The <i>n</i> value is an int8. The default value is 1.</p> <p>An insert result means that a record exists in the after data set but not in the before data set as defined by the key fields.</p>
-keepCopy -dropCopy -keepDelete -dropDelete -keepEdit -dropEdit -keepInsert -dropInsert	<p>-keepCopy -dropCopy -keepDelete -dropDelete -keepEdit -dropEdit -keepInsert -dropInsert</p> <p>Optionally specifies whether to keep or drop copy records at output. By default, the operator creates an output record for all differences except copy.</p>
-value	<p>-value <i>field_name</i> [-ci -cs] [-param <i>params</i>]</p> <p>Optionally specifies one or more value fields.</p> <p>When a before and after record are determined to be copies based on the difference keys (as defined by -key), the value keys can then be used to determine if the after record is an edited version of the before record.</p> <p>Note that you cannot use a vector, subrecord, or tagged aggregate field as a value key.</p> <p>The -ci option specifies that the comparison of values is case insensitive. The -cs option specifies a case-sensitive comparison, which is the default.</p> <p>The -param option allows you to specify extra parameters for a key. Specify parameters using <i>property=value</i> pairs separated by commas.</p> <p>The -value and -allvalues options are mutually exclusive.</p>

Changecapture example 1: all output results

This example assumes that the input data set records contain customer, month, and balance fields. The operation examines the customer and month fields of each input record for differences. By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the changecapture operator and other operators.

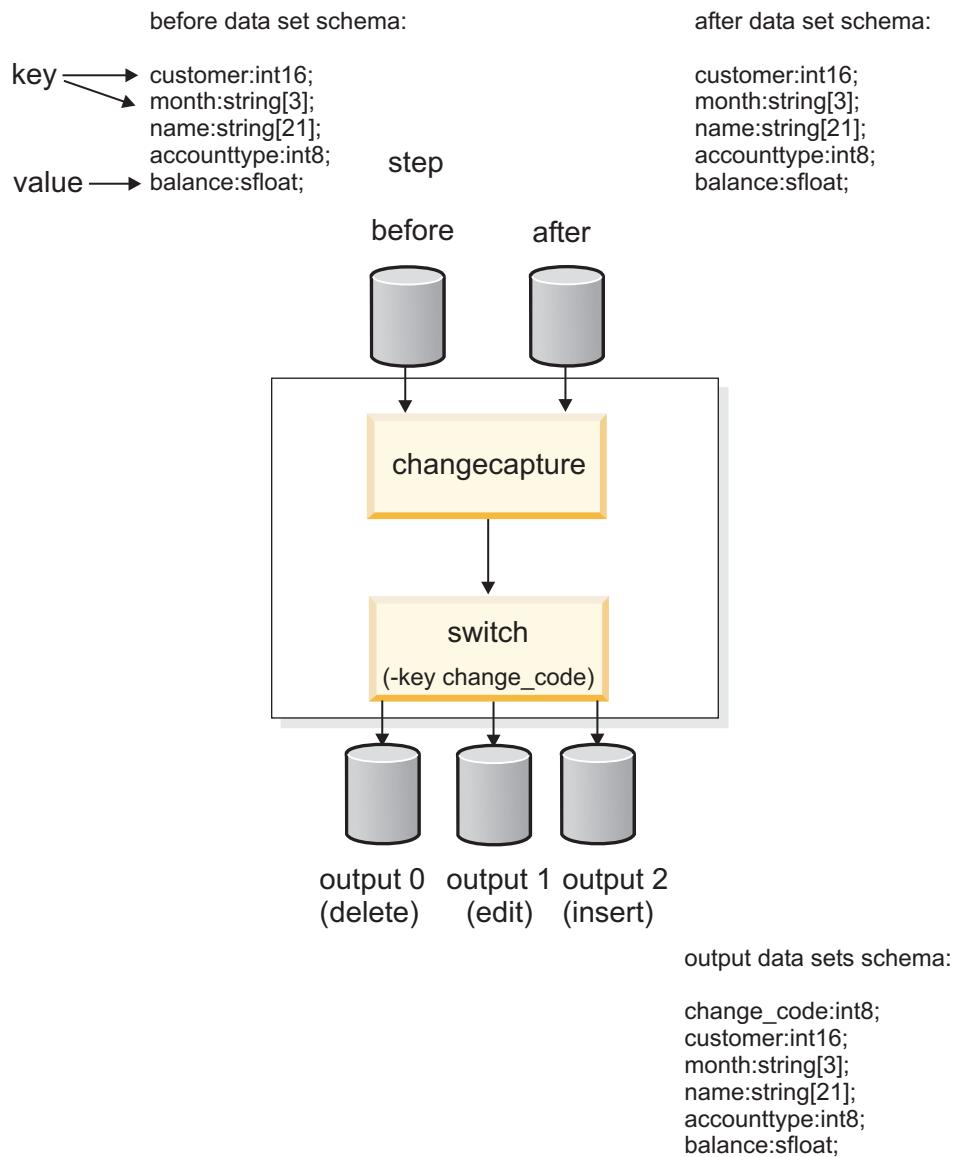
Here is the osh command:

```
$osh "changecapture -key month -key customer
      -value balance
      < before_capture.v < after.v > change.ds"
```

Example 2: dropping output results

In some cases, you might be interested only in some results of the changecapture operator. In this example, you keep only the output records of the edit, delete and insert results. That is, you explicitly drop the copy results so that the output data set contains records only when there is a difference between the before and after data records.

As in Example 1, this example assumes that the before and after data sets are already sorted. Shown below is the data flow diagram for this example:



You specify these key and value fields to the changecapture operator:

```
-key month
-key customer
-value balance
```

After you run the changecapture operator, you invoke the switch operator to divide the output records into data sets based on the result type. The switch operator in this example creates three output data sets: one for delete results, one for edit results, and one for insert results. It creates only three data sets,

because you have explicitly dropped copy results from the changecapture operator by specifying -dropCopy. By creating a separate data set for each of the three remaining result types, you can handle each one differently:

```
-deleteCode 0
-editCode 1
-insertCode 2
```

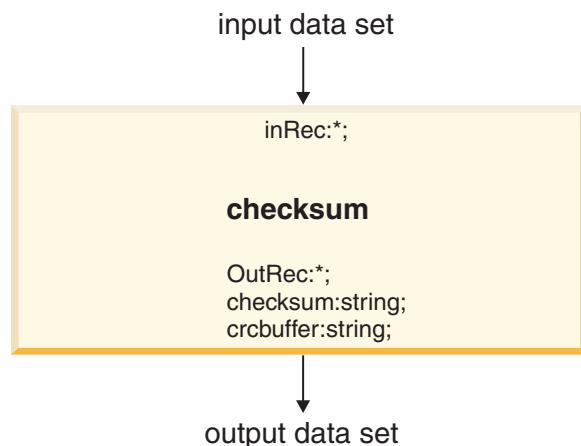
Here is the osh command:

```
$ osh "changecapture -key month -key customer
      -value balance -dropCopy -deleteCode 0 -editCode 1
      -insertCode 2
      < before.ds < after.ds | switch -key changecapture
      > outDelete.ds > outEdit.ds > outInsert.ds"
```

Checksum operator

You can use the checksum operator to add a checksum field to your data records. You can use the same operator later in the flow to validate the data.

Data flow diagram



Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec:*
Output interface schema	outRec:*, string:checksum; string:crcbuffer
	note that the checksum field name can be changed, and the crcbuffer field is optional.
Transfer behavior	inRec -> outRec without record modification
Execution mode	parallel (default) or sequential
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated

Property	Value
Composite operator	no
Combinable operator	yes

The checksum operator:

- Takes any single data set as input
- Has an input interface schema consisting of a single schema variable *inRec* and an output interface schema consisting of a single schema variable *outRec*
- Copies the input data set to the output data set, and adds one or possibly two, fields.

Checksum: syntax and options

```
checksum [-checksum_name field_name]
[-export_name field_name]
[-dropcol field | keepcol field]
```

Table 11. Checksum options

Option	Use
-checksum_name	-checksum_name <i>field_name</i> Specifies a name for the output field containing the checksum value. By default the field is named checksum.
-export_name	-export_name <i>field_name</i> Specifies the name of the output field containing the buffer the checksum algorithm was run with. If this option is not specified, the checksum buffer is not written to the output data set.
-dropcol	-dropcol <i>field</i> Specifies a field that will not be used to generate the checksum. This option can be repeated to specify multiple fields. This option is mutually exclusive with -keepcol.
-keepcol	-keepcol <i>field</i> Specifies a list of fields that will be used to generate the checksum. This option can be repeated to specify multiple fields. This option is mutually exclusive with -dropcol.

Checksum: example

In this example you use checksum to add a checksum field named "check" that is calculated from the fields week_total, month_total, and quarter_total.

The osh command is:

```
$ osh "checksum -checksum_name 'check' -keepcol 'week_total'
-keepcol 'month_total' -keepcol 'quarter_total' < in.ds > out0.ds"
```

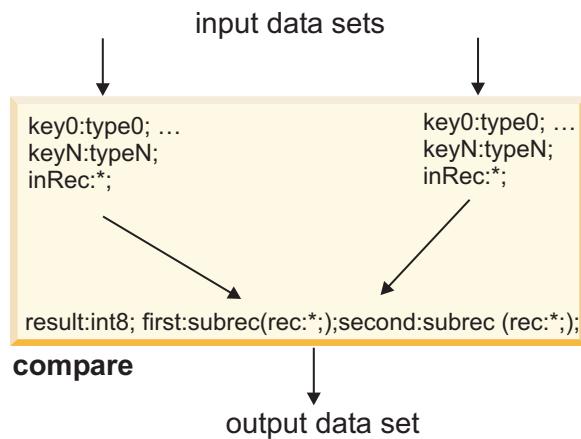
Compare operator

The compare operator performs a field-by-field comparison of records in two presorted input data sets. This operator compares the values of top-level non-vector data types such as strings. All appropriate comparison parameters are supported, for example, case sensitivity and insensitivity for string comparisons.

The compare operator does not change the schema, partitioning, or content of the records in either input data set. It transfers both data sets intact to a single output data set generated by the operator. The comparison results are also recorded in the output data set.

By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the changecapture operator and other operators.

Data flow diagram



Note: If you do not specify key fields, the operator treats all fields as key fields.

compare: properties

Table 12. Compare properties

Property	Value
Number of input data sets	2
Number of output data sets	1
Input interface schema	key0:type0; ... keyN:typeN; inRec:__*;
Output interface schema	result:int8; first:subrec(rec:__); second:subrec (rec:__);;
Transfer behavior	The first input data set is transferred to first.rec. The second input data set is transferred to second.rec
Execution mode	parallel (default) or sequential
Input partitioning style	keys in same partition
Partitioning method	same (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	no

The compare operator:

- Compares only scalar data types. See "Restrictions".
- Takes two presorted data sets as input and outputs one data set.
- Has an input interface schema consisting of the key fields and the schema variable inRec, and an output interface schema consisting of the result field of the comparison and a subrecord field containing each input record.
 - Performs a field-by-field comparison of the records of the input data sets.
 - Transfers the two input data sets to the single output data set without altering the input schemas, partitioning, or values.
 - Writes to the output data set signed integers that indicate comparison results.

Restrictions

The compare operator:

- Compares only scalar data types, specifically string, integer, float, decimal, raw, date, time, and timestamp; you cannot use the operator to compare data types such as tagged aggregate, subrec, vector, and so on.
- Compares only fields explicitly specified as key fields, except when you do not explicitly specify any key field. In that case, the operator compares all fields that occur in both records.

Results field

The operator writes the following default comparison results to the output data set. In each case, you can specify an alternate value:

Description of Comparison Results	Default Value
The record in the first input data set is greater than the corresponding record in the second input data set.	1
The record in the first input data set is equal to the corresponding record in the second input data set.	0
The record in the first input data set is less than the corresponding record in the second input data set.	-1
The number of records in the first input data is greater than the number of records in the second input data set.	2
The number of records in the first input data set is less than the number of records in the second input data set.	-2

When this operator encounters any of the mismatches described in the table shown above, you can force it to take one or both of the following actions:

- Terminate the remainder of the current comparison
- Output a warning message to the screen

Compare: syntax and options

```
compare
[-abortOnDifference]
[-field fieldname [-ci | -cs] [-param params] ...]
[[-key fieldname [-ci | -cs] [-param params] ...]
[-collation_sequence locale | collation_file_pathname | OFF]
[-first n]
[-gt n | -eq n | -lt n]
[-second n]
[-warnRecordCountMismatch]
```

None of the options are required.

Table 13. Compare options

Option	Use
-abortOnDifference	<p>-abortOnDifference</p> <p>Forces the operator to abort its operation each time a difference is encountered between two corresponding fields in any record of the two input data sets.</p> <p>This option is mutually exclusive with -warnRecordCountMismatch, -lt, -gt, -first, and -second.</p>
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none">• Specify a predefined IBM ICU <i>locale</i>• Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i>• Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>
-field or -key	<p>-field <i>fieldname</i> [-ci -cs] [-param <i>params</i>] -key <i>fieldname</i> [-ci -cs] [-param <i>params</i>]</p> <p>-field or -key is a key field to compare in the two input data sets. The maximum number of fields is the number of fields in the input data sets. If no key fields are explicitly specified, all fields shared by the two records being processed are compared.</p> <p><i>fieldname</i> specifies the name of the field.</p> <p>-ci specifies that the comparison of strings is case-insensitive.</p> <p>-cs specifies case-sensitive string comparison, which is the default.</p> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property=value</i> pairs separated by commas.</p>
-first	<p>-first <i>n</i></p> <p>Configures the operator to write <i>n</i> (a signed integer between -128 and 127) to the output data set if the number of records in the second input data set exceeds the number of records in the first input data set. The default value is -2.</p>

Table 13. Compare options (continued)

Option	Use
-gt -eq -lt	<p>-gt n -eq n -lt n</p> <p>Configures the operator to write n (a signed integer between -128 and 127) to the output data set if the record in the first input data set is:</p> <ul style="list-style-type: none"> Greater than (-gt) the equivalent record in the second input data set. The default is 1. Equal to (-eq) the equivalent record in the second input data set. The default is 0. Less than (-lt) the equivalent record in the second input data set. The default is -1.
-second	<p>-second n</p> <p>Configures the operator to write n (an integer between -128 and 127) to the output data set if the number of records in the first input data set exceeds the number of records in the second input data set. The default value is 2.</p>
-warnRecordCountMismatch	<p>-warnRecordCountMismatch</p> <p>Forces the operator to output a warning message when a comparison is aborted due to a mismatch in the number of records in the two input data sets.</p>

Compare example 1: running the compare operator in parallel

Each record has the fields name, age, and gender. All operations are performed on the key fields, age and gender. By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the compare operator and other operators.

The compare operator runs in parallel mode which is the default mode for this operator; and the -abortOnDifference option is selected to force the operator to abort at the first indication of mismatched records.

Here is the osh code corresponding to these operations:

```
$ osh "compare -abortOnDifference -field age -field gender < sortedDS0.v < sortedDS1.v > outDS.ds"
```

The output record format for a successful comparison of records looks like this, assuming all default values are used:

```
result:0 first:name; second:age; third:gender;
```

Example 2: running the compare operator sequentially

By default, the compare operator executes in parallel on all processing nodes defined in the default node pool.

However, you might want to run the operator sequentially on a single node. This could be useful when you intend to store a persistent data set to disk in a single partition. For example, your parallel job might perform data cleansing and data reduction on its input to produce an output data set that is much smaller than the input. Before storing the results to disk, or passing the result to a sequential job, you can use a sequential compare operator to store the data set to disk with a single partition.

To force the operator to execute sequentially specify the [-seq] framework argument. When executed sequentially, the operator uses a collection method of any. A sequential operator using this collection method can have its collection method overridden by an input data set to the operator.

Suppose you want to run the same job as shown in "Example 1: Running the compare Operator in Parallel" but you want the compare operator to run sequentially.

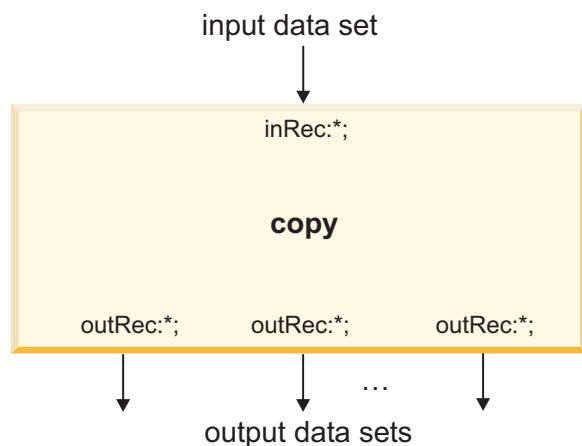
Issue this osh command:

```
$ osh "compare -field gender -field age [-seq] < inDS0.ds
      < inDS1.ds > outDS.ds"
```

Copy operator

You can use the modify operator with the copy operator to modify the data set as the operator performs the copy operation. See "Modify Operator" for more information on modifying data.

Data flow diagram



Copy: properties

Table 14. Copy properties

Property	Value
Number of input data sets	1
Number of output data sets	0 or more (0 - n) set by user
Input interface schema	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec -> outRec without record modification
Execution mode	parallel (default) or sequential
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	no
Combinable operator	yes

The copy operator:

- Takes any single data set as input
- Has an input interface schema consisting of a single schema variable *inRec* and an output interface schema consisting of a single schema variable *outRec*
- Copies the input data set to the output data sets without affecting the record schema or contents

Copy: syntax and options

`copy [-checkpoint n] [-force]`

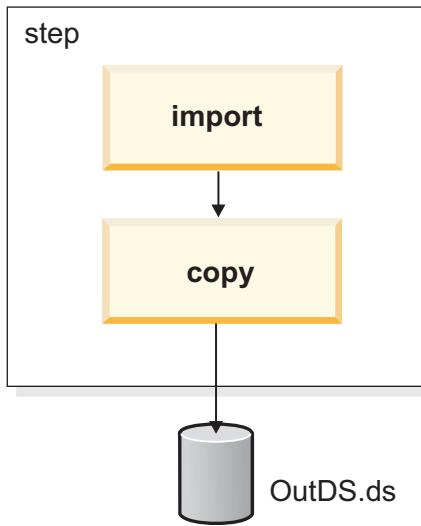
Table 15. Copy options

Option	Use
-checkpoint	<p>-checkpoint <i>n</i></p> <p>Specifies the number of records copied from the input persistent data set to each segment of each partition of the output data set. The value of <i>n</i> must be positive. Its default value is 1.</p> <p>In order for this option to be specified, the input data set to the copy operator must be persistent and the operator must be run in parallel.</p> <p>The step containing the copy operator must be checkpointed, that is, you must have specified the keyword -restartable as part of the step definition.</p>
-force	<p>-force</p> <p>Specifies that WebSphere DataStage cannot attempt to optimize the step by removing the copy operator.</p> <p>In some cases, WebSphere DataStage can remove a copy operator if it determines that the copy operator is unnecessary. However, your job might require the copy operator to execute. In this case, you use the -force option. See "Preventing WebSphere DataStage from Removing a copy Operator".</p>

Preventing WebSphere DataStage from removing a copy operator

Before running a job, WebSphere DataStage optimizes each step. As part of this optimization, WebSphere DataStage removes unnecessary copy operators. However, this optimization can sometimes remove a copy operator that you do not want removed.

For example, the following data flow imports a single file into a virtual data set, then copies the resulting data set to a new data set:



Here is the osh command:

```
$ osh "import -file inFile.dat -schema recordSchema | copy > outDS.ds"
```

This occurs:

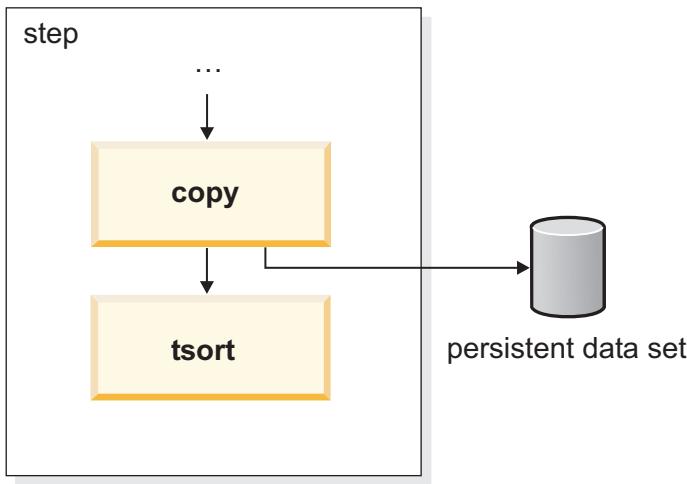
1. The import operator reads the data file, inFile.dat, into a virtual data set. The virtual data set is written to a single partition because it reads a single data file. In addition, the import operator executes only on the processing node containing the file.
2. The copy operator runs on all processing nodes in the default node pool, because no constraints have been applied to the input operator. Thus, it writes one partition of outDS.ds to each processing node in the default node pool.

However, if WebSphere DataStage removes the copy operator as part of optimization, the resultant persistent data set, outDS.ds, would be stored only on the processing node executing the import operator. In this example, outDS.ds would be stored as a single partition data set on one node.

To prevent removal specify the -force option. The operator explicitly performs the repartitioning operation to spread the data over the system.

Copy example 1: The copy operator

In this example, you sort the records of a data set. However, before you perform the sort, you use the copy operator to create two copies of the data set: a persistent copy, which is saved to disk, and a virtual data set, which is passed to the sort operator. Here is a data flow diagram of the operation:



Output data set 0 from the copy operator is written to outDS1.ds and output data set 1 is written to the tsort operator.

The syntax is as follows:

```
$ osh "... | copy > outDS1.ds | tsort options ..."
```

Example 2: running the copy operator sequentially

By default, the copy operator executes in parallel on all processing nodes defined in the default node pool. However, you might have a job in which you want to run the operator sequentially, that is, on a single node. For example, you might want to store a persistent data set to disk in a single partition.

You can run the operator sequentially by specifying the [seq] framework argument to the copy operator.

When run sequentially, the operator uses a collection method of any. However, you can override the collection method of a sequential operator. This can be useful when you want to store a sorted data set to a single partition. Shown below is a osh command data flow example using the ordered collection operator with a sequential copy operator.

```
$ osh "... opt1 | ordered | copy [seq] > outDS.ds"
```

Diff operator

Note: The diff operator has been superseded by the changecapture operator. While the diff operator has been retained for backwards compatibility, you might use the changecapture operator for new development.

The diff operator performs a record-by-record comparison of two versions of the same data set (the before and after data sets) and outputs a data set whose records represent the difference between them. The operator assumes that the input data sets are hash-partitioned and sorted in ascending order on the key fields you specify for the comparison.

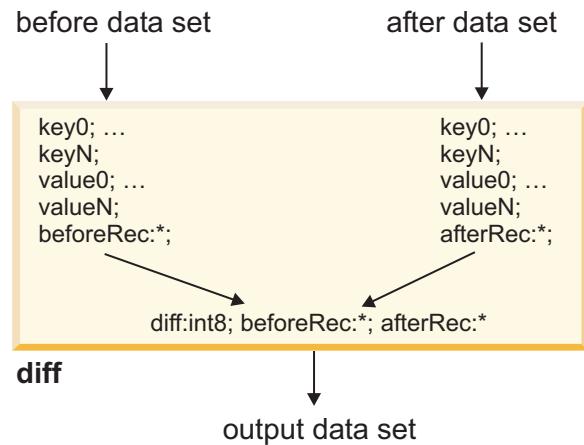
The comparison is performed based on a set of difference key fields. Two records are copies of one another if they have the same value for all difference keys. In addition, you can specify a set of value key fields. If two records are copies based on the difference key fields, the value key fields determine if one record is a copy or an edited version of the other.

The diff operator is very similar to the changecapture operator described in "Changecapture Operator". In most cases, you should use the changecapture operator rather than the diff operator.

By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the diff operator and other operators. The diff operator does not behave like Unix diff.

Data flow diagram

The input data sets are known as the before and after data sets.



diff: properties

Table 16. Diff properties

Property	Value
Number of input data sets	2
Number of output data sets	1
Input interface schema before data set: after data sets:	key0; ... keyn; value0; ... valuen; beforeRec: *; key0; ... keyn; value0; ... valuen; afterRec: *;
Output interface schema	diff:int8; beforeRec: *; afterRec: *;
Transfer behavior before to output: after to output:	beforeRec -> beforeRec without record modification afterRec -> afterRec without record modification
Execution mode	parallel (default) or sequential
Input partitioning style	keys in same partition
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	no

Transfer behavior

The operator produces a single output data set, whose schema is the catenation of the before and after input schemas. Each record of the output data set has the following format:

diff:int8	Fields from before record	Fields from after record that are not in before record
-----------	---------------------------	---

The usual name conflict resolution rules apply.

The output data set contains a number of records in the range:

`num_in_before <= num_in_output <= (num_in_before + num_in_after)`

The number of records in the output data set depends on how many records are copies, edits, and deletes. If the before and after data sets are exactly the same, the number of records in the output data set equals the number of records in the before data set. If the before and after data sets are completely different, the output data set contains one record for each before and one record for each after data set record.

Key fields

The before data set's schema determines the difference key type. You can use an upstream modify operator to alter it. The after data set's key field(s) must have the same name as the before key field(s) and be either of the same data type or of a compatible data type.

The same rule holds true for the value fields: The after data set's value field(s) must be of the same name and data type as the before value field(s). You can use an upstream modify operator to bring this about.

Only top-level, non-vector, non-nullable fields might be used as difference keys. Only top-level, non-vector fields might be used as value fields. Value fields might be nullable.

Identical field names

When the two input data sets have the same field name, the diff operator retains the field of the first input, drops the identically named field from the second output, and issues a warning for each dropped field. Override the default behavior by modifying the second field name so that both versions of the field are retained in the output. (See "Modify Operator" .) You can then write a custom operator to select the version you require for a given job.

Determining differences

The diff operator reads the current record from the before data set, reads the current record from the after data set, and compares the records of the input data sets using the difference keys. The comparison results are classified as follows:

- Insert: A record exists in the after data set but not the before data set.
 - The operator transfers the after record to the output.
 - The operator does not copy the current before record to the output but retains it for the next iteration of the operator.
 - The data type's default value is written to each before field in the output.
 - By default the operator writes a 0 to the diff field of the output record.
- Delete: A record exists in the before data set but not the after data set.
 - The operator transfers the before record to the output
 - The operator does not copy the current after record to the output but retains it for the next iteration of the operator.
 - The data type's default value is written to each after field in the output.
 - By default, the operator writes a 1 to the diff field of the output record.

- Copy: The record exists in both the before and after data sets and the specified value field values have not been changed.
 - The before and after records are both transferred to the output.
 - By default, the operator writes a 2 to the diff (first) field of the output record.
- Edit: The record exists in both the before and after data sets; however, one or more of the specified value field values have been changed.
 - The before and after records are both transferred to the output.
 - By default, the operator writes a 3 to the diff (first) field of the output record.

Options are provided to drop each kind of output record and to change the numerical value written to the diff (first) field of the output record.

In addition to the difference key fields, you can optionally define one or more value key fields. If two records are determined to be copies because they have equal values for all the difference key fields, the operator then examines the value key fields.

- Records whose difference and value key fields are equal are considered copies of one another. By default, the operator writes a 2 to the diff (first) field of the output record.
- Records whose difference key fields are equal but whose value key fields are not equal are considered edited copies of one another. By default, the operator writes a 3 to the diff (first) field of the output record.

Diff: syntax and options

You must specify at least one difference key to the operator using -key.

```
diff
-key field [-ci | -cs] [-param params] [-key field [-ci | -cs] [-param params]...]
[-allValues [-ci | -cs] [-param params]]
[-collation_sequence locale | collation_file_pathname | OFF]
[-copyCode n]
[-deleteCode n]
[-dropCopy]
[-dropDelete]
[-dropEdit]
[-dropInsert]
[-editCode n]
[-insertCode n]
[-stats]
[-tolerateUnsorted]
[-value field [-ci | -cs] [-param params] ...]
```

Table 17. Diff options

Option	Use
-key	<p>-key field [-ci -cs] [-param params]</p> <p>Specifies the name of a difference key field. The -key option might be repeated if there are multiple key fields.</p> <p>Note that you cannot use a nullable, vector, subrecord, or tagged aggregate field as a difference key.</p> <p>The -ci option specifies that the comparison of difference key values is case insensitive. The -csoption specifies a case-sensitive comparison, which is the default.</p> <p>The -params suboption allows you to specify extra parameters for a key. Specify parameters using <i>property=value</i> pairs separated by commas.</p>

Table 17. Diff options (continued)

Option	Use
-allValues	<p><code>-allValues [-ci -cs] [-param params]</code> Specifies that all fields other than the difference key fields identified by <code>-key</code> are used as value key fields. The operator does not use vector, subrecord, and tagged aggregate fields as value keys and skips fields of these data types.</p> <p>When a before and after record are determined to be copies based on the difference keys, the value keys can then be used to determine if the after record is an edited version of the before record.</p> <p>The <code>-ci</code> option specifies that the comparison of value keys is case insensitive. The <code>-cs</code> option specifies a case-sensitive comparison, which is the default.</p> <p>The <code>-params</code> suboption allows you to specify extra parameters for a key. Specify parameters using <code>property=value</code> pairs separated by commas.</p>
-collation_sequence	<p><code>-collation_sequence locale collation_file_pathname OFF</code></p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> Specify a predefined IBM ICU locale Write your own collation sequence using ICU syntax, and supply its <code>collation_file_pathname</code> Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>
-copyCode	<p><code>-copyCode n</code></p> <p>Specifies the value for the diff field in the output record when the before and after records are copies. The <code>n</code> value is an int8. The default value is 2.</p> <p>A copy means all key fields and all optional value fields are equal.</p>
-deleteCode	<p><code>-deleteCode n</code></p> <p>Specifies the value for the diff field in the output record for the delete result. The <code>n</code> value is an int8. The default value is 1.</p> <p>A delete result means that a record exists in the <code>before</code> data set but not in the after data set as defined by the difference key fields.</p>

Table 17. Diff options (continued)

Option	Use
<code>-dropCopy -dropDelete -dropEdit -dropInsert</code>	<p><code>-dropCopy -dropDelete -dropEdit -dropInsert</code></p> <p>Specifies to drop the output record, meaning not generate it, for any one of the four difference result types. By default, an output record is always created by the operator.</p> <p>You can specify any combination of these four options.</p>
<code>-editCode</code>	<p><code>-editCode n</code></p> <p>Specifies the value for the diff field in the output record for the edit result. The <i>n</i> value is an int8. The default value is 3.</p> <p>An edit result means all difference key fields are equal but one or more value key fields are different.</p>
<code>-insertCode</code>	<p><code>-insertCode n</code></p> <p>Specifies the value for the diff field in the output record for the insert result. The <i>n</i> value is an int8. The default value is 0.</p> <p>An insert result means that a record exists in the <i>after</i> data set but not in the before data set as defined by the difference key fields.</p>
<code>-stats</code>	<p><code>-stats</code></p> <p>Configures the operator to display result information containing the number of input records and the number of copy, delete, edit, and insert records.</p>
<code>-tolerateUnsorted</code>	<p><code>-tolerateUnsorted</code></p> <p>Specifies that the input data sets are not sorted. By default, the operator generates an error and aborts the step when detecting unsorted inputs.</p> <p>This option allows you to process groups of records that might be arranged by the difference key fields but not sorted. The operator consumes input records in the order in which they appear on its input.</p> <p>If you use this option, no automatic partitioner or sort insertions are made.</p>

Table 17. Diff options (continued)

Option	Use
-value	<p>-value <i>field</i> [-ci -cs]</p> <p> Optionally specifies the name of a value key field. The -value option might be repeated if there are multiple value fields.</p> <p> When a before and after record are determined to be copies based on the difference keys (as defined by -key), the value keys can then be used to determine if the after record is an edited version of the before record.</p> <p> Note that you cannot use a vector, subrecord, or tagged aggregate field as a value key.</p> <p> The -ci option specifies that the comparison of value keys is case insensitive. The -cs option specifies a case-sensitive comparison, which is the default.</p> <p> The -params suboption allows you to specify extra parameters for a key. Specify parameters using <i>property=value</i> pairs separated by commas.</p>

Diff example 1: general example

The following example assumes that the input data set records contain a customer and month field. The operator examines the customer and month fields of each input record for differences. By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the diff operator and other operators.

Here is the osh command:

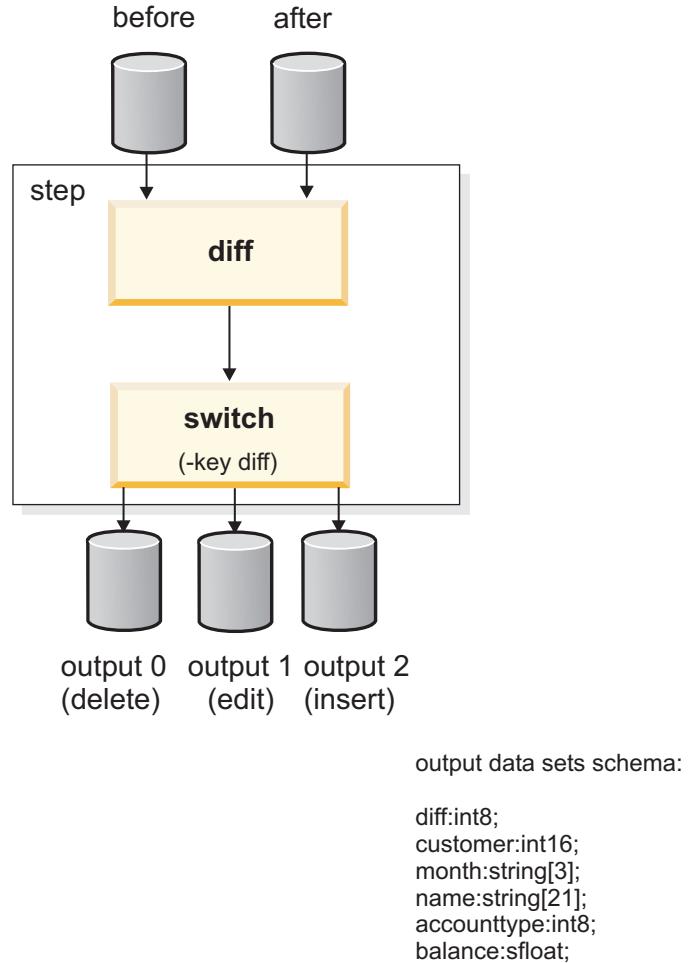
```
$ osh " diff -key month -key customer < before.v < after.v > outDS.ds"
```

Example 2: Dropping Output Results

In some cases, you might be interested only in some results of the diff operator. In this example, you keep only the output records of the edit, delete and insert results. That is, you explicitly drop the copy results so that the output data set contains records only when there is a difference between the before and after data records.

before data set schema:
difference → customer:int16;
key → month:string[3];
name → name:string[21];
accounttype → int8;
value key → balance:sfloat;

after data set schema:
customer:int16;
month:string[3];
name:string[21];
accounttype:int8;
balance:sfloat;



Here is the data flow for this example:

You specify these key and value fields to the diff operator:

```
key=month
key=customer
value=balance
```

After you run the diff operator, you invoke the switch operator to divide the output records into data sets based on the result type. The switch operator in this example creates three output data sets: one for delete results, one for edit results, and one for insert results. It creates only three data sets, because you have explicitly dropped copy results from the diff operation by specifying -dropCopy. By creating a separate data set for each of the three remaining result types, you can handle each one differently:

```
deleteCode=0
editCode=1
insertCode=2
```

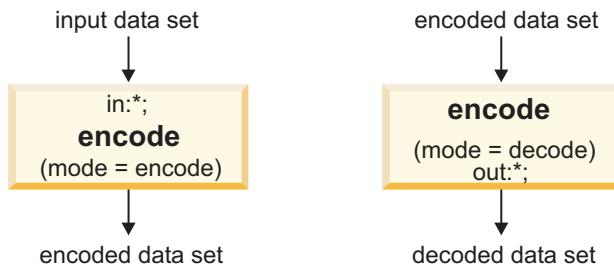
Here is the osh command:

```
$ osh "diff -key month -key customer -value balance
-dropCopy -deleteCode 0 -editCode 1 -insertCode 2
< before.ds < after.ds | switch -key diff
> outDelete.ds > outEdit.ds > outInsert.ds"
```

Encode operator

The encode operator encodes or decodes a WebSphere DataStage data set using a UNIX encoding command that you supply. The operator can convert a WebSphere DataStage data set from a sequence of records into a stream of raw binary data. The operator can also reconvert the data stream to a WebSphere DataStage data set.

Data flow diagram



In the figure shown above, the mode argument specifies whether the operator is performing an encoding or decoding operation. Possible values for mode are:

- encode: encode the input data set
- decode: decode the input data set

encode: properties

Table 18. encode properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	mode = encode: in:*; mode = decode: none
Output interface schema	mode = encode: none mode = decode: out:*
Transfer behavior	in -> out without record modification for an encode/decode cycle
Execution mode	parallel (default) or sequential
Partitioning method	mode = encode: any mode = decode: same
Collection method	any
Preserve-partitioning flag in output data set	mode = encode: sets mode = decode: propagates
Composite operator	no
Combinable operator	no

Encode: syntax and options

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```

encode
  -command command_line
  [-direction encode | decode]
  [-mode [encode | decode]]
  [-encode | -decode]

```

Table 19. Encode options

Option	Use
-command	-command <i>command_line</i> Specifies the command line used for encoding/decoding. The command line must configure the UNIX command to accept input from stdin and write its results to stdout. The command must be located in the search path of your job and be accessible by every processing node on which the encode operator executes.
-direction or -mode	-direction encode decode -mode encode decode Specifies the mode of the operator. If you do not select a direction, it defaults to encode.
-encode	Specify encoding of the data set. Encoding is the default mode.
-decode	Specify decoding of the data set.

Encoding WebSphere DataStage data sets

Each record of a data set has defined boundaries that mark its beginning and end. The encode operator lets you invoke a UNIX command that encodes a WebSphere DataStage data set, which is in record format, into raw binary data and vice versa.

Processing encoded data sets

An encoded data set is similar to a WebSphere DataStage data set. An encoded, persistent data set is stored on disk in the same way as a normal data set, by two or more files:

- A single descriptor file
- One or more data files

However, an encoded data set cannot be accessed like a standard WebSphere DataStage data set, because its records are in an encoded binary format. Nonetheless, you can specify an encoded data set to any operator that does no field-based processing or reordering of the records. For example, you can invoke the copy operator to create a copy of the encoded data set.

You can further encode a data set using a different encoding operator to create an encoded-compressed data set. For example, you might compress the encoded file using WebSphere DataStage's pcompress operator (see "Pcompress Operator"), then invoke the unencode command to convert a binary file to an emailable format. You would then restore the data set by first decompressing and then decoding the data set.

Encoded data sets and partitioning

When you encode a data set, you remove its normal record boundaries. The encoded data set cannot be repartitioned, because partitioning in WebSphere DataStage is performed record-by-record. For that reason, the encode operator sets the preserve-partitioning flag in the output data set. This prevents an WebSphere DataStage operator that uses a partitioning method of any from repartitioning the data set and causes WebSphere DataStage to issue a warning if any operator attempts to repartition the data set.

For a decoding operation, the operator takes as input a previously encoded data set. The preserve-partitioning flag is propagated from the input to the output data set.

Example

In the following example, the encode operator compresses a data set using the UNIX gzip utility. By default, gzip takes its input from stdin. You specify the -c switch to configure the operator to write its results to stdout as required by the operator:

Here is the osh code for this example:

```
$ osh " ... op1 | encode -command 'gzip' > encodedDS.ds"
```

The following example decodes the previously compressed data set so that it might be used by other WebSphere DataStage operators. To do so, you use an instance of the encode operator with a mode of decode. In a converse operation to the encoding, you specify the same operator, gzip, with the -cd option to decode its input.

Here is the osh command for this example:

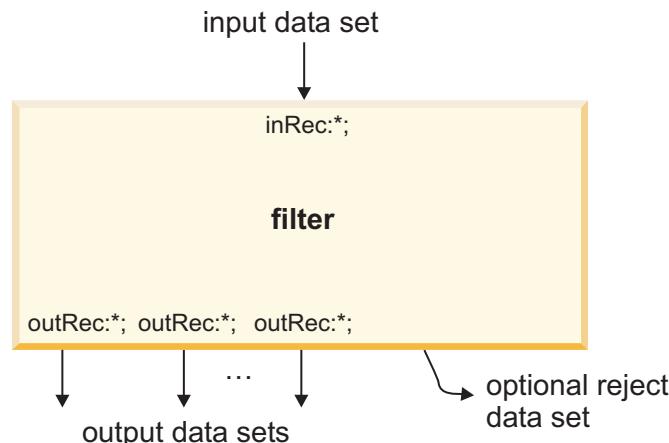
```
$ osh "encode -decode -command 'gzip -d' < inDS.ds | op2 ..."
```

In this example, the command line uses the -d switch to specify the decompress mode of gzip.

Filter operator

The filter operator transfers the input records in which one or more fields meet the requirements you specify. If you request a reject data set, the filter operator transfers records that do not meet the requirements to the reject data set.

Data flow diagram



filter: properties

Table 20. filter properties

Property	Value
Number of input data sets	1
Number of output data sets	1 or more, and, optionally, a reject data set
Input interface schema	inRec:*

Table 20. *filter properties (continued)*

Property	Value
Output interface schema	outRec:*
Transfer behavior	inRec -> outRec without record modification
Execution mode	parallel by default, or sequential
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	no
Combinable operator	yes

Filter: syntax and options

The `-where` option is required. Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
filter
-where 'P'[-target dsNum] [-where 'P' [-target dsNumm] ... ]
[-collation_sequence locale | collation_file_pathname | OFF]
[-first]
[-nulls first | last]
[-reject]
```

Table 21. *Filter options*

Option	Use
<code>-where</code>	<p><code>-where P [-target dsNum]</code></p> <p>Specifies the predicate which determines the filter. In SQL, a predicate is an expression which evaluates as TRUE, FALSE, or UNKNOWN and whose value depends on the value of one or more field values.</p> <p>Enclose the predicate in single quotes. Single quotes within the predicate must be preceded by the backslash character (\), as in "Example 3: Evaluating Input Records" below. If a field is formatted as a special WebSphere DataStage data type, such as date or timestamp, enclose it in single quotes.</p> <p>Multi-byte Unicode character data is supported in predicate field names, constants, and literals.</p> <p>Multiple <code>-where</code> options are allowed. Each occurrence of <code>-where</code> causes the output data set to be incremented by one, unless you use the <code>-target</code> suboption.</p>
<code>-first</code>	<p><code>-first</code></p> <p>Records are output only to the data set corresponding to the first <code>-where</code> clause they match. The default is to write a record to the data sets corresponding to all <code>-where</code> clauses they match.</p>

Table 21. Filter options (continued)

Option	Use												
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> Specify a predefined IBM ICU <i>locale</i> Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>												
-nulls	<p>-nulls first last</p> <p>By default, nulls are evaluated first, before other values. To override this default, specify -nulls last.</p>												
-reject	<p>-reject</p> <p>By default, records that do not meet specified requirements are dropped. Specify this option to override the default. If you do, attach a reject output data set to the operator.</p>												
-target	<p>-target <i>dsNum</i></p> <p>An optional sub-property of where. Use it to specify the target data set for a where clause. Multiple -where clauses can direct records to the same output data set. If a target data set is not specified for a particular -where clause, the output data set for that clause is implied by the order of all -where properties that do not have the -target sub-property. For example:</p> <table> <thead> <tr> <th>Property</th> <th>Data set</th> </tr> </thead> <tbody> <tr> <td>-where "field1 < 4"</td> <td>0</td> </tr> <tr> <td>-where "field2 like 'bb'"</td> <td>1</td> </tr> <tr> <td>-where "field3 like 'aa'" -target</td> <td>2</td> </tr> <tr> <td>-where "field4 > 10" -target</td> <td>0</td> </tr> <tr> <td>-where "field5 like 'c.*'"</td> <td>2</td> </tr> </tbody> </table>	Property	Data set	-where "field1 < 4"	0	-where "field2 like 'bb'"	1	-where "field3 like 'aa'" -target	2	-where "field4 > 10" -target	0	-where "field5 like 'c.*'"	2
Property	Data set												
-where "field1 < 4"	0												
-where "field2 like 'bb'"	1												
-where "field3 like 'aa'" -target	2												
-where "field4 > 10" -target	0												
-where "field5 like 'c.*'"	2												

Job monitoring information

The filter operator reports business logic information which can be used to make decisions about how to process data. It also reports summary statistics based on the business logic.

The business logic is included in the metadata messages generated by WebSphere DataStage as custom information. It is identified with:

```
name="BusinessLogic"
```

The output summary per criterion is included in the summary messages generated by WebSphere DataStage as custom information. It is identified with:

```
name="CriterionSummary"
```

The XML tags, criterion, case and where, are used by the filter operator when generating business logic and criterion summary custom information. These tags are used in the example information below.

Example metadata and summary messages

```
<response type="metadata">
  <component ident="filter">
    <componentstats startTime="2002-08-08 14:41:56"/>
    <linkstats portNum="0" portType="in"/>
    <linkstats portNum="0" portType="out"/>
    <linkstats portNum="1" portType="out"/>
    <custom_info Name="BusinessLogic"
      Desc="User-supplied logic to filter operator">
      <criterion name="where">
        <where value="true" output_port="0"/>
        <where value="false" output_port="1"/>
      </criterion>
    </custom_info>
  </component>
</response>

<response type="summary">
  <component ident="filter" pid="2239">
    <componentstats startTime=
      "2002-08-08 14:41:59" stopTime="2002-08-08 14:42:40" percentCPU="99.5"/>
    <linkstats portNum="0" portType="in" recProcessed="1000000"/>
    <linkstats portNum="0" portType="out" recProcessed="500000"/>
    <linkstats portNum="1" portType="out" recProcessed="500000"/>
    <custom_info Name="CriterionSummary" Desc="Output summary per riterion">
      <where value="true" output_port="0" recProcessed="500000"/>
      <where value="false" output_port="0" recProcessed="500000"/>
    </custom_info>
  </component>
</response>
```

Customizing job monitor messages

WebSphere DataStage specifies the business logic and criterion summary information for the filter operator using the functions addCustomMetadata() and addCustomSummary(). You can also use these functions to generate similar information for the operators you write.

Expressions

The behavior of the filter operator is governed by expressions that you set. You can use the following elements to specify the expressions:

- Fields of the input data set
- Requirements involving the contents of the fields
- Optional constants to be used in comparisons
- The Boolean operators AND and OR to combine requirements

When a record meets the requirements, it is written unchanged to an output data set. Which of the output data sets it is written to is either implied by the order of your -where options or explicitly defined by means of the -target suboption.

The filter operator supports standard SQL expressions, except when comparing strings.

Input data types

If you specify a single field for evaluation, that field can be of any data type. Note that WebSphere DataStage's treatment of strings differs slightly from that of standard SQL.

If you compare fields they must be of the same or compatible data types. Otherwise, the operation terminates with an error. Compatible data types are those that WebSphere DataStage converts by default. Regardless of any conversions the whole record is transferred unchanged to the output. If the fields are not compatible upstream of the filter, you can convert the types by using the modify operator prior to using the filter.

Field data type conversion is based on the following rules:

- Any integer, signed or unsigned, when compared to a floating-point type, is converted to floating-point.
- Comparisons within a general type convert the smaller to the larger size (sfloat to dfloat, uint8 to uint16, and so on)
- When signed and unsigned integers are compared, unsigned are converted to signed.
- Decimal, raw, string, time, date, and timestamp do not figure in type conversions. When any of these is compared to another type, filter returns an error and terminates.

Note: The conversion of numeric data types might result in a loss of range and cause incorrect results. WebSphere DataStage displays a warning messages to that effect when range is lost.

The input field can contain nulls. If it does, null values are less than all non-null values, unless you specify the operators's nulls last option.

Supported Boolean expressions and operators

The following list summarizes the Boolean expressions that WebSphere DataStage supports. In the list, BOOLEAN denotes any Boolean expression.

1. true
2. false
3. six comparison operators: $=$, $<>$, $<$, $>$, \leq , \geq
4. is null
5. is not null
6. like 'abc'
7. The second operand must be a regular expression. See "Regular Expressions" .
8. between (for example, A between B and C is equivalent to $B \leq A$ and $A \leq C$)
9. not BOOLEAN
10. BOOLEAN is true
11. BOOLEAN is false
12. BOOLEAN is not true
13. BOOLEAN is not false

Any of these can be combined using AND or OR.

Regular expressions

The description of regular expressions in this section has been taken from this publication: Rouge Wave[®], Tools.h++.

One-character regular expressions

The following rules determine one-character regular expressions that match a single character:

- Any character that is not a special character matches itself. Special characters are defined below.
- A backslash (\) followed by any special character matches the literal character itself; the backslash "escapes" the special character.
- The special characters are:
+ * ? . [] ^ \$
- The period (.) matches any character except the new line; for example, ".umpty" matches either "Humpty" or "Dumpty".
- A set of characters enclosed in brackets ([]) is a one-character regular expression that matches any of the characters in that set. For example, "[akm]" matches either an "a", "k", or "m".

A range of characters can be indicated with a dash. For example, "[a-z]" matches any lowercase letter. However, if the first character of the set is the caret (^), then the regular expression matches any character except those in the set. It does not match the empty string. For example, "[^akm]" matches any character except "a", "k", or "m". The caret loses its special meaning if it is not the first character of the set.

Multi-character regular expressions

The following rules can be used to build multi-character regular expressions:

- A one-character regular expression followed by an asterisk (*) matches zero or more occurrences of the regular expression. For example, "[a-z]*" matches zero or more lowercase characters.
- A one-character regular expression followed by a plus (+) matches one or more occurrences of the regular expression. For example, "[a-z]+" matches one or more lowercase characters.
- A question mark (?) is an optional element. The preceding regular expression can occur zero or once in the string, no more. For example, "xy?z" matches either "xyz" or "xz".

Order of association

As in SQL, expressions are associated left to right. AND and OR have the same precedence. You might group fields and expressions in parentheses to affect the order of evaluation.

String comparison

WebSphere DataStage operators sort string values according to these general rules:

- Characters are sorted in lexicographic order
- Strings are evaluated by their ASCII value
- Sorting is case sensitive, that is, uppercase letters appear before lowercase letter in sorted data
- Null characters appear before non-null characters in a sorted data set, unless you specify the nulls last option
- Byte-for-byte comparison is performed

Filter example 1: comparing two fields

You want to compare fields A and O. If the data in field A is greater than the data in field O, the corresponding records are to be written to the output data set.

Use the following osh command:

```
$ osh "... | filter -where 'A > O' ..."
```

Example 2: testing for a null

You want to test field A to see if it contains a null. If it does, you want to write the corresponding records to the output data set.

Use the following osh command:

```
$ osh "... | filter -where 'A is null' ..."
```

Example 3: evaluating input records

You want to evaluate each input record to see if these conditions prevail:

- EITHER all the following are true
 - Field A does not have the value 0
 - Field a does not have the value 3
 - Field o has the value 0
- OR field q equals the string 'ZAG'

Here is the osh command for this example:

```
$ osh "... | filter -where 'A <> 0 and a <> 3 and o=0 or  
q = \'ZAG\'' ... "
```

Job scenario: mailing list for a wine auction

The following extended example illustrates the use of the filter operator to extract a list of prospects who should be sent a wine auction catalog, drawn from a large list of leads. A -where clause selects individuals at or above legal drinking age (adult) with sufficient income to be likely to respond to such a catalog (rich).

The example illustrates the use of the where clause by not only producing the list of prospects, but by also producing a list of all individuals who are either adult or rich (or both) and a list of all individuals who are adult.

Schema for implicit import

The example assumes you have created the following schema and stored it as filter_example.schema:

```
record (  
    first_name: string[max=16];  
    last_name: string[max=20];  
    gender: string[1];  
    age: uint8;  
    income: decimal[9,2];  
    state: string[2];  
)
```

OSH syntax

```
osh " filter  
    -where 'age >= 21 and income > 50000.00'  
    -where 'income > 50000.00'  
    -where 'age >= 21' -target 1  
    -where 'age >= 21'  
    < [record@'filter_example.schema']  all12.txt  
0>| AdultAndRich.txt  
1>| AdultOrRich.txt  
2>| Adult.txt  
"
```

The first -where option directs all records that have age ≥ 21 and income > 50000.00 to output 0, which is then directed to the file AdultAndRich.txt.

The second -where option directs all records that have income > 50000.00 to output 1, which is then directed to AdultOrRich.txt.

The third -where option directs all records that have age ≥ 21 also to output 1 (because of the expression -target 1) which is then directed to AdultOrRich.txt.

The result of the second and third -where options is that records that satisfy either of the two conditions income > 50000.00 or age ≥ 21 are sent to output 1. A record that satisfies multiple -where options that are directed to the same output are only written to output once, so the effect of these two options is exactly the same as:

```
-where 'income > 50000.00' or age >= 21'
```

The fourth -where option causes all records satisfying the condition age ≥ 21 to be sent to the output 2, because the last -where option without a -target suboption directs records to output 1. This output is then sent to Adult.txt.

Input data

As a test case, the following twelve data records exist in an input file all12.txt.

```
John Parker M 24 0087228.46 MA
Susan Calvin F 24 0091312.42 IL
William Mandella M 67 0040676.94 CA
Ann Claybourne F 29 0061774.32 FL
Frank Chalmers M 19 0004881.94 NY
Jane Studdock F 24 0075990.80 TX
Seymour Glass M 18 0051531.56 NJ
Laura Engels F 57 0015280.31 KY
John Boone M 16 0042729.03 CO
Jennifer Sarandon F 58 0081319.09 ND
William Tell M 73 0021008.45 SD
Ann Dillard F 21 0004552.65 MI
Jennifer Sarandon F 58 0081319.09 ND
```

Outputs

The following output comes from running WebSphere DataStage. Because of parallelism, the order of the records might be different for your installation. If order matters, you can apply the psort or tsort operator to the output of the filter operator.

After the WebSphere DataStage job is run, the file AdultAndRich.txt contains:

```
John Parker M 24 0087228.46 MA
Susan Calvin F 24 0091312.42 IL
Ann Claybourne F 29 0061774.32 FL
Jane Studdock F 24 0075990.80 TX
Jennifer Sarandon F 58 0081319.09 ND
```

After the WebSphere DataStage job is run, the file AdultOrRich.txt contains:

```
John Parker M 24 0087228.46 MA
Susan Calvin F 24 0091312.42 IL
William Mandella M 67 0040676.94 CA
Ann Claybourne F 29 0061774.32 FL
Jane Studdock F 24 0075990.80 TX
Seymour Glass M 18 0051531.56 NJ
Laura Engels F 57 0015280.31 KY
Jennifer Sarandon F 58 0081319.09 ND
William Tell M 73 0021008.45 SD
Ann Dillard F 21 0004552.65 MI
```

After the WebSphere DataStage job is run, the file Adult.txt contains:

```
John Parker M 24 0087228.46 MA
Susan Calvin F 24 0091312.42 IL
William Mandella M 67 0040676.94 CA
Ann Claybourne F 29 0061774.32 FL
Jane Studdock F 24 0075990.80 TX
Laura Engels F 57 0015280.31 KY
Jennifer Sarandon F 58 0081319.09 ND
William Tell M 73 0021008.45 SD
Ann Dillard F 21 0004552.65 MI
```

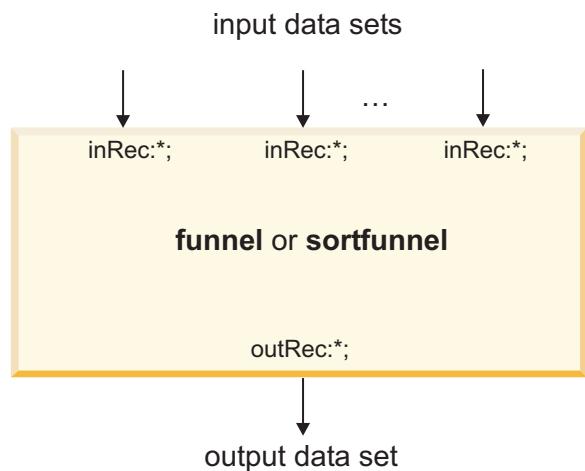
Funnel operators

The funnel operators copy multiple input data sets to a single output data set. This operation is useful for combining separate data sets into a single large data set.

WebSphere DataStage provides two funnel operators:

- The funnel operator combines the records of the input data in no guaranteed order.
- The sortfunnel operator combines the input records in the order defined by the value(s) of one or more key fields and the order of the output records is determined by these sorting keys. By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the sortfunnel operator and other operators.

Data flow diagram



sortfunnel: properties

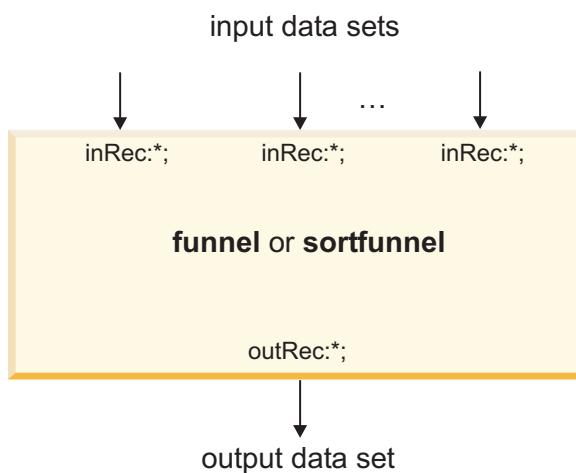
Table 22. sortfunnel properties

Property	Value
Number of input data sets	N (set by user)
Number of output data sets	1
Input interface schema	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec -> outRec without record modification
Execution mode	parallel (default) or sequential
Input partitioning style	sortfunnel operator: keys in same partition
Output partitioning style	sortfunnel operator: distributed keys

Table 22. sortfunnel properties (continued)

Property	Value
Partitioning method	funnel operator: round robin (parallel mode) sortfunnel operator: hash
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	no

Funnel operator



Non-deterministic input sequencing

The funnel operator processes its inputs using non-deterministic selection based on record availability. The funnel operator examines its input data sets in round-robin order. If the current record in a data set is ready for processing, the operator processes it. However, if the current record in a data set is not ready for processing, the operator does not halt execution. Instead, it moves on to the next data set and examines its current record for availability. This process continues until all the records have been transferred to output.

The funnel operator is not combinable.

Syntax

The funnel operator has no options. Its syntax is simply:

funnel

Note: We do not guarantee the output order of records transferred by means of the funnel operator. Use the sortfunnel operator to guarantee transfer order.

Sort funnel operators

Input requirements

The sortfunnel operator guarantees the order of the output records, because it combines the input records in the order defined by the value(s) of one or more key fields. The default partitioner and sort operator with the same keys are automatically inserted before the sortfunnel operator.

The sortfunnel operator requires that the record schema of all input data sets be identical.

A parallel sortfunnel operator uses the default partitioning method local keys. See "The Partitioning Library" for more information on partitioning styles.

Primary and secondary keys

The sortfunnel operator allows you to set one primary key and multiple secondary keys. The sortfunnel operator first examines the primary key in each input record. For multiple records with the same primary key value, the sortfunnel operator then examines secondary keys to determine the order of records it outputs.

For example, the following figure shows the current record in each of three input data sets:

data set 0	data set 1	data set 2
"Jane" "Smith" 42	"Paul" "Smith" 34	"Mary" "Davis" 42

current record

If the data set shown above is sortfunneled on the primary key, LastName, and then on the secondary key, Age, here is the result:

primary key	↓	
"Mary" "Davis" 42		
"Paul" "Smith" 34		
"Jane" "Smith" 42		

key, Age, here is the result:

secondary key

Funnel: syntax and options

The -key option is required. Multiple key options are allowed.

```
sortfunnel -key field [-cs | -ci] [-asc | -desc] [-nulls first | last] [-ebcdic] [-param params]
[-key field [-cs | -ci] [-asc | -desc] [-nulls first | last] [-ebcdic] [-param params] ...]
[-collation_sequence locale | collation_file_pathname | OFF]
```

Table 23. Funnel: syntax and options

Option	Use
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none">• Specify a predefined IBM ICU locale• Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i>• Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.htm</p>

Table 23. Funnel: syntax and options (continued)

Option	Use
-key	<p>-key field [-cs -ci] [-asc -desc] [-nulls first last] [-ebcdic] [-param <i>params</i>]</p> <p>Specifies a key field of the sorting operation. The first -key defines the primary key field of the sort; lower-priority key fields are supplied on subsequent -key specifications.</p> <p>You must define a single primary key to the sortfunnel operator. You can define as many secondary keys as are required by your job. For each key, select the option and supply the field name. Each record field can be used only once as a key. Therefore, the total number of primary and secondary keys must be less than or equal to the total number of fields in the record.</p> <p>-cs -ci are optional arguments for specifying case-sensitive or case-insensitive sorting. By default, the operator uses a case-sensitive algorithm for sorting, that is, uppercase strings appear before lowercase strings in the sorted data set. Specify -ci to override this default and perform case-insensitive sorting of string fields.</p> <p>-asc -desc are optional arguments for specifying ascending or descending sorting. By default, the operator uses ascending sorting order, that is, smaller values appear before larger values in the sorted data set. Specify -desc to sort in descending sorting order instead, so that larger values appear before smaller values in the sorted data set.</p> <p>-nulls first last By default fields containing null values appear first in the sorted data set. To override this default so that fields containing null values appear last in the sorted data set, specify nulls last.</p> <p>-ebcdic By default data is represented in the ASCII character set. To represent data in the EBCDIC character set, specify this option.</p> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using property=value pairs separated by commas.</p>

In this osh example, the sortfunnel operator combines two input data sets into one sorted output data set:

```
$ osh "sortfunnel -key Lastname -key Age < out0.v < out1.v > combined.ds
```

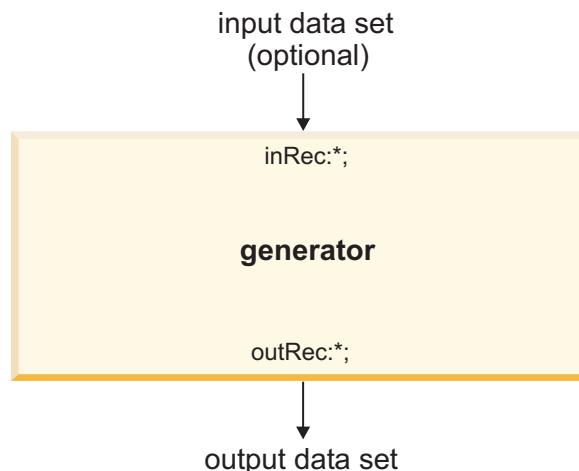
Generator operator

Often during the development of a WebSphere DataStage job, you will want to test the job using valid data. However, you might not have any data available to run the test, your data might be too large to execute the test in a timely manner, or you might not have data with the characteristics required to test the job.

The WebSphere DataStage generator operator lets you create a data set with the record layout that you pass to the operator. In addition, you can control the number of records in the data set, as well as the value of all record fields. You can then use this generated data set while developing, debugging, and testing your WebSphere DataStage job.

To generate a data set, you pass to the operator a schema defining the field layout of the data set and any information used to control generated field values. This topic describes how to use the generator operator, including information on the schema options you use to control the generated field values.

Data flow diagram



generator: properties

Table 24. generator properties

Property	Value
Number of input data sets	0 or 1
Number of output data sets	1
Input interface schema	inRec:*
Output interface schema	supplied_schema; outRec:*
Transfer behavior	inRec -> outRec without record modification
Execution mode	sequential (default) or parallel
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated

Generator: syntax and options

```
generator
-schema schema |
-schemafile filename
[-records num_recs]
[-resetForEachEOW]
```

You must use either the `-schema` or the `-schemafile` argument to specify a schema to the operator. Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

Table 25. generator Operator Options

Option	Use
-schema	<p>-schema <i>schema</i></p> <p>Specifies the schema for the generated data set.</p> <p>You must specify either -schema or -schemafile to the operator.</p> <p>If you supply an input data set to the operator, new fields with the specified schema are prepended to the beginning of each record.</p>
-schemafile	<p>-schemafile <i>filename</i></p> <p>Specifies the name of a file containing the schema for the generated data set.</p> <p>You must specify either -schema or -schemafile to the operator.</p> <p>If you supply an input data set to the operator, new fields with the supplied schema are prepended to the beginning of each record.</p>
-records	<p>-records <i>num_recs</i></p> <p>Specifies the number of records to generate. By default the operator generates an output data set with 10 records (in sequential mode) or 10 records per partition (in parallel mode).</p> <p>If you supply an input data set to the operator, any specification for -records is ignored. In this case, the operator generates one record for each record in the input data set.</p>
-resetForEachEOW	<p>-resetForEachEOW</p> <p>Specifies that the cycle should be repeated for each EOW.</p>

Using the generator operator

During the development of an WebSphere DataStage job, you might find it convenient to execute your job against a data set with a well-defined content. This might be necessary because you want to:

- Run the program against a small number of records to test its functionality
- Control the field values of the data set to examine job output
- Test the program against a variety of data sets
- Run the program but have no available data

You pass to the generator operator a schema that defines the field layout of the data set. By default, the generator operator initializes record fields using a well-defined generation algorithm. For example, an 8-bit unsigned integer field in the first record of a generated data set is set to 0. The field value in each subsequently generated record is incremented by 1 until the generated field value is 255. The field value then wraps back to 0.

However, you can also include information in the schema passed to the operator to control the field values of the generated data set. See "Numeric Fields" for more information on these options.

By default, the operator executes sequentially to generate a data set with a single partition containing 10 records. However, you can configure the operator to generate any number of records. If you configure the operator to execute in parallel, you control the number of records generated in each partition of the output data set.

You can also pass an input data set to the operator. In this case, the operator prepends the generated record fields to the beginning of each record of the input data set to create the output.

Supported data types

The generator operator supports the creation of data sets containing most WebSphere DataStage data types, including fixed-length vectors and nullable fields. However, the generator operator does not support the following data types:

- Variable-length string and ustring types (unless you include a maximum-length specification)
- Variable-length raws (unless you include a maximum-length specification)
- Subrecords
- Tagged aggregates
- Variable-length vectors

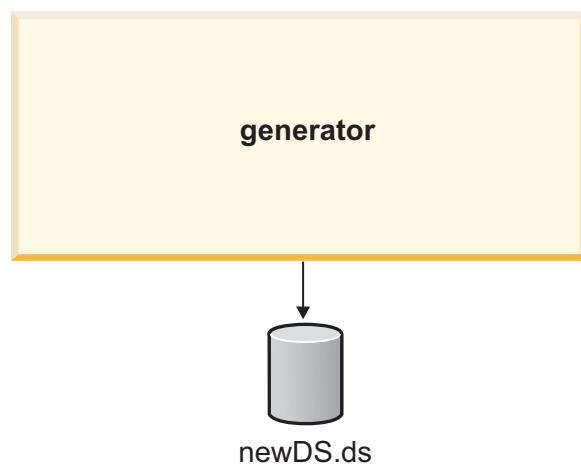
Example 1: using the generator operator

In this example, you use the generator operator to create a data set with 1000 records where each record contains five fields. You also allow the operator to generate default field values.

Here is the schema for the generated data set for this example:

```
record (
    a:int32;
    b:int16;
    c:sfloat;
    d:string[10];
    e:dfloat; )
```

This figure shows the data flow diagram for this example:



To use the generator operator, first configure the schema:

```
$ rec_schema="record
( a:int32;
  b:int16;
  c:sfloat;
  d:string[10];
  e:dfloat; )"
```

Then issue the generator command:

```
$ osh "generator -schema $rec_schema -records 1000 > newDS.ds"
```

This example defines an environment variable (\$rec_schema) to hold the schema passed to the operator.

Alternatively you can specify the name of a file containing the schema, as shown below:

```
$ osh "generator -schemafile s_file.txt -records 1000 > newDS.ds"
```

where the text file s_file.txt contains the schema.

Example 2: executing the operator in parallel

In the previous example, the operator executed sequentially to create an output data set with 1000 records in a single partition. You can also execute the operator in parallel. When executed in parallel, each partition of the generated data set contains the same number of records as determined by the setting for the -records option.

For example, the following osh command executes the operator in parallel to create an output data set with 500 records per partition:

```
$ osh "generator -schemafile s_file -records 500 [par] > newDS.ds"
```

Note that the keyword [par] has been added to the example to configure the generator operator to execute in parallel.

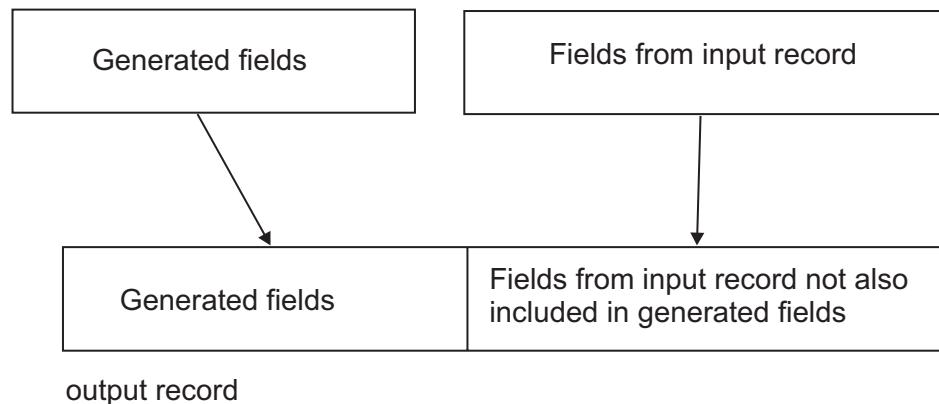
Example 3: using generator with an input data set

You can pass an input data set to the generator operator. In this case, the generated fields are prepended to the beginning of each input record. The operator generates an output data set with the same number of records as the input data set; you cannot specify a record count.

The following command creates an output data set from an input data set and a schema file:

```
$ osh "generator -schemafile s_file [par] < oldDS.ds > newDS.ds"
```

The figure below shows the output record of the generator operator:



For example, you can enumerate the records of a data set by appending an int32 field that cycles from 0 upward.

The generated fields are prepended to the beginning of each record. This means conflicts caused by duplicate field names in the generator schema and the input data set result in the field from the input data set being dropped. Note that WebSphere DataStage issues a warning message to inform you of the naming conflict.

You can use the modify operator to rename the fields in the input data set to avoid the name collision. See "Transform Operator" for more information.

Defining the schema for the operator

The schema passed to the generator operator defines the record layout of the output data set. For example, the previous section showed examples using the following schema:

```
record (
    a:int32;
    b:int16;
    c:sfloat;
    d:string[10];
    e:dfloat; )
```

In the absence of any other specifications in the schema, the operator assigns default values to the fields of the output data set. However, you can also include information in the schema to control the values of the generated fields. This section describes the default values generated for all WebSphere DataStage data types and the use of options in the schema to control field values.

Schema syntax for generator options

You specify generator options within the schema in the same way you specify import/export properties. The following example shows the basic syntax of the generator properties:

```
record (
    a:int32 {generator_options};
    b:int16 {generator_options};
    c:sfloat {generator_options};
    d:string[10] {generator_options};
    e:dfloat {generator_options}; )
```

Note that you include the generator options as part of the schema definition for a field. The options must be included within braces and before the trailing semicolon. Use commas to separate options for fields that accept multiple options.

This table lists all options for the different WebSphere DataStage data types. Detailed information on these options follows the table.

Data Type	Generator Options for the Schema
numeric (also decimal, date, time, timestamp)	cycle = {init = <i>init_val</i> , incr = <i>incr_val</i> , limit = <i>limit_val</i> } random = {limit = <i>limit_val</i> , seed = <i>seed_val</i> , signed}
date	epoch = ' <i>date</i> ' invalids = <i>percentage</i> function = <i>rundate</i>
decimal	zeros = <i>percentage</i> invalids = <i>percentage</i>

Data Type	Generator Options for the Schema
raw	no options available
string	cycle = {value = 'string_1', value = 'string_2', ... } alphabet = 'alpha_numeric_string'
wstring	cycle = {value = 'wstring_1', value = 'wstring_2', ... } alphabet = 'alpha_numeric_wstring'
time	scale = <i>factor</i> invalids = <i>percentage</i>
timestamp	epoch = 'date' scale = <i>factor</i> invalids = <i>percentage</i>
nullable fields	nulls = <i>percentage</i> nullseed = <i>number</i>

Numeric fields

By default, the value of an integer or floating point field in the first record created by the operator is 0 (integer) or 0.0 (float). The field in each successive record generated by the operator is incremented by 1 (integer) or 1.0 (float).

The generator operator supports the use of the cycle and random options that you can use with integer and floating point fields (as well as with all other fields except raw and string). The cycle option generates a repeating pattern of values for a field. The random option generates random values for a field. These options are mutually exclusive; that is, you can only use one option with a field.

- cycle generates a repeating pattern of values for a field. Shown below is the syntax for this option:

```
cycle = {init = init_val, incr = incr_val ,  
         limit = limit_val}
```

where:

- *init_val* is the initial field value (value of the first output record). The default value is 0.
- *incr_val* is the increment value added to produce the field value in the next output record. The default value is 1 (integer) or 1.0 (float).
- *limit_val* is the maximum field value. When the generated field value is greater than *limit_val*, it wraps back to *init_val*. The default value of *limit_val* is the maximum allowable value for the field's data type.

You can specify the keyword part or partcount for any of these three option values. Specifying part uses the partition number of the operator on each processing node for the option value. The partition number is 0 on the first processing node, 1 on the next, and so on. Specifying partcount uses the number of partitions executing the operator for the option value. For example, if the operator executes on four processing nodes, partcount corresponds to a value of 4.

- random generates random values for a field. Shown below is the syntax for this option (all arguments to random are optional):

```
random = {limit = limit_val, seed = seed_val, signed}
```

where:

- *limit_val* is the maximum generated field value. The default value of *limit_val* is the maximum allowable value for the field's data type.

- *seed_val* is the seed value for the random number generator used by the operator for the field. You do not have to specify *seed_val*. By default, the operator uses the same seed value for all fields containing the random option.
- *signed* specifies that signed values are generated for the field (values between *-limit_val* and *+limit_val*.) Otherwise, the operator creates values between 0 and *+limit_val*.

You can also specify the keyword part for *seed_val* and *partcount* for *limit_val*.

For example, the following schema generates a repeating cycle of values for the *AccountType* field and a random number for *balance*:

```
record (
    AccountType:int8 {cycle={init=0, incr=1, limit=24}};
    Balance:dfloat {random={limit=100000, seed=34455}};
)
```

Date fields

By default, a date field in the first record created by the operator is set to January 1, 1960. The field in each successive record generated by the operator is incremented by one day.

You can use the *cycle* and *random* options for date fields as shown above. When using these options, you specify the option values as a number of days. For example, to set the increment value for a date field to seven days, you use the following syntax:

```
record (
    transDate:date {cycle={incr=7}};
    transAmount:dfloat {random={limit=100000, seed=34455}};
)
```

In addition, you can use the following options: *epoch*, *invalids*, and *functions*.

The *epoch* option sets the earliest generated date value for a field. You can use this option with any other date options. The syntax of *epoch* is:

```
epoch = 'date'
```

where *date* sets the earliest generated date for the field. The *date* must be in yyyy-mm-dd format and leading zeros must be supplied for all portions of the date. If an epoch is not specified, the operator uses 1960-01-01.

For example, the following schema sets the initial field value of *transDate* to January 1, 1998:

```
record (
    transDate:date {epoch='1998-01-01'};
    transAmount:dfloat {random={limit=100000, seed=34455}};
)
```

You can also specify the *invalids* option for a date field. This option specifies the percentage of generated fields containing invalid dates:

```
invalids = percentage
```

where *percentage* is a value between 0.0 and 100.0. WebSphere DataStage operators that process date fields can detect an invalid date during processing.

The following example causes approximately 10% of *transDate* fields to be invalid:

```
record (
    transDate:date {epoch='1998-01-01', invalids=10.0};
    transAmount:dfloat {random={limit=100000, seed=34455}};
)
```

You can use the *function* option to set date fields to the current date:

```
function = rundate
```

There must be no other options specified to a field using function. The following schema causes *transDate* to have the current date in all generated records:

```
record (
    transDate:date {function=rundate};
    transAmount:dfloat {random={limit=100000, seed=34455}};
)
```

Decimal fields

By default, a decimal field in the first record created by the operator is set to 0. The field in each successive record generated by the operator is incremented by 1. The maximum value of the decimal is determined by the decimal's scale and precision. When the maximum value is reached, the decimal field wraps back to 0.

You can use the cycle and random options with decimal fields. See "Numeric Fields" for information on these options. In addition, you can use the zeros and invalids options with decimal fields. These options are described below.

The zeros option specifies the percentage of generated decimal fields where all bytes of the decimal are set to binary zero (0x00). Many operations performed on a decimal can detect this condition and either fail or return a flag signifying an invalid decimal value. The syntax for the zeros options is:

```
zeros = percentage
```

where *percentage* is a value between 0.0 and 100.0.

The invalids options specifies the percentage of generated decimal fields containing and invalid representation of 0xFF in all bytes of the field. Any operation performed on an invalid decimal detects this condition and either fails or returns a flag signifying an invalid decimal value. The syntax for invalids is:

```
invalids = percentage
```

where *percentage* is a value between 0.0 and 100.0.

If you specify both zeros and invalids, the percentage for invalids is applied to the fields that are not first made zero. For example, if you specify zeros=50 and invalids=50, the operator generates approximately 50% of all values to be all zeros and only 25% (50% of the remainder) to be invalid.

Raw fields

You can use the generator operator to create fixed-length raw fields or raw fields with a specified maximum length; you cannot use the operator to generate variable-length raw fields. If the field has a maximum specified length, the length of the string is a random number between 1 and the maximum length.

Maximum-length raw fields are variable-length fields with a maximum length defined by the max parameter in the form:

```
max_r:raw [max=10];
```

By default, all bytes of a raw field in the first record created by the operator are set to 0x00. The bytes of each successive record generated by the operator are incremented by 1 until a maximum value of 0xFF is reached. The operator then wraps byte values to 0x00 and repeats the cycle.

You cannot specify any options to raw fields.

String fields

You can use the generator operator to create fixed-length string and ustring fields or string and ustring fields with a specified maximum length; you cannot use the operator to generate variable-length string fields. If the field has a maximum specified length, the length of the string is a random number between 0 and the maximum length.

Note that maximum-length string fields are variable-length fields with a maximum length defined by the max parameter in the form:

```
max_s: string [max=10];
```

In this example, the field max_s is variable length up to 10 bytes long.

By default, the generator operator initializes all bytes of a string field to the same alphanumeric character. When generating a string field, the operators uses the following characters, in the following order:

```
abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

For example, the following field specification:

```
s: string[5];
```

produces successive string fields with the values:

```
aaaaa  
bbbbb  
ccccc  
ddddd  
...
```

After the last character, capital Z, values wrap back to lowercase a and the cycle repeats.

Note: The alphabet property for ustring values accepts Unicode characters.

You can use the alphabet property to define your own list of alphanumeric characters used for generated string fields:

```
alphabet = 'alpha_numeric_string'
```

This option sets all characters in the field to successive characters in the *alpha_numeric_string*. For example, this field specification:

```
s: string[3] {alphabet='abc'};
```

produces strings with the following values:

```
aaa  
bbb  
ccc  
aaa  
...
```

Note: The cycle option for ustring values accepts Unicode characters.

The cycle option specifies the list of string values assigned to generated string field:

```
cycle = { value = 'string_1', value = 'string_2', ... }
```

The operator assigns *string_1* to the string field in the first generated record, *string_2* to the field in the second generated record, and so on. In addition:

- If you specify only a single value, all string fields are set to that value.

- If the generated string field is fixed length, the value string is truncated or padded with the default pad character 0x00 to the fixed length of the string.
- If the string field contains a maximum length setting, the length of the string field is set to the length of the value string. If the length of the value string is longer than the maximum string length, the value string is truncated to the maximum length.

Time fields

By default, a time field in the first record created by the operator is set to 00:00:00 (midnight). The field in each successive record generated by the operator is incremented by one second. After reaching a time of 23:59:59, time fields wrap back to 00:00:00.

You can use the cycle and random options with time fields. See "Numeric Fields" for information on these options. When using these options, you specify the options values in numbers of seconds. For example, to set the value for a time field to a random value between midnight and noon, you use the following syntax:

```
record (
    transTime:time {random={limit=43200, seed=83344}};
)
```

For a time field, midnight corresponds to an initial value of 0 and noon corresponds to 43,200 seconds (12 hours * 60 minutes * 60 seconds).

In addition, you can use the scale and invalids options with time fields.

The scale option allows you to specify a multiplier to the increment value for time. The syntax of this options is:

```
scale = factor
```

The increment value is multiplied by *factor* before being added to the field. For example, the following schema generates two time fields:

```
record (
    timeMinutes:time {scale=60};
    timeSeconds:time;
)
```

In this example, the first field increments by 60 seconds per record (one minute), and the second field increments by seconds.

You use the invalids option to specify the percentage of invalid time fields generated:

```
invalids = percentage
```

where *percentage* is a value between 0.0 and 100.0. The following schema generates two time fields with different percentages of invalid values:

```
record (
    timeMinutes:time {scale=60, invalids=10};
    timeSeconds:time {invalids=15};
)
```

Timestamp fields

A timestamp field consists of both a time and date portion. Timestamp fields support all valid options for both date and time fields. See "Date Fields" or "Time Fields" for more information.

By default, a timestamp field in the first record created by the operator is set to 00:00:00 (midnight) on January 1, 1960. The time portion of the timestamp is incremented by one second for each successive record. After reaching a time of 23:59:59, the time portion wraps back to 00:00:00 and the date portion increments by one day.

Null fields

By default, schema fields are not nullable. Specifying a field as nullable allows you to use the nulls and nullseed options within the schema passed to the generator operator.

Note: If you specify these options for a non-nullable field, the operator issues a warning and the field is set to its default value.

The nulls option specifies the percentage of generated fields that are set to null:

`nulls = percentage`

where *percentage* is a value between 0.0 and 100.0.

The following example specifies that approximately 15% of all generated records contain a null for field a:

```
record (
    a:nullable int32 {random={limit=100000, seed=34455}, nulls=15.0};
    b:int16; )
```

The nullseed option sets the seed for the random number generator used to decide whether a given field will be null.

`nullseed = seed`

where *seed* specifies the seed value and must be an integer larger than 0.

In some cases, you might have multiple fields in a schema that support nulls. You can set all nullable fields in a record to null by giving them the same nulls and nullseed values. For example, the following schema defines two fields as nullable:

```
record (
    a:nullable int32 {nulls=10.0, nullseed=5663};
    b:int16;
    c:nullable sfloat {nulls=10.0, nullseed=5663};
    d:string[10];
    e:dfloat; )
```

Since both fields a and c have the same settings for nulls and nullseed, whenever one field in a record is null the other is null as well.

Head operator

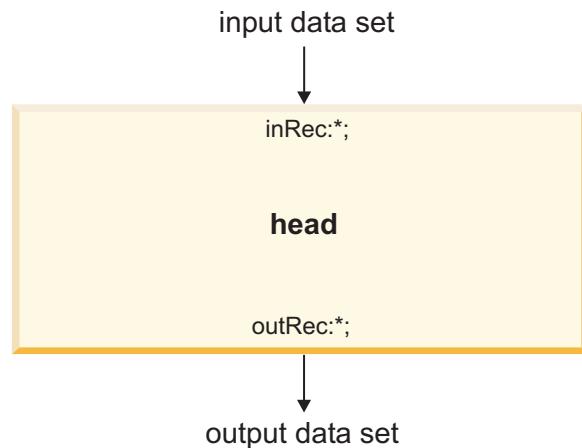
The head operator selects the first *n* records from each partition of an input data set and copies the selected records to an output data set. By default, *n* is 10 records. However, you can determine the following by means of options:

- The number of records to copy
- The partition from which the records are copied
- The location of the records to copy
- The number of records to skip before the copying operation begins.

This control is helpful in testing and debugging jobs with large data sets. For example, the `-part` option lets you see data from a single partition to ascertain if the data is being partitioned as you want. The `-skip` option lets you access a portion of a data set.

The tail operator performs a similar operation, copying the last n records from each partition. See "Tail Operator".

Data flow diagram



head: properties

Table 26. head properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	<code>inRec:*</code>
Output interface schema	<code>outRec:*</code>
Transfer behavior	<code>inRec -> outRec without record modification</code>

Head: syntax and options

```
head
[-all | -nrecs count]
[-part partition_number]
[-period P]
[-skip recs]
```

Table 27. Head options

Option	Use
<code>-all</code>	<code>-all</code> Copy all input records to the output data set. You can skip records before head performs its copy operation by means of the <code>-skip</code> option. You cannot select the <code>-all</code> option and the <code>-nrecs</code> option at the same time.

Table 27. Head options (continued)

-nrecs	-nrecs <i>count</i> Specify the number of records (<i>count</i>) to copy from each partition of the input data set to the output data set. The default value of <i>count</i> is 10. You cannot specify this option and the -all option at the same time.
-part	-part <i>partition_number</i> Copy records only from the indicated partition, <i>partition_number</i> . By default, the operator copies records from all partitions. You can specify -part multiple times to specify multiple partition numbers. Each time you do, specify the option followed by the number of the partition.
-period	-period <i>P</i> Copy every <i>P</i> th record in a partition, where <i>P</i> is the period. You can start the copy operation after records have been skipped (as defined by -skip). <i>P</i> must equal or be greater than 1. The default value of <i>P</i> is 1.
-skip	-skip <i>recs</i> Ignore the first <i>recs</i> records of each partition of the input data set, where <i>recs</i> is the number of records to skip. The default skip count is 0.

Head example 1: head operator default behavior

In this example, no options have been specified to the head operator. The input data set consists of 100 sorted positive integers hashed into four partitions. The output data set consists of the first ten integers of each partition. The next table lists the input and output data sets by partition.

The osh command is:

```
$osh "head < in.ds > out.ds"
```

Partition 0		Partition 1		Partition 2		Partition 3	
Input	Output	Input	Output	Input	Output	Input	Output
0	0	3	3	6	6	1	1
9	9	5	5	7	7	2	2
18	18	11	11	8	8	4	4
19	19	12	12	22	22	10	10
23	23	13	13	29	29	20	20
25	25	14	14	30	30	21	21
36	36	15	15	33	33	24	24
37	37	16	16	41	41	26	26
40	40	17	17	43	43	27	27
47	47	35	35	44	44	28	28
51				45		31	
				48		32	
				55		34	
				56		38	
				58		39	
						52	
						54	

Example 2: extracting records from a large data set

In this example you use the head operator to extract the first 1000 records of each partition of a large data set, in.ds.

To perform this operation, use the osh command:

```
$ osh "head -nrecs 1000 < in.ds > out0.ds"
```

For example, if in.ds is a data set of one megabyte, with 2500 K records, out0.ds is a data set of 15.6 kilobytes with 4K records.

Example 3: locating a single record

In this example you use head to extract a single record from a particular partition to diagnose the record.

The osh command is:

```
$ osh "head -nrecs 1 -skip 1234 -part 2 < in.ds > out0.ds"
```

Lookup operator

With the lookup operator, you can create and use lookup tables to modify your input data set. For example, you could map a field that should contain a two letter U. S. state postal code to the name of the state, adding a *FullStateName* field to the output schema.

The operator performs in two modes: lookup mode and create-only mode:

- In lookup mode, the operator takes as input a single source data set, one or more lookup tables represented as WebSphere DataStage data sets, and one or more file sets. A file set is a lookup table that contains key-field information. There must be at least one lookup table or file set.

For each input record of the source data set, the operator performs a table lookup on each of the input lookup tables. The table lookup is based on the values of a set of lookup key fields, one set for each table. A source record and lookup record correspond when all of the specified lookup key fields have matching values.

Each record of the output data set contains all of the fields from a source record. Concatenated to the end of the output records are the fields from all the corresponding lookup records where corresponding source and lookup records have the same value for the lookup key fields.

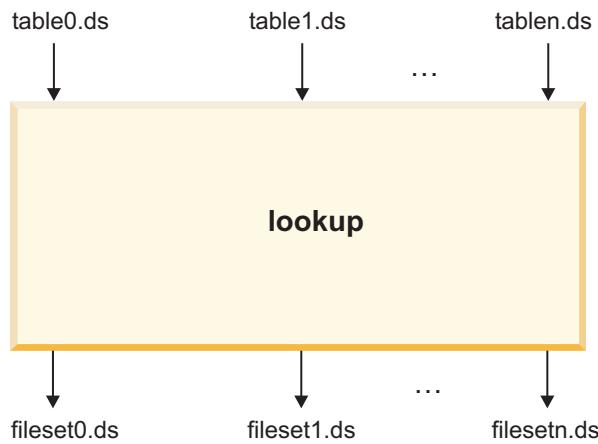
The reject data set is an optional second output of the operator. This data set contains source records that do not have a corresponding entry in every input lookup table.

- In create-only mode, you use the -createOnly option to create lookup tables without doing the lookup processing step. This allows you to make and save lookup tables that you expect to need at a later time, making for faster start-up of subsequent lookup operations.

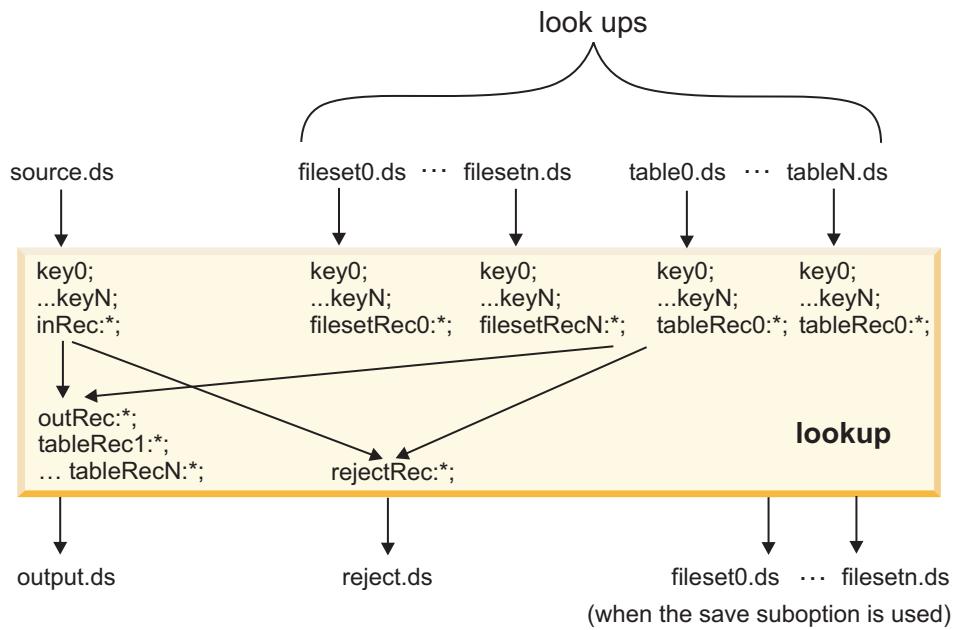
The lookup operator is similar in function to the merge operator and the join operators. To understand the similarities and differences see "Comparison with Other Operators".

Data flow diagrams

Create-only mode



Lookup mode



lookup: properties

Table 28. *lookup* properties

Property	Normal mode	Create-only mode
Number of input data sets	$T + 1$	T
Number of output data sets	1 or 2 (output and optional reject)	0
Input interface schema input data set: lookup data sets:	<code>key0:data_type; ... keyN:data_type;</code> <code>inRec:*</code> <code>key0:data_type; ... keyM:data_type;</code> <code>tableRec:*</code>	<code>n/a</code> <code>key0:data_type; ... keyM:data_type;</code> <code>tableRec:*</code>

Table 28. *lookup properties (continued)*

Property	Normal mode	Create-only mode
Output interface schema output data set: reject data sets:	outRec:*; with lookup fields missing from the input data set concatenated rejectRec:*	n/a
Transfer behavior source to output: lookup to output: source to reject: table to file set	inRec -> outRec without record modification tableRecN -> tableRecN, minus lookup keys and other duplicate fields inRec -> rejectRec without record modification (optional) key-field information is added to the table	n/a key-field information is added to the table
Partitioning method	any (parallel mode); the default for table inputs is entire	any (default is entire)
Collection method	any (sequential mode)	any
Preserve-partitioning flag in output data set	propagated	n/a
Composite operator	yes	yes

Lookup: syntax and options

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

The syntax for the lookup operator in an osh command has two forms, depending on whether you are only creating one or more lookup tables or doing the lookup (matching) itself and creating file sets or using existing file sets.

```
lookup -createOnly
-table -key field [-cs | -ci] [-param parameters]
  [-key field [-cs | -ci] [-param parameters]...]
  [-allow_dups]
  -save lookup_fileset
  [-diskpool pool]
[-table -key field [-cs | -ci] [-param parameters]
  [-key field [-cs | -ci] [-param parameters]...]
  [-allow_dups]
  -save fileset_descriptor [-diskpool pool] ...]
```

or

```
lookup [-fileset fileset_descriptor]
  [-collation_sequence locale | collation_file_pathname | OFF]
  [-table key_specifications [-allow_dups]
  -save fileset_descriptor
  [-diskpool pool]...]
  [-ifNotFound continue | drop | fail | reject]
```

where a fileset, or a table, or both, must be specified, and *key_specifications* is a list of one or more strings of this form:

-key *field* [-cs | -ci]

Table 29. Lookup options

Option	Use
- <i>collation_sequence</i>	<p>-<i>collation_sequence locale collation_file_pathname OFF</i></p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> • Specify a predefined IBM ICU locale • Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> • Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>
- <i>createOnly</i>	- <i>createOnly</i> <p>Specifies the creation of one or more lookup tables; no lookup processing is to be done.</p>
- <i>fileset</i>	<p>[<i>-fileset fileset_descriptor ...</i>]</p> <p>Specify the name of a fileset containing one or more lookup tables to be matched. These are tables that have been created and saved by an earlier execution of the lookup operator using the -<i>createOnly</i> option.</p> <p>In lookup mode, you must specify either the -<i>fileset</i> option, or a table specification, or both, in order to designate the lookup table(s) to be matched against. There can be zero or more occurrences of the -<i>fileset</i> option. It cannot be specified in create-only mode.</p> <p>Warning: The fileset already contains key specifications. When you follow -<i>fileset fileset_descriptor</i> by <i>key_specifications</i>, the keys specified do not apply to the fileset; rather, they apply to the first lookup table. For example, <i>lookup -fileset file -key field</i>, is the same as:</p> <pre>lookup -fileset file1 -table -key field</pre>

Table 29. Lookup options (continued)

Option	Use
-ifNotFound	<p>-ifNotFound continue drop fail reject</p> <p>Specifies the operator action when a record of an input data set does not have a corresponding record in every input lookup table. The default action of the operator is to fail and terminate the step.</p> <p>continue tells the operator to continue execution when a record of an input data set does not have a corresponding record in every input lookup table. The input record is transferred to the output data set along with the corresponding records from the lookup tables that matched. The fields in the output record corresponding to the lookup table(s) with no corresponding record are set to their default value or null if the field supports nulls.</p> <p>drop tells the operator to drop the input record (refrain from creating an output record).</p> <p>fail sets the operator to abort. This is the default.</p> <p>reject tells the operator to copy the input record to the reject data set. In this case, a reject output data set must be specified.</p>

Table 29. Lookup options (continued)

Option	Use
-table	<p>-table -key field [-ci -cs] [-param parameters] [-key field [-ci cs] [-param parameters] ...] [-allow_dups] -save fileset_descriptor [-diskpool pool] ...]</p> <p>Specifies the beginning of a list of key fields and other specifications for a lookup table. The first occurrence of -table marks the beginning of the key field list for lookup table1; the next occurrence of -table marks the beginning of the key fields for lookup table2, and so on. For example:</p> <pre>lookup -table -key field -table -key field</pre> <p>The -key option specifies the name of a lookup key field. The -key option must be repeated if there are multiple key fields. You must specify at least one key for each table. You cannot use a vector, subrecord, or tagged aggregate field as a lookup key.</p> <p>The -ci suboption specifies that the string comparison of lookup key values is to be case insensitive; the -cs option specifies case-sensitive comparison, which is the default.</p> <p>The -params suboption provides extra parameters for the lookup key. Specify property=value pairs, without curly braces.</p> <p>In create-only mode, the -allow_dups option causes the operator to save multiple copies of duplicate records in the lookup table without issuing a warning. Two lookup records are duplicates when all lookup key fields have the same value in the two records. If you do not specify this option, WebSphere DataStage issues a warning message when it encounters duplicate records and discards all but the first of the matching records.</p> <p>In normal lookup mode, only one lookup table (specified by either -table or -fileset) can have been created with -allow_dups set.</p> <p>The -save option lets you specify the name of a fileset to write this lookup table to; if -save is omitted, tables are written as scratch files and deleted at the end of the lookup. In create-only mode, -save is, of course, required.</p> <p>The -diskpool option lets you specify a disk pool in which to create lookup tables. By default, the operator looks first for a "lookup" disk pool, then uses the default pool (""). Use this option to specify a different disk pool to use.</p>

Lookup table characteristics

The lookup tables input to the operator are created from WebSphere DataStage data sets. The lookup tables do not have to be sorted and should be small enough that all tables fit into physical memory on the processing nodes in your system. Lookup tables larger than physical memory do not cause an error, but they adversely affect the execution speed of the operator.

The memory used to hold a lookup table is shared among the lookup processes running on each machine. Thus, on an SMP, all instances of a lookup operator share a single copy of the lookup table, rather than having a private copy of the table for each process. This reduces memory consumption to that of a single sequential lookup process. This is why partitioning the data, which in a non-shared-memory environment saves memory by creating smaller tables, also has the effect of disabling this memory sharing, so that there is no benefit to partitioning lookup tables on an SMP or cluster.

Partitioning

Normally (and by default), lookup tables are partitioned using the entire partitioning method so that each processing node receives a complete copy of the lookup table. You can partition lookup tables using another partitioning method, such as hash, as long as you ensure that all records with the same lookup keys are partitioned identically. Otherwise, source records might be directed to a partition that doesn't have the proper table entry.

For example, if you are doing a lookup on keys a, b, and c, having both the source data set and the lookup table hash partitioned on the same keys would permit the lookup tables to be broken up rather than copied in their entirety to each partition. This explicit partitioning disables memory sharing, but the lookup operation consumes less memory, since the entire table is not duplicated. Note, though, that on a single SMP, hash partitioning does not actually save memory. On MPPs, or where shared memory can be used only in a limited way, or not at all, it can be beneficial.

Create-only mode

In its normal mode of operation, the lookup operator takes in a source data set and one or more data sets from which the lookup tables are built. The lookup tables are actually represented as file sets, which can be saved if you wish but which are normally deleted as soon as the lookup operation is finished. There is also a mode, selected by the -createOnly option, in which there is no source data set; only the data sets from which the lookup tables are to be built are used as input. The resulting file sets, containing lookup tables, are saved to persistent storage.

This create-only mode of operation allows you to build lookup tables when it is convenient, and use them for doing lookups at a later time. In addition, initialization time for the lookup processing phase is considerably shorter when lookup tables already exist.

For example, suppose you have data sets data1.ds and data2.ds and you want to create persistent lookup tables from them using the name and ID fields as lookup keys in one table and the name and accountType fields in the other.

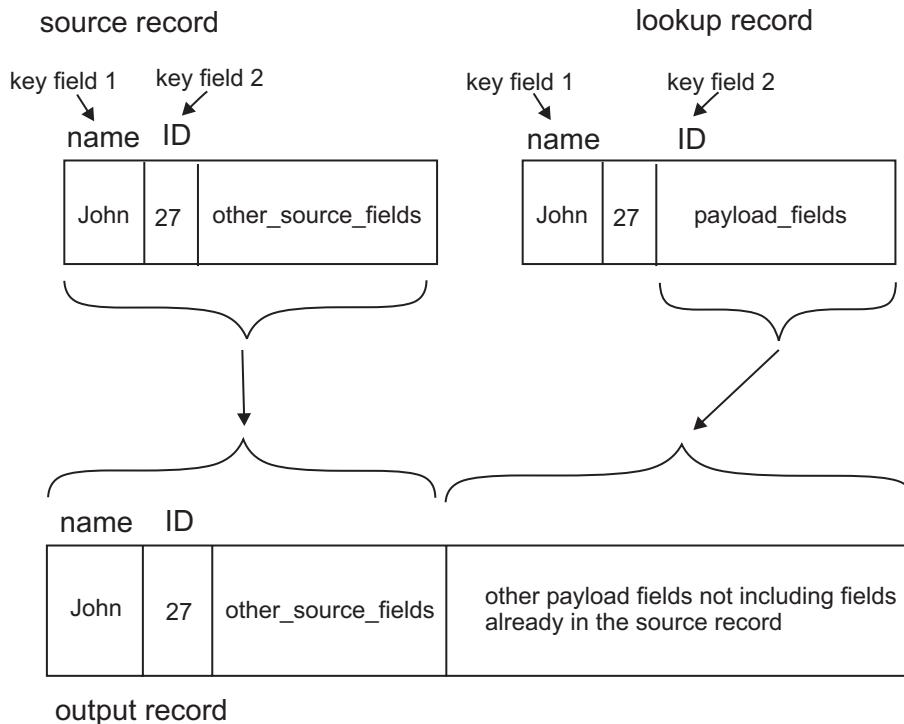
For this use of the lookup operator, you specify the -createOnly option and two -table options. In this case, two suboptions for the -table options are specified: -key and -save.

In osh, use the following command:

```
$ osh " lookup -createOnly -table -key name -key ID
      -save fs1.fs
      -table -key name -key accountType -save fs2.fs
      < data1.ds < data2.ds"
```

Lookup example 1: single lookup table record

This figure shows the lookup of a source record and a single lookup table record:



This figure shows the source and lookup record and the resultant output record. A source record and lookup record are matched if they have the same values for the key field(s). In this example, both records have John as the name and 27 as the ID number. In this example, the lookup keys are the first fields in the record. You can use any field in the record as a lookup key.

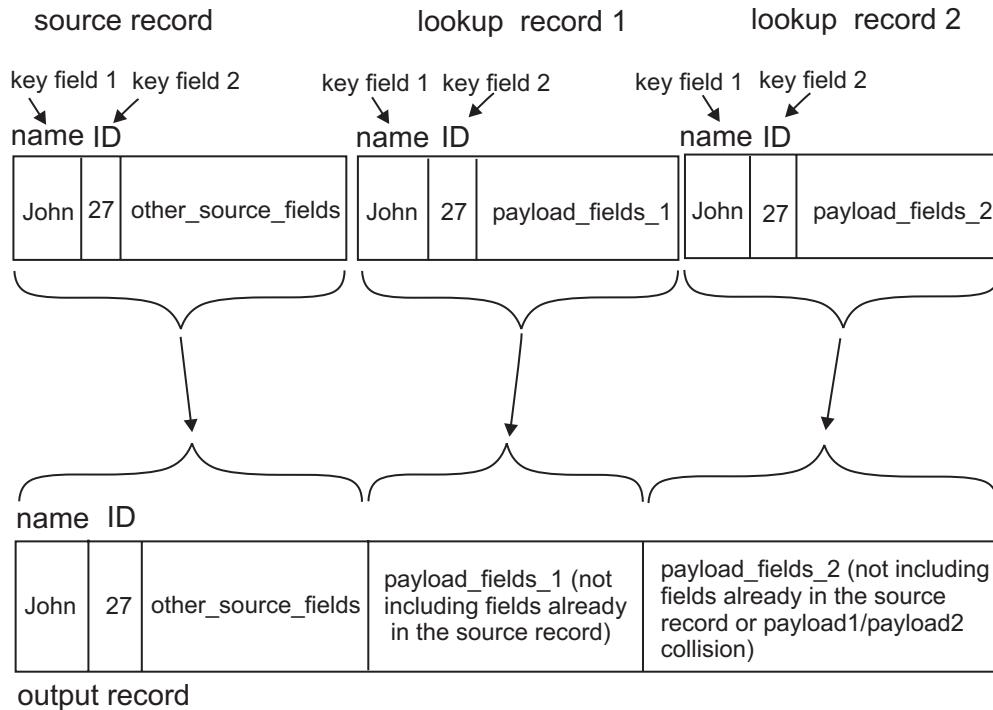
Note that fields in a lookup table that match fields in the source record are dropped. That is, the output record contains all of the fields from the source record plus any fields from the lookup record that were not in the source record. Whenever any field in the lookup record has the same name as a field in the source record, the data comes from the source record and the lookup record field is ignored.

Here is the command for this example:

```
$ osh "lookup -table -key Name -key ID  
< inSrc.ds < inLU1.ds > outDS.ds"
```

Example 2: multiple lookup table record

When there are multiple lookup tables as input, the lookup tables can all use the same key fields, or they can use different sets. The diagram shows the lookup of a source record and two lookup records where both lookup tables have the same key fields.



The osh command for this example is:

```
$ osh " lookup -table -key name -key ID -table -key name -key ID
      < inSrc.ds < inLU1.ds < inLU2.ds > outDS.ds"
```

Note that in this example you specify the same key fields for both lookup tables.

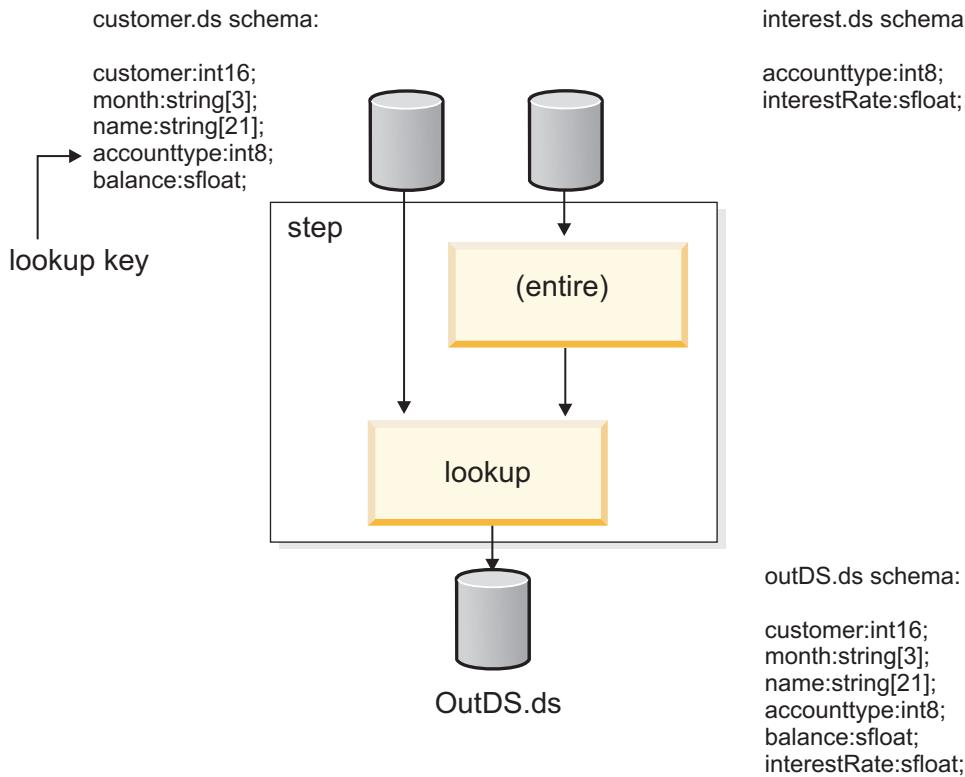
Alternatively, you can specify a different set of lookup keys for each lookup table. For example, you could use name and ID for the first lookup table and the fields accountType and minBalance (not shown in the figure) for the second lookup table. Each of the resulting output records would contain those four fields, where the values matched appropriately, and the remaining fields from each of the three input records.

Here is the osh command for this example:

```
$ osh " lookup -table -key name -key ID -table -key accountType
      -key minBalance < inSrc.ds < inLU1.ds < inLU2.ds
      > outDS.ds"
```

Example 3: interest rate lookup example

The following figure shows the schemas for a source data set customer.ds and a lookup data set interest.ds. This operator looks up the interest rate for each customer based on the customer's account type. In this example, WebSphere DataStage inserts the entire partitioner (this happens automatically; you do not need to explicitly include it in your program) so that each processing node receives a copy of the entire lookup table.



Since the interest rate is not represented in the source data set record schema, the *interestRate* field from the lookup record has been concatenated to the source record.

Here is the osh code for this example:

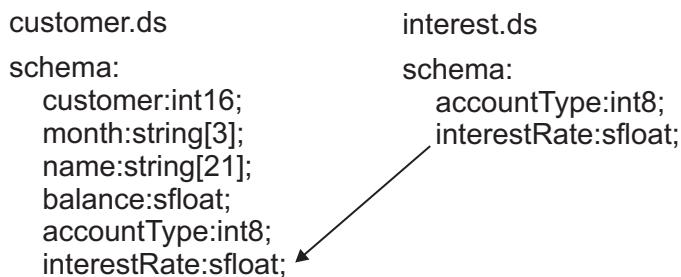
```
$ osh " lookup -table -key accountType < customers.ds
      < interest.ds      > outDS.ds"
```

Example 4: handling duplicate fields example

If, in the previous example, the record schema for customer.ds also contained a field named *interestRate*, both the source and the lookup data sets would have a non-lookup-key field with the same name. By default, the *interestRate* field from the source record is output to the lookup record and the field from the lookup data set is ignored.

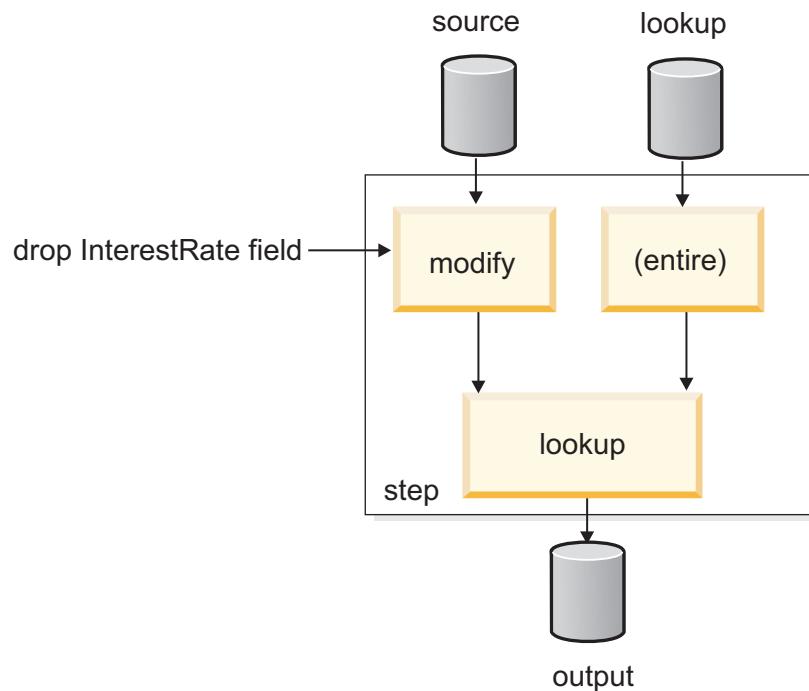
If you want the *interestRate* field from the lookup data set to be output, rather than the value from the source record, you can use a modify operator before the lookup operator to drop the *interestRate* field from the source record.

The following diagram shows record schemas for the customer.ds and interest.ds in which both schemas have a field named *interestRate*.



To make the lookup table's interestRate field the one that is retained in the output, use a modify operator to drop interestRate from the source record. The interestRate field from the lookup table record is propagated to the output data set, because it is now the only field of that name.

The following figure shows how to use a modify operator to drop the interestRate field:



The osh command for this example is:

```
$ osh " modify -spec 'drop interestRate;' < customer.ds |
      lookup -table -key accountType < interest.ds > outDS.ds"
```

Note that this is unrelated to using the `-allow_dups` option on a table, which deals with the case where two records in a lookup table are identical in all the key fields.

Merge operator

The merge operator combines a sorted master data set with one or more sorted update data sets. The fields from the records in the master and update data sets are merged so that the output record contains all the fields from the master record plus any additional fields from matching update record.

A master record and an update record are merged only if both of them have the same values for the merge key field(s) that you specify. Merge key fields are one or more fields that exist in both the master and update records.

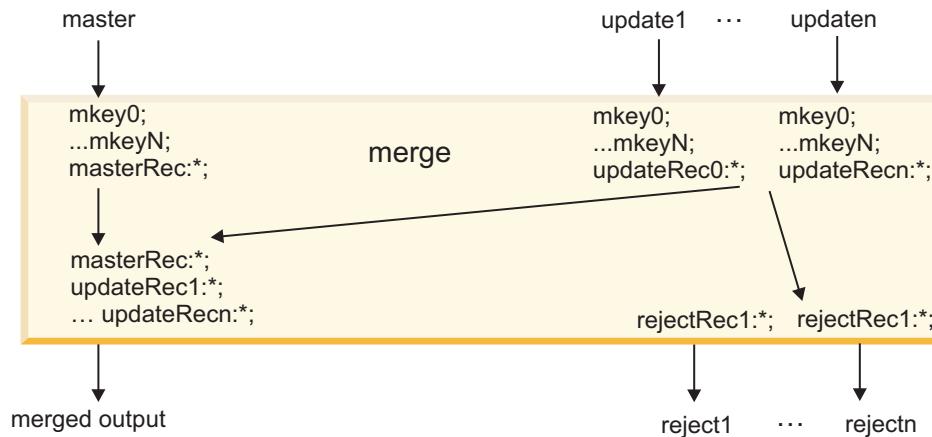
By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the merge operator and other operators.

As part of preprocessing your data for the merge operator, you must remove duplicate records from the master data set. If you have more than one update data set, you must remove duplicate records from the update data sets as well. This section describes how to use the merge operator. Included in this topic are examples using the remdup operator to preprocess your data.

The merge operator is similar in function to the lookup operator and the join operators. To understand the similarities and differences see "Comparison with Other Operators".

Data flow diagram

The merge operator merges a master data set with one or more update data sets



merge: properties

Table 30. merge properties

Property	Value
Number of input data sets	1 master; 1-n update
Number of output data sets	1 output; 1-n reject (optional)
Input interface schema	mKey0:data_type; ... mKeyk:data_type; masterRec.*;
master data set:	mKey0:data_type; ... mKeyk:data_type; updateRecr.*;
update data sets:	rejectRecr.*;
Output interface schema	masterRec.*; updateRec1.*; updateRec2.*; ... updateRecn.*;
output data set:	rejectRecn.*
reject data sets:	
Transfer behavior	masterRec -> masterRec without record modification
master to output:	updateRecn-> outputRejectRecn -> updateRecn -> rejectRecn without record modification (optional)
update to output: update to reject:	
Input partitioning style	keys in same partition
Output partitioning style	distributed keys
Execution mode	parallel (default) or sequential
Preserve-partitioning flag in output data set	propagated

Merge: syntax and options

```

merge
-key field [-ci | -cs] [-asc | -desc] [-ebcdic] [-nulls first | last] [param params]
[-key field [-ci | -cs] [-asc | -desc] [-ebcdic] [-nulls first | last] [param params] ...]
[-collation_sequence locale | collation_file_pathname | OFF]
[-dropBadMasters | -keepBadMasters]
[-nowarnBadMasters | -warnBadMasters]
[-nowarnBadUpdates | -warnBadUpdates]
  
```

Table 31. Merge options

Option	Use
-key	<p>-key <i>field</i> [-ci -cs] [-asc -desc] [-ebcdic] [-nulls first last] [param <i>params</i>] [-key <i>field</i> [-ci -cs] [-asc -desc] [-ebcdic] [-nulls first last] [param <i>params</i>] ...]</p> <p>Specifies the name of a merge key field. The -key option might be repeated if there are multiple merge key fields.</p> <p>The -ci option specifies that the comparison of merge key values is case insensitive. The -cs option specifies a case-sensitive comparison, which is the default.</p> <p>-asc -desc are optional arguments for specifying ascending or descending sorting. By default, the operator uses ascending sorting order, that is, smaller values appear before larger values in the sorted data set. Specify -desc to sort in descending sorting order instead, so that larger values appear before smaller values in the sorted data set.</p> <p>-nulls first last. By default fields containing null values appear first in the sorted data set. To override this default so that fields containing null values appear last in the sorted data set, specify nulls last.</p> <p>-ebcdic. By default data is represented in the ASCII character set. To represent data in the EBCDIC character set, specify this option.</p> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property=value</i> pairs separated by commas.</p>
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> Specify a predefined IBM ICU locale Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.htm</p>
-dropBadMasters	<p>-dropBadMasters</p> <p>Rejected masters are not output to the merged data set.</p>
-keepBadMasters	<p>-keepBadMasters</p> <p>Rejected masters are output to the merged data set. This is the default.</p>

Table 31. Merge options (continued)

Option	Use
-nowarnBadMasters	-nowarnBadMasters Do not warn when rejecting bad masters.
-nowarnBadUpdates	-nowarnBadUpdates Do not warn when rejecting bad updates.
-warnBadMasters	-warnBadMasters Warn when rejecting bad masters. This is the default.
-warnBadUpdates	-warnBadUpdates Warn when rejecting bad updates. This is the default.

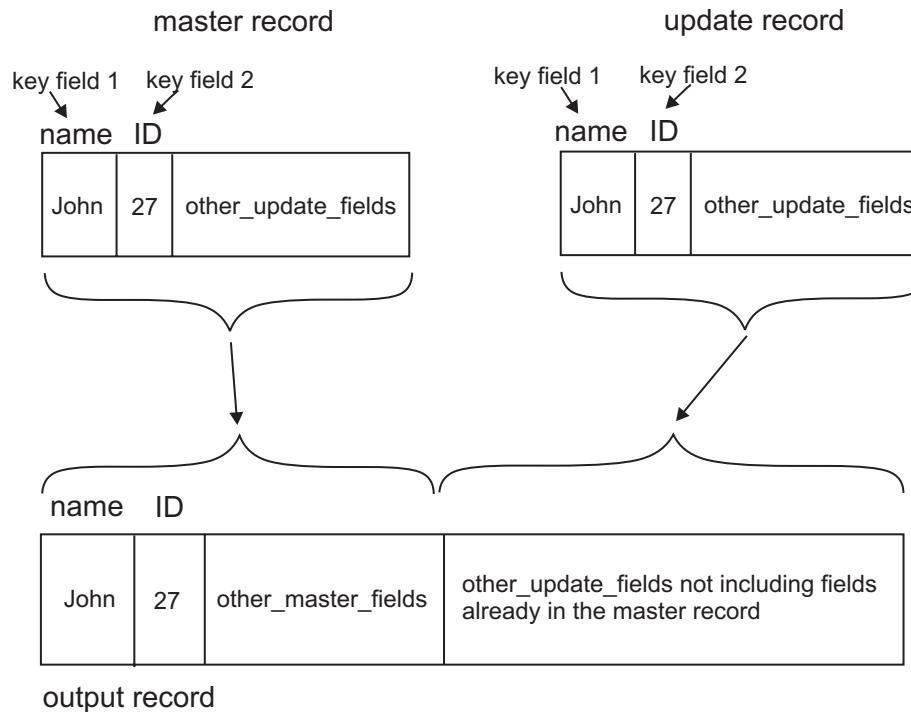
Merging records

The merge operator combines a master and one or more update data sets into a single, merged data set based upon the values of a set of merge key fields. Each record of the output data set contains all of the fields from a master record. Concatenated to the end of the output records are any fields from the corresponding update records that are not already in the master record. Corresponding master and update records have the same value for the specified merge key fields.

The action of the merge operator depends on whether you specify multiple update data sets or a single update data set. When merging a master data set with multiple update data sets, each update data set might contain only one record for each master record. When merging with a single update data set, the update data set might contain multiple records for a single master record. The following sections describe merging a master data set with a single update data set and with multiple update data sets.

Merging with a single update data set

The following diagram shows the merge of a master record and a single update record.



The figure shows the master and update records and the resultant merged record. A master record and an update record are merged only if they have the same values for the key field(s). In this example, both records have "John" as the Name and 27 as the ID value. Note that in this example the merge keys are the first fields in the record. You can use any field in the record as a merge key, regardless of its location.

The schema of the master data set determines the data types of the merge key fields. The schemas of the update data sets might be dissimilar but they must contain all merge key fields (either directly or through adapters).

The merged record contains all of the fields from the master record plus any fields from the update record which were not in the master record. Thus, if a field in the update record has the same name as a field in the master record, the data comes from the master record and the update field is ignored.

The master data set of a merge must not contain duplicate records where duplicates are based on the merge keys. That is, no two master records can have the same values for all merge keys.

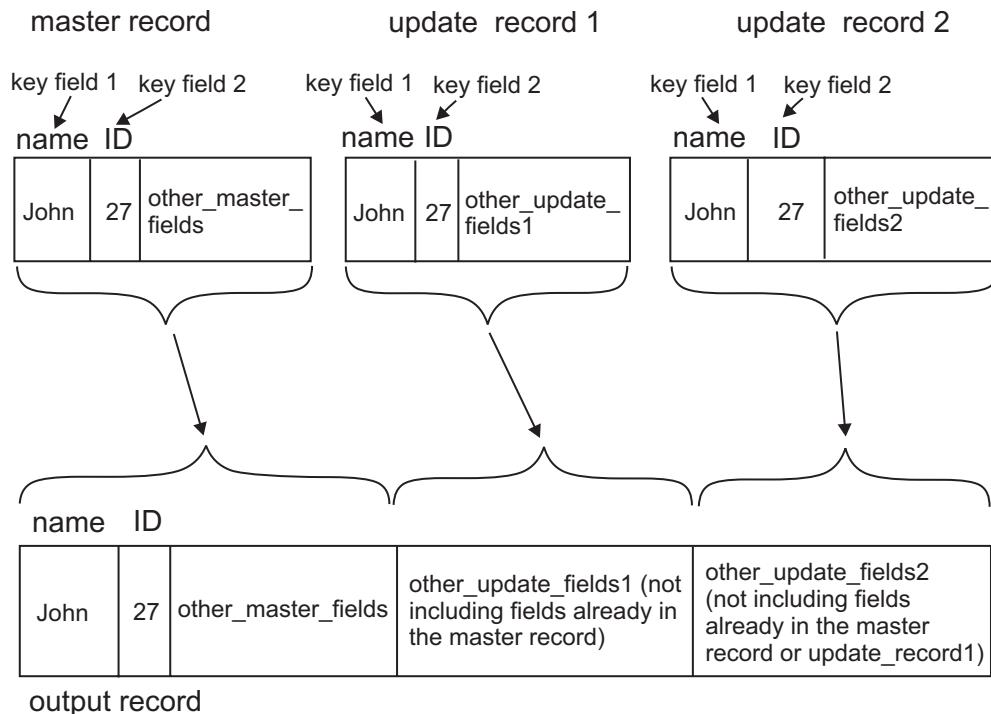
For a merge using a single update data set, you can have multiple update records, as defined by the merge keys, for the same master record. In this case, you get one output record for each master/update record pair. In the figure above, if you had two update records with "John" as the Name and 27 as the value of ID, you would get two output records.

Merging with multiple update data sets

In order to merge a master and multiple update data sets, all data sets must be sorted and contain no duplicate records where duplicates are based on the merge keys. That is, there must be at most one update record in each update data set with the same combination of merge key field values for each master record. In this case, the merge operator outputs a single record for each unique combination of merge key fields.

By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the merge operator and other operators.

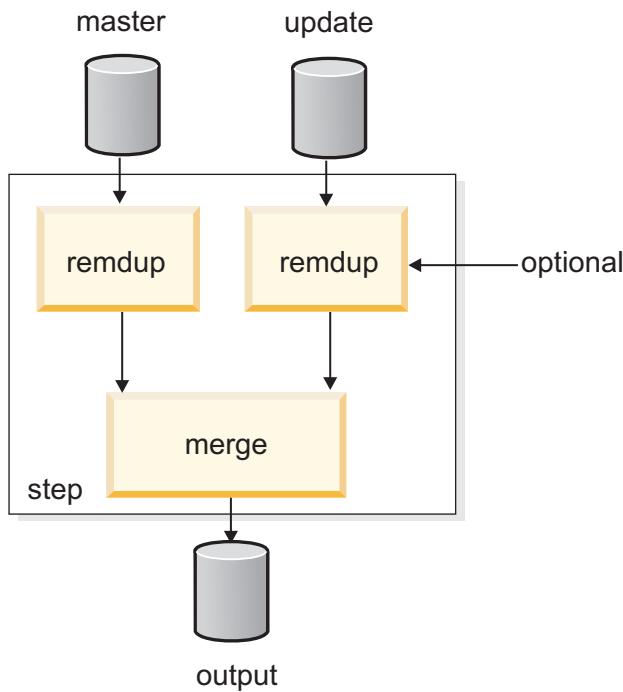
The following figure shows the merge of a master record and two update records (one update record from each of two update data sets):



Any fields in the first update record not in the master record are concatenated to the output record. Then, any fields in the second update record not in the master record or the first update record are concatenated to the output record. For each additional update data set, any fields not already in the output record are concatenated to the end of the output record.

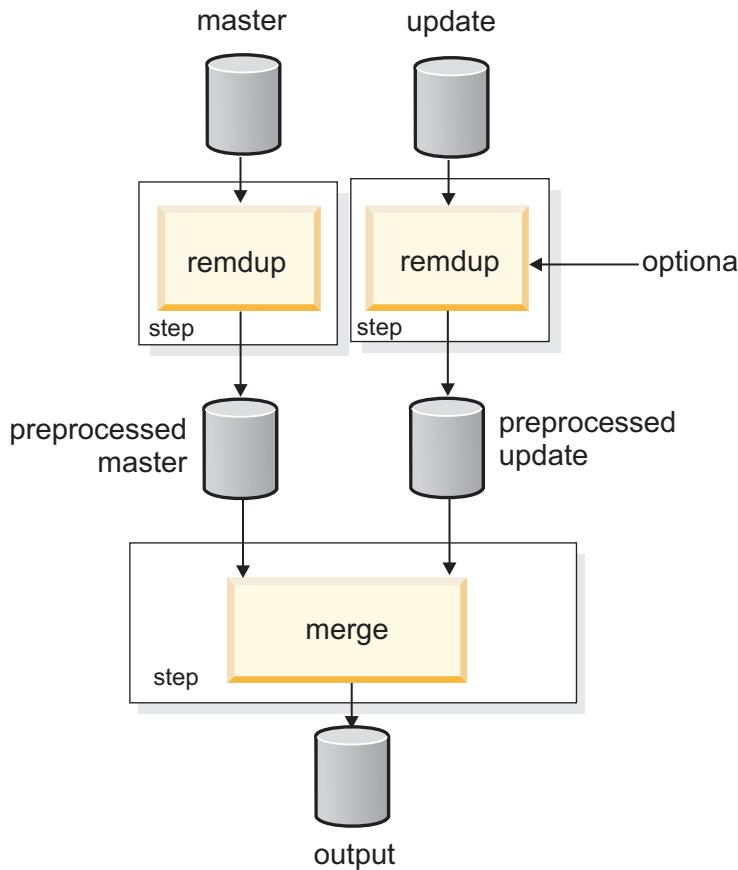
Understanding the merge operator

The following diagram shows the overall data flow for a typical use of the merge operator:



This diagram shows the overall process as one step. Note that the remdup operator is required only if you have multiple update data sets. If you have only a single update data set, the data set might contain more than one update record for each master record.

Another method is to save the output of the remdup operator to a data set and then pass that data set to the merge operator, as shown in the following figure:

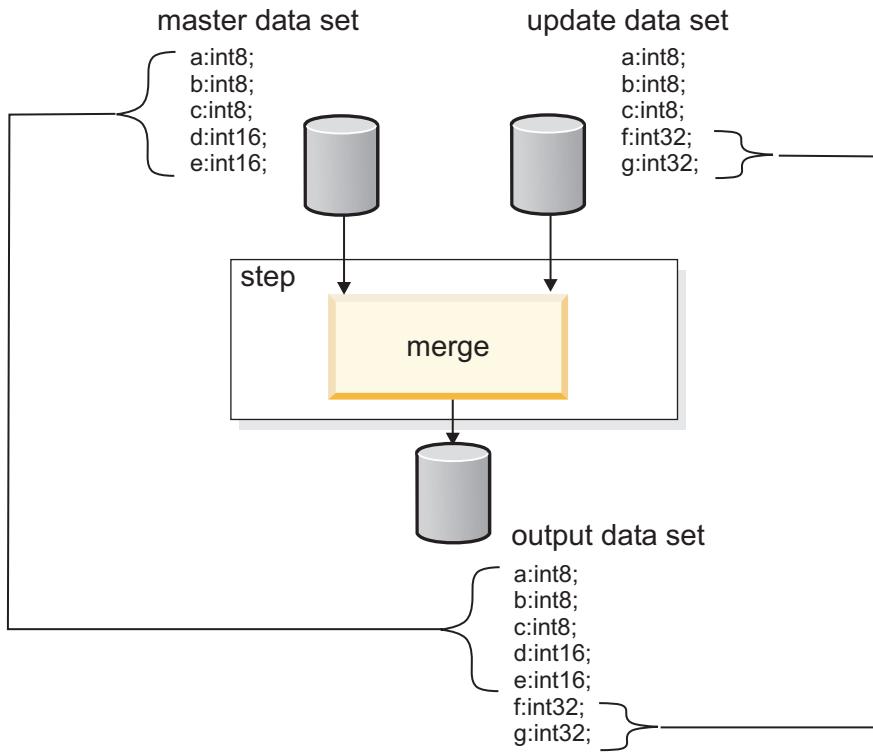


This method has the disadvantage that you need the disk space to store the pre-processed master and update data sets and the merge must be in a separate step from the remove duplicates operator. However, the intermediate files can be checked for accuracy before merging them, or used by other processing steps that require records without duplicates.

The merging operation

When data sets are merged, one is the master and all other are update data sets. The master data set is always connected to input 0 of the operator. The merged output data set always contains all of the fields from the records in the master data set. In addition, it contains any additional fields from the update data sets.

The following diagram shows the record schema of the output data set of the merged operator, based on the record schema of the master and update data sets:



This data-flow diagram shows the record schema of the master and update data sets. The record schema of the master data set has five fields and all five of these appear in the record schema of the output data set. The update data set also has five fields, but only two of these (f and g) are copied to the output data set because the remaining fields (a, b, and c) already exist in the master data set.

If the example above is extended to include a second update data set with a record schema containing the following fields:

a b d h i

Then the fields in the merged output record are now:

a b c e d f g h i

because the last two fields (h and i) occur only in the second update data set and not in the master or first update data set. The unique fields from each additional update data set are concatenated to the end of the merged output.

If there is a third update data set with a schema that contains the fields:

a b d h

it adds nothing to the merged output since none of the fields is unique. Thus if master and five update data sets are represented as:

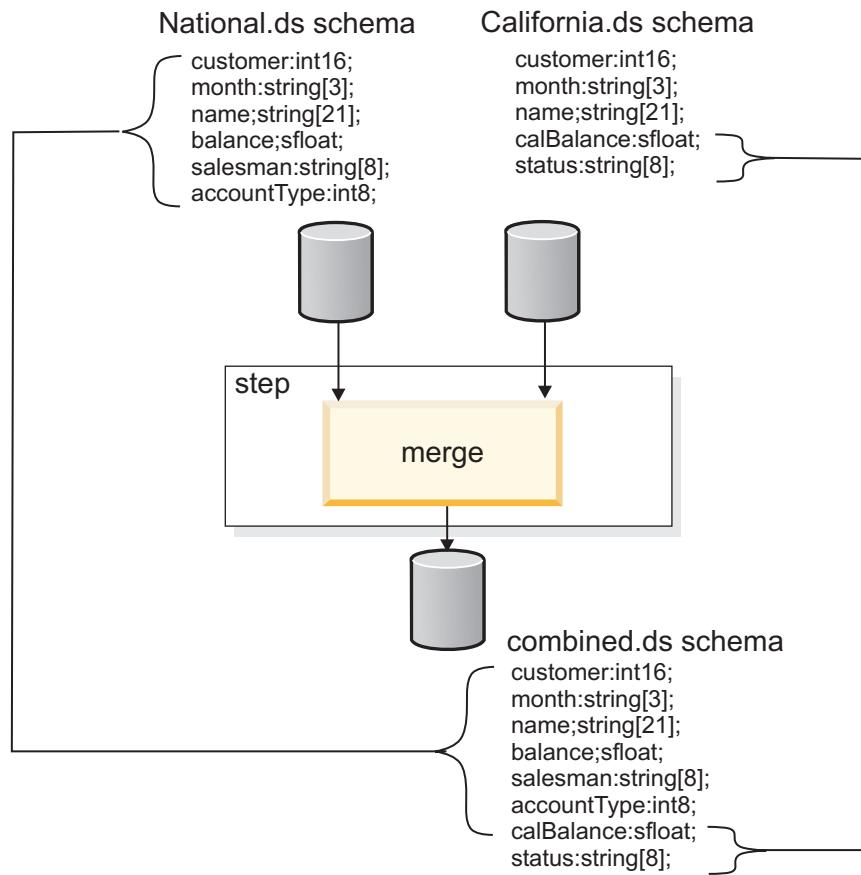
M U1 U2 U3 U4 U5

where M represents the master data set and Un represent update data set n, and if the records in all six data sets contain a field named b, the output record has a value for b taken from the master data set. If a field named e occurs in the U2, U3, and U5 update data sets, the value in the output comes from the U2 data set since it is the first one encountered.

Therefore, the record schema of the merged output record is the sequential concatenation of the master and update schema(s) with overlapping fields having values taken from the master data set or from the first update record containing the field. The values for the merge key fields are taken from the master record, but are identical values to those in the update record(s).

Example 1: updating national data with state data

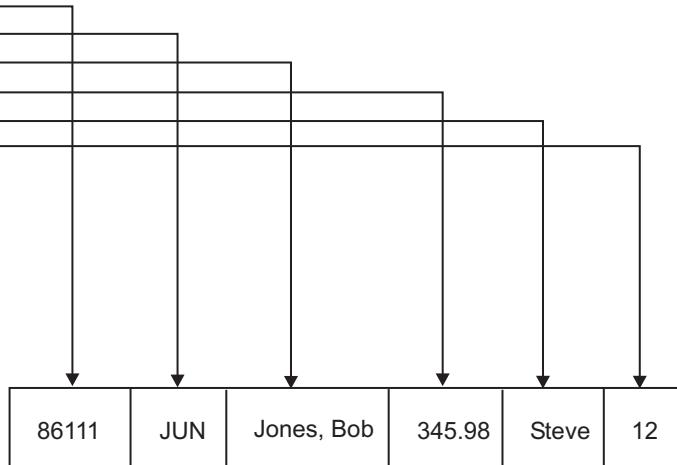
The following figure shows the schemas for a master data set named National.ds and an update data set named California.ds. The merge operation is performed to combine the two; the output is saved into a new data set named Combined.ds.



The National.ds master data set contains the following record:

National.ds schema

```
customer:int16;
month:string[3];
name:string[21];
balance:sfloat;
salesman:string[8];
accountType:int8;
```

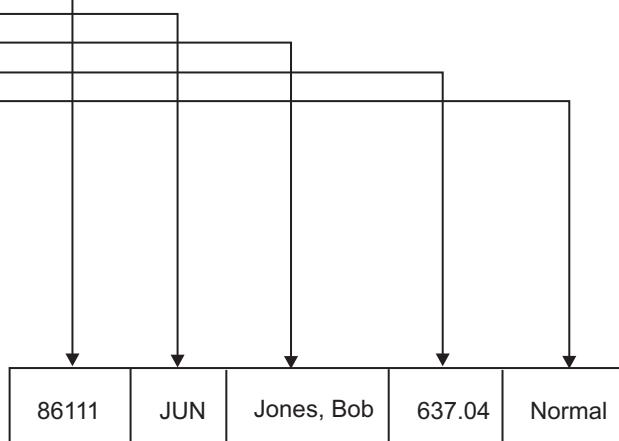


Record in National.ds data set

The Customer and Month fields are used as the merge key fields. You also have a record in the update data set named California.ds that contains the following record:

California.ds schema

```
customer:int16;
month:string[3];
name:string[21];
CalBalance:sfloat;
status:string[8];
```



Record in California.ds data set

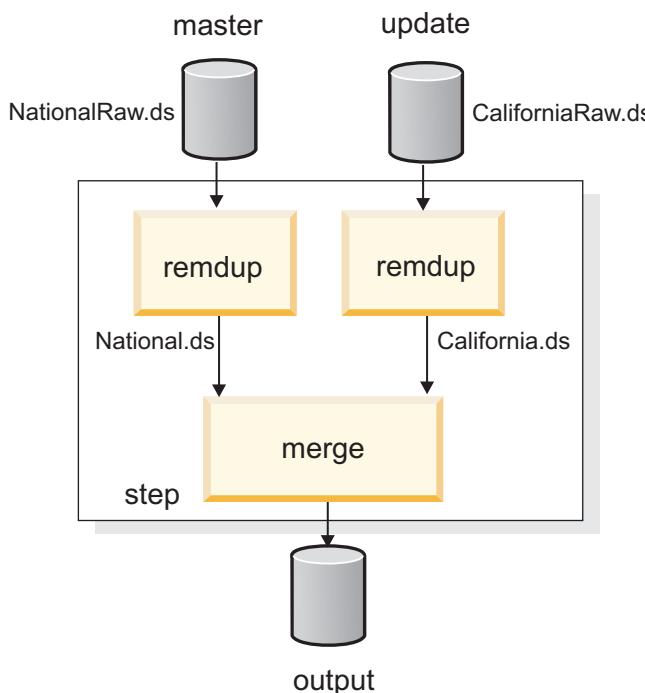
After you merge these records, the result is:

86111	JUN	Jones, Bob	Steve	12	637.04	Normal
-------	-----	------------	-------	----	--------	--------

Record in combined.ds data set

This example shows that the CalBalance and Status fields from the update record have been concatenated to the fields from the master record. The combined record has the same values for the key fields as do both the master and the update records since they must be the same for the records to be merged.

The following figure shows the data flow for this example. The original data comes from the data sets NationalRaw.ds and CaliforniaRaw.ds. National.ds and California.ds are created by first sorting and then removing duplicates from NationalRaw.ds and CaliforniaRaw.ds.



For the remdup operators and for the merge operator you specify the same key two fields:

- Option: key Value: Month
- Option: key Value: Customer

The steps for this example have been written separately so that you can check the output after each step. Because each step is separate, it is easier to understand the entire process. Later all of the steps are combined together into one step.

The separate steps, shown as osh commands, are:

```

# Produce National.ds
$ osh "remdup -key Month -key Customer < NationalRaw.ds > National.ds"
# Produce California.ds
$ osh "remdup -key Month -key Customer < CaliforniaRaw.ds > California.ds"
# Perform the merge
$ osh "merge -key Month -key Customer < National.ds < California.ds > Combined.ds"

```

This example takes NationalRaw.ds and CaliforniaRaw.ds and produces Combined.ds without creating the intermediate files.

When combining these three steps into one, you use a named virtual data sets to connect the operators.

```

$ osh "remdup -key Month -key Customer < CaliforniaRaw.ds > California.v;
remdup -key Month -key Customer < NationalRaw.ds
|
merge -key Month -key Customer < California.v > Combined.ds"

```

In this example, California.v is a named virtual data set used as input to merge.

Example 2: handling duplicate fields

If the record schema for CaliforniaRaw.ds from the previous example is changed so that it now has a field named Balance, both the master and the update data sets will have a field with the same name. By default, the Balance field from the master record is output to the merged record and the field from the update data set is ignored.

The following figure shows record schemas for the NationalRaw.ds and CaliforniaRaw.ds in which both schemas have a field named Balance:

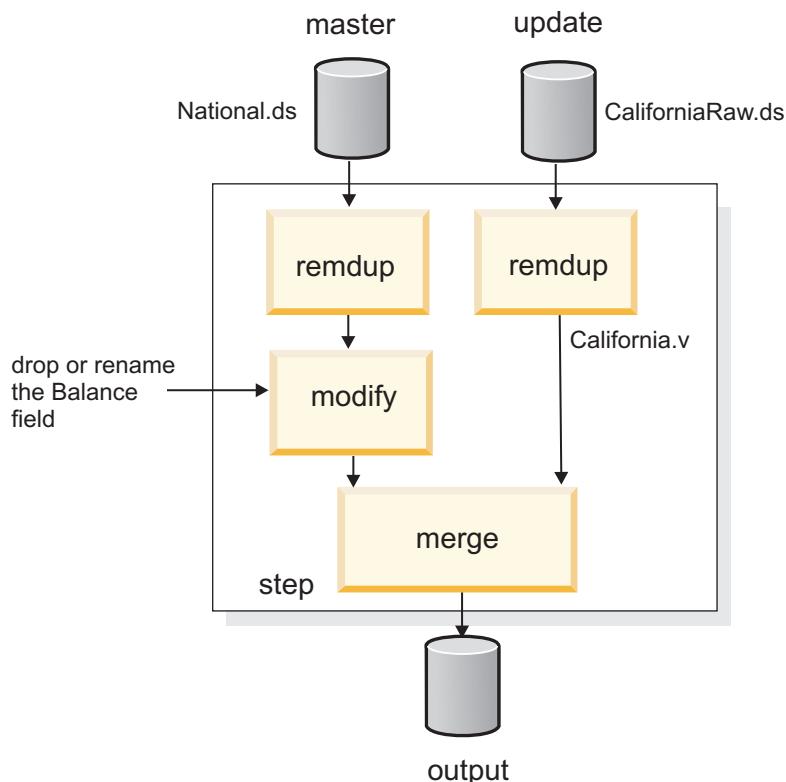
Master data set	Update data set
NationalRaw.ds	CaliforniaRaw.ds
schema:	schema:
customer:int16;	customer:int16;
month:string[3];	month:string[3];
balance:sfloat; ←	balance:sfloat;
salesman:string[8];	CalBalance:sfloat;
accountType:int8;	status:string[8];

If you want the Balance field from the update data set to be output by the merge operator, you have two alternatives, both using the modify operator.

- Rename the Balance field in the master data set.
- Drop the Balance field from the master record.

In either case, the Balance field from the update data set propagates to the output record because it is the only Balance field.

The following figure shows the data flow for both methods.



Renaming a duplicate field

The osh command for this approach is:

```
$ osh "remdup -key Month -key Customer < CaliforniaRaw.ds > California.v;
remdup -key Month -key Customer < NationalRaw.ds
```

```

modify 'OldBalance = Balance'
|
merge -key Month -key Customer < California.v > Combined.ds"

```

The name of the Balance field has been changed to OldBalance. The Balance field from the update data set no longer conflicts with a field in the master data set and is added to records by the merge.

Dropping duplicate fields

Another method of handling duplicate field names is to drop Balance from the master record. The Balance field from the update record is written out to the merged record because it is now the only field with that name.

The osh command for this approach is:

```

$ osh "remdup -key Month -key Customer < CaliforniaRaw.ds > California.v;
remdup -key Month -key Customer < NationalRaw.ds
|
modify 'DROP Balance'
|
merge -key Month -key Customer < California.v > Combined.ds"

```

Job scenario: galactic industries

This section contains an extended example that illustrates the use of the merge operator in a semi-realistic data flow. The example is followed by an explanation of why the operators were chosen. Files have been provided to allow you to run this example yourself. The files are in \$APT_ORCHHOME/examples/doc/mergeop subdirectory of the parallel engine directory.

Galactic Industries stores certain customer data in one database table and orders received for a given month in another table.

The customer table contains one entry per customer, indicating the location of the customer, how long she has been a customer, the customer contact, and other customer data. Each customer in the table is also assigned a unique identifier, cust_id. However, the customer table contains no information concerning what the customer has ordered.

The order table contains details about orders placed by customers; for each product ordered, the table lists the product name, amount, price per unit, and other product information. The order table can contain many entries for a given customer. However, the only information about customers in the order table is the customer identification field, indicated by a cust_id field which matches an entry in the customer table.

Each month Galactic Industries needs to merge the customer information with the order information to produce reports, such as how many of a given product were ordered by customers in a given region. Because the reports are reviewed by human eyes, they also need to perform a lookup operation which ties a description of each product to a product_id. Galactic Industries performs this merge and lookup operation using WebSphere DataStage.

The WebSphere DataStage solution is based on the fact that Galactic Industries has billions of customers, trillions of orders, and needs the reports fast.

The osh script for the solution follows.

```

# import the customer file; store as a virtual data set.
import
  -schema $CUSTOMER_SCHEMA
  -file customers.txt
  -readers 4 |
peek -name -nrecs 1 >customers.v;

```

```

# import the order file; store as a virtual data set.
import
  -schema $ORDER_SCHEMA
  -file orders.txt
  -readers 4 |
peek -name -nrecs 1 >orders.v;
# import the product lookup table; store as a virtual data set.
import
  -schema $LOOKUP_SCHEMA
  -file lookup_product_id.txt |
entire | # entire partitioning only necessary in MPP environments
peek -name -nrecs 1 >lookup_product_id.v;
# merge customer data with order data; lookup product descriptions;
# store as a persistent data set.
merge
  -key cust_id
  -dropBadMasters # customer did not place an order this period
  < customers.v
  < orders.v
  1>| orders_without_customers.ds | # if not empty, we have a problem
lookup
  -key product_id
  -ifNotFound continue # allow products that don't have a description
  < lookup_product_id.v |
peek -name -nrecs 10 >| customer_orders.ds;

```

Why the merge operator is used

The merge operator is not the only component in the WebSphere DataStage library capable of merging the customer and order tables. An identical merged output data set could be produced with either the lookup or innerjoin operator. Furthermore, if the -dropBadMasters behavior was not chosen, merging could also be performed using the leftouterjoin operator.

Galactic Industries' needs make the merge operator the best choice. If the lookup operator were used the customer table would be used as the lookup table. Since Galactic Industries has billions of customers and only a few Gigabytes of RAM on its SMP the data would have to spill over onto paging space, resulting in a dramatic decrease in processing speed. Because Galactic Industries is interested in identifying entries in the order table that do not have a corresponding entry in the customer table, the merge operator is a better choice than the innerjoin or leftouterjoin operator, because the merge operator allows for the capture of bad update records in a reject data set (orders_without_customers.ds in the script above).

Why the lookup operator is used

Similar functionality can be obtained from merge, lookup, or one of the join operators. For the task of appending a descriptions of product field to each record the lookup operator is most suitable in this case for the following reasons.

- Since there are around 500 different products and the length of each description is approximately 50 bytes, a lookup table consists of only about 25 Kilobytes of data. The size of the data makes the implementation of a lookup table in memory feasible, and means that the scan of the lookup table based on the key field is a relatively fast operation.
- Use of either the merge or one of the join operators would necessitate a repartition and resort of the data based on the lookup key. In other words, having partitioned and sorted the data by cust_id to accomplish the merge of customer and order tables, Galactic industries would then have to perform a second partitioning and sorting operation based on product_id in order to accomplish the lookup of the product description using either merge or innerjoin. Given the small size of the lookup table and the huge size of the merged customer/order table, the lookup operator is clearly the more efficient choice.

Why the entire operator is used

The lookup data set `lookup_product_id.v` is entire partitioned. The entire partitioner copies all records in the lookup table to all partitions ensuring that all values are available to all records for which lookup entries are sought. The entire partitioner is only required in MPP environments, due to the fact that memory is not shared between nodes of an MPP. In an SMP environment, a single copy of the lookup table is stored in memory that can be accessed by all nodes of the SMP. Using the entire partitioner in the flow makes this example portable to MPP environments, and due to the small size of the lookup table is not particularly wasteful of resources.

Missing records

The merge operator expects that for each master record there exists a corresponding update record, based on the merge key fields, and vice versa. If the merge operator takes a single update data set as input, the update data set might contain multiple update records for a single master record.

By using command-line options to the operator, you can specify the action of the operator when a master record has no corresponding update record (a bad master record) or when an update record has no corresponding master record (a bad update record).

Handling bad master records

A master record with no corresponding update record is called a bad master. When a master record is encountered which has no corresponding update record, you can specify whether the master record is to be copied to the output or dropped. You can also request that you get a warning message whenever this happens.

By default, the merge operator writes a bad master to the output data set and issues a warning message. Default values are used for fields in the output record which would normally have values taken from the update data set. You can specify `-nowarnBadMasters` to the merge operator to suppress the warning message issued for each bad master record.

Suppose the data in the master record is:

86111	JUN	Lee, Mary	345.98	Steve	12
-------	-----	-----------	--------	-------	----

Record in National.ds data set

The first field, Customer, and the second field, Month, are the key fields.

If the merge operator cannot find a record in the update data set for customer 86111 for the month of June, then the output record is:

The last two fields in the output record, OldBalance and Status, come from the update data set. Since there is no update record from which to get values for the OldBalance and Status fields, default values are written to the output record. This default value of a field is the default for that particular data type. Thus, if the value is an sfloat, the field in the output data set has a value of 0.0. For a fixed-length string, the default value for every byte is 0x00.

If you specify `-dropBadMasters`, master records with no corresponding update record are discarded (not copied to the output data set).

Handling bad update records

When an update record is encountered which has no associated master record, you can control whether the update record is dropped or is written out to a separate reject data set.

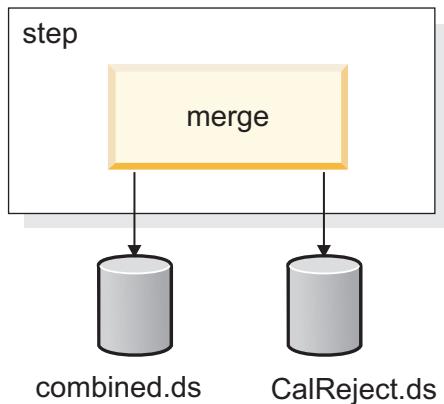
In order to collect bad update records from an update data set, you attach one output data set, called a reject data set, for each update data set. The presence of a reject data set configures the merge operator to write bad update records to the reject data set.

In the case of a merge operator taking as input multiple update data sets, you must attach a reject data set for each update data set if you want to save bad update records. You cannot selectively collect bad update records from a subset of the update data sets.

By default, the merge operator issues a warning message when it encounters a bad update record. You can use the `-nowarnBadMasters` option to the operator to suppress this warning.

For example, suppose you have a data set named `National.ds` that has one record per the key field `Customer`. You also have an update data set named `California.ds`, which also has one record per `Customer`. If you now merge these two data sets, and include a reject data set, bad update records are written to the reject data set for all customer records from `California.ds` that are not already in `National.ds`. If the reject data set is empty after the completion of the operator, it means that all of the `California.ds` customers already have `National.ds` records.

The following diagram shows an example using a reject data set.

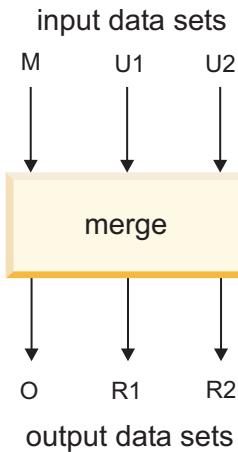


In osh, the command is:

```
$ osh "merge -key customer < National.ds < California.ds  
> Combined.ds > CalReject.ds"
```

After this step executes, `CalReject.ds` contains all records from update data set that did not have a corresponding record in the master data set.

The following diagram shows the merge operator with multiple update sets (`U1` and `U2`) and reject data sets (`R1` and `R2`). In the figure, `M` indicates the master data set and `O` indicates the merged output data set.



As you can see, you must specify a reject data set for each update data set in order to save bad update records. You must also specify the output reject data sets in the same order as you specified the input update data sets.

For example:

```
$ osh "merge -key customer
< National.ds < California.ds < NewYork.ds
> Combined.ds > CalRejects.ds > NewYorkRejects.ds"
```

Modify operator

The modify operator takes a single data set as input and alters (modifies) the record schema of the input data set to create the output data set. The modify operator changes the representation of data before or after it is processed by another operator, or both. Use it to modify:

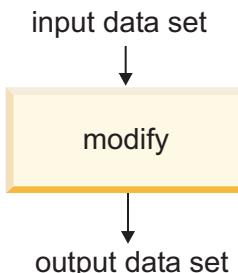
- Elements of the record schema of an input data set to the interface required by the operator to which it is input
- Elements of an operator's output to those required by the data set that receive the results

The operator performs the following modifications:

- Keeping and dropping fields
- Renaming fields
- Changing a field's data type
- Changing the null attribute of a field

The modify operator has no usage string.

Data flow diagram



modify: properties

Table 32. modify properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	no

Modify: syntax and options

Terms in *italic* typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
modify 'modify_spec1 ;modify_spec2 ; ... modify_specn ;'
```

where each *modify_spec* specifies a conversion you want to perform. "Performing Conversions" describes the conversions that the *modify* operator can perform.

- Enclose the list of modifications in single quotation marks.
- Separate the modifications with a semi-colon.
- If *modify_spec* takes more than one argument, separate the arguments with a comma and terminate the argument list with a semi-colon, as in the following example:

```
modify 'keep field1,field2, ... fieldn;'
```

Multi-byte Unicode character data is supported for fieldnames in the *modify* specifications below.

The *modify_spec* can be one of the following:

- **DROP**
- **KEEP**
- *replacement_spec*
- **NOWARN**

To drop a field:

```
DROP fieldname [, fieldname ...]
```

To keep a field:

```
KEEP fieldname [, fieldname ...]
```

To change the name or data type of a field, or both, specify a *replacement-spec*, which takes the form:

```
new-fieldname [:new-type] = [explicit-conversion-spec]  
old-fieldname
```

Replace the old field name with the new one. The default type of the new field is the same as that if the old field unless it is specified by the output type of the *conversion-spec* if provided. Multiple new fields can be instantiated based on the same old field.

When there is an attempt to put a null in a field that has not been defined as nullable, WebSphere DataStage issues an error message and terminates the job. However, a warning is issued at step-check time. To disable the warning, specify the NOWARNF option.

Transfer behavior

Fields of the input data set that are not acted on by the modify operator are transferred to the output data set unchanged.

In addition, changes made to fields are permanent. Thus:

- If you drop a field from processing by means of the modify operator, it does not appear in the output of the operation for which you have dropped it.
- If you use an upstream modify operator to change the name, type, or both of an input field, the change is permanent in the output unless you restore the field name, type, or both by invoking the modify operator downstream of the operation for whose sake the field name was changed.

In the following example, the modify operator changes field names upstream of an operation and restores them downstream of the operation, as indicated in the following table.

	Source Field Name	Destination Field Name
Upstream of Operator	aField bField cField	field1 field2 field3
Downstream of Operator	field1 field2 field3	aField bField cField

You set these fields with the command:

```
$ osh " ... | modify ' field1=aField; field2=bField; field3=cField; '  
      | op | modify ' aField=field1; bField=field2; cField=field3; '  
      ..."
```

Avoiding contiguous modify operators

Set the APT_INSERT_COPY_BEFORE MODIFY environment variable to enable the automatic insertion of a copy operator before a modify operator. This process ensures that your data flow does not have contiguous modify operators, a practice which is not supported in WebSphere DataStage.

When this variable is not set and the operator immediately preceding a modify operator in the data flow also includes a modify operator, WebSphere DataStage removes the downstream modify operator.

Performing conversions

The section "Allowed Conversions" provides a complete list of conversions you can effect using the modify operator.

This section discusses these topics:

- "Performing Conversions"
- "Keeping and Dropping Fields"
- "Renaming Fields"
- "Duplicating a Field and Giving It a New Name"
- "Changing a Field's Data Type"
- "Default Data Type Conversion"

- "Date Field Conversions"
- "Decimal Field Conversions"
- "Raw Field Length Extraction"
- "String and Ustring Field Conversions"
- "String Conversions and Lookup Tables"
- "Time Field Conversions"
- "Timestamp Field Conversions"
- "The modify Operator and Nulls"
- "The modify Operator and Partial Schemas"
- "The modify Operator and Vectors"
- "The modify Operator and Aggregate Schema Components"

Keeping and dropping fields

Invoke the modify operator to keep fields in or drop fields from the output. Here are the effects of keeping and dropping fields:

- If you choose to drop a field or fields, all fields are retained except those you explicitly drop.
- If you chose to keep a field or fields, all fields are excluded except those you explicitly keep.

In osh you specify either the keyword `keep` or the keyword `drop` to keep or drop a field, as follows:

```
modify 'keep field1, field2, ... fieldn;'  
modify 'drop field1, field2, ... fieldn;'
```

Renaming fields

To rename a field specify the attribution operator (=) , as follows:

```
modify ' newField1=oldField1; newField2=oldField2; ...newFieldn=oldFieldn; '
```

Duplicating a field and giving it a new name

You can duplicate a field and give it a new name, that is, create multiple new names for the same old name. You can also convert the data type of a field and give it a new name.

Note: This does not work with aggregates.

To duplicate and rename a field or duplicate it and change its data type use the attribution operator (=) . The operation must be performed by one modify operator, that is, the renaming and duplication must be specified in the same command as follows:

```
$ osh "modify 'a_1 = a; a_2 = a;' " $ osh "modify c_1 = conversionSpec(c); c_2 = conversionSpec(c);' "
```

where:

- a and c are the original field names; a_1, a_2 are the duplicated field names; c_1, and c_2 are the duplicated and converted field names
- *conversionSpec* is the data type conversion specification, discussed in the next section

Changing a field's data type

Sometimes, although field names are the same, an input field is of a type that differs from that of the same field in the output, and conversion must be performed. WebSphere DataStage often automatically changes the type of the source field to match that of the destination field. Sometimes, however, you must invoke the modify operator to perform explicit conversion. The next sections discuss default data type conversion and data type conversion errors. The subsequent sections discuss non-default conversions of WebSphere DataStage data types.

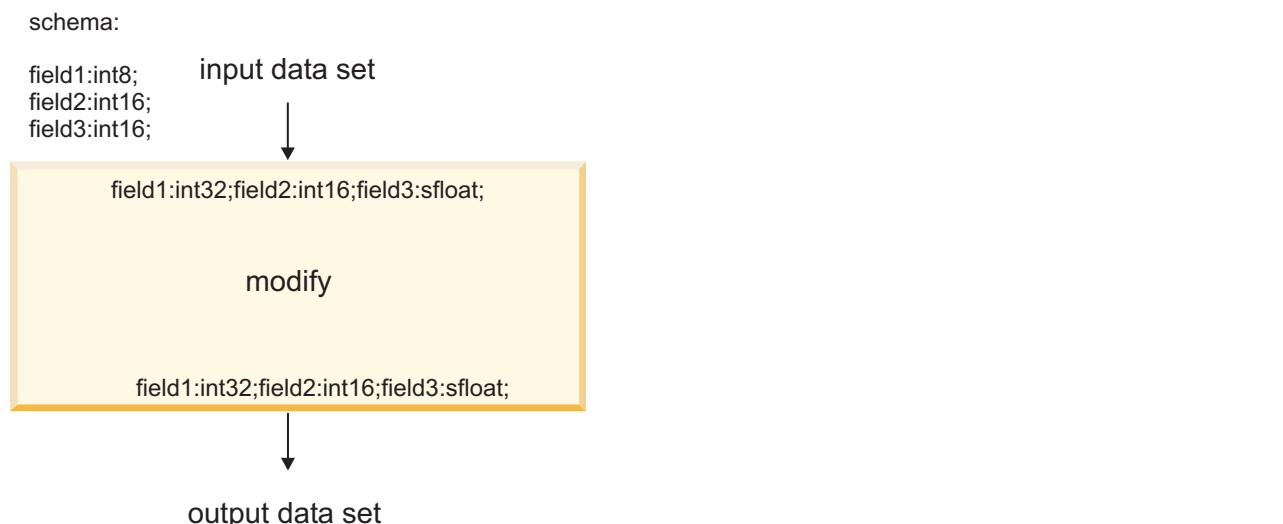
Default data type conversion

For a data set to be used as input to or output from an operator, its record schema must be compatible with that of the operator's interface.

That is:

- The names of the data set's fields must be identical to the names of the corresponding fields in the operator interface. Use the modify operator to change them if they are not (see "Renaming Fields").
- The data type of each field in the data set must be compatible with that of the corresponding field in the operator interface. Data types are compatible if WebSphere DataStage can perform a default data type conversion, translating a value in a source field to the data type of a destination field.

The following figure shows an input data set schema in which the data types of some fields do not match those of the corresponding fields and WebSphere DataStage's default conversion of these types:



The following table shows the default data conversion types.

In this example, the disparate fields are compatible and:

- The data type of field1 is automatically converted from int8 to int32.
- The data type of field3 is automatically converted from int16 to sfloat.

WebSphere DataStage performs default type conversions on WebSphere DataStage built-in numeric types (integer and floating point) as defined in C: A Reference Manual (3rd edition) by Harbison and Steele. WebSphere DataStage also performs default data conversions involving decimal, date, time, and timestamp fields. The remaining allowable data type conversions are performed explicitly, using the modify operator, as described in this topic.

The tables shows the default data type conversions performed by WebSphere DataStage and the conversions that you can perform with the modify operator.

Source Field	Destination Field									
	int8	uint8	int16	uint16	int32	uint32	int64	uint64	sfloat	dfloat
int8		d	d	d	d	d	d	d	d	d m
uint8	d		d	d	d	d	d	d	d	d

Source Field	Destination Field										
	int8	uint8	int16	uint16	int32	uint32	int64	uint64	sfloat	dfloat	
int16	d m	d		d	d	d	d	d	d	d	
uint16	d	d	d		d	d	d	d	d	d	
int32	d m	d	d	d		d	d	d	d	d	
uint32	d	d	d	d	d		d	d	d	d	
int64	d m	d	d	d	d	d		d	d	d	
uint64	d	d	d	d	d	d	d		d	d	
sfloat	d m	d	d	d	d	d	d	d		d	
dfloat	d m	d	d	d	d	d	d	d	d		
decimal	d m	d	d	d	d m	d	d m	d m	d	d m	
string	d m	d	d m	d	d	d m	d	d	d	d m	
ustring	d m	d	d m	d	d	d m	d	d	d	d m	
raw	m				m						
date	m		m		m	m					
time	m				m					m	
time stamp	m				m					m	

Source Field	Destination Field							
	decimal	string	ustring	raw	date	time	timestamp	
int8	d	d m	d m		m	m		m
uint8	d	d	d					
int16	d	d m	d m					
uint16	d	d m	d m					
int32	d	d m	d m		m			m
uint32	d	m	m		m			
int64	d	d	d					
uint64	d	d	d					
sfloat	d	d	d					
dfloat	d m	d m	d m			m		m
decimal		d m	d m					
string	d m		d		m	m		m
ustring	d m	d				m		m
raw								
date		m	m					m
time		m	m					d m
timestamp		m	m		m	m		

d = default conversion; m = modify operator conversion; blank = no conversion needed or provided

Data type conversion errors

A data type conversion error occurs when a conversion cannot be performed. WebSphere DataStage's action when it detects such an error differs according to whether the destination field has been defined as nullable, according to the following three rules:

- If the destination field has been defined as nullable, WebSphere DataStage sets it to null.
- If the destination field has not been defined as nullable but you have directed modify to convert a null to a value, WebSphere DataStage sets the destination field to the value. To convert a null to a value supply the handle_null conversion specification.
- For complete information on converting a null to a value, see "Out-of-Band to Normal Representation".
- If the destination field has not been defined as nullable, WebSphere DataStage issues an error message and terminates the job. However, a warning is issued at step-check time. To disable the warning specify the nowarn option.

How to convert a data type

To convert the data type of a field, pass the following argument to the modify operator:

```
destField[ : dataType] = [conversionSpec](sourceField);
```

where:

- *destField* is the field in the output data set
- *dataType* optionally specifies the data type of the output field. This option is allowed only when the output data set does not already have a record schema, which is typically the case.
- *sourceField* specifies the field in the input data set
- *conversionSpec* specifies the data type conversion specification; you need not specify it if a default conversion exists (see "Default Data Type Conversion"). A conversion specification can be double quoted, single quoted, or not quoted, but it cannot be a variable.

Note that once you have used a conversion specification to perform a conversion, WebSphere DataStage performs the necessary modifications to translate a conversion result to the numeric data type of the destination. For example, you can use the conversion hours_from_time to convert a time to an int8, or to an int16, int32, dfloat, and so on.

Date field conversions

WebSphere DataStage performs no automatic type conversion of date fields. Either an input data set must match the operator interface or you must effect a type conversion by means of the modify operator.

The following table lists the conversions involving the date field. For a description of the formats, refer to "date Formats".

Conversion Specification	Description
<code>dateField = date_from_days_since[date] (int32Field)</code>	date from days since Converts an integer field into a date by adding the integer to the specified base <i>date</i> . The <i>date</i> must be in the format yyyy-mm-dd.
<code>dateField = date_from_julian_day(uint32Field)</code>	date from Julian day

Conversion Specification	Description
<code>dateField = date_from_string [date_format date_uformat] (stringField)</code>	date from string or ustring Converts the <i>string</i> or <i>ustring</i> field to a date representation using the specified <i>date_format</i> .
<code>dateField = date_from_ustring [date_format date_uformat] (ustringField)</code>	By default, the string format is yyyy-mm-dd. <i>date_format</i> and <i>date_uformat</i> are described in "date Formats".
<code>dateField = date_from_timestamp(tsField)</code>	date from timestamp Converts the timestamp to a date representation.
<code>int8Field = month_day_from_date(dateField)</code>	day of month from date
<code>int8Field = weekday_from_date [originDay](dateField)</code>	day of week from date <i>originDay</i> is a string specifying the day considered to be day zero of the week. You can specify the day using either the first three characters of the day name or the full day name. If omitted, Sunday is defined as day zero. The <i>originDay</i> can be either single- or double-quoted or the quotes can be omitted.
<code>int16Field = year_day_from_date(dateField)</code>	day of year from date (returned value 1-366)
<code>int32Field = days_since_from_date[source_date] (dateField)</code>	days since date Returns a value corresponding to the number of days from <i>source_date</i> to the contents of <i>dateField</i> . <i>source_date</i> must be in the form yyyy-mm-dd and can be quoted or unquoted.
<code>uint32Field = julian_day_from_date(dateField)</code>	Julian day from date
<code>int8Field = month_from_date(dateField)</code>	month from date
<code>dateField = next_weekday_from_date[day] (dateField)</code>	next weekday from date The destination contains the date of the specified day of the week soonest after the source date (including the source date). <i>day</i> is a string specifying a day of the week. You can specify <i>day</i> by either the first three characters of the day name or the full day name. The <i>day</i> can be quoted in either single or double quotes or quotes can be omitted.
<code>dateField = previous_weekday_from_date[day] (dateField)</code>	previous weekday from date The destination contains the closest date for the specified day of the week earlier than the source date (including the source date) The <i>day</i> is a string specifying a day of the week. You can specify <i>day</i> using either the first three characters of the day name or the full day name. The <i>day</i> can be either single- or double- quoted or the quotes can be omitted.

Conversion Specification	Description
<code>stringField = string_from_date[date_format uformat](dateField)</code>	strings and ustrings from date Converts the date to a string or ustring representation using the specified <code>date_format</code> . By default, the string format is yyyy-mm-dd. <code>date_format</code> and <code>date_uformat</code> are described in "date Formats".
<code>ustringField = ustring_from_date [date_format date_uformat] (dateField)</code>	
<code>tsField = timestamp_from_date[time](dateField)</code>	timestamp from date The <code>time</code> argument optionally specifies the time to be used in building the timestamp result and must be in the form hh:nn:ss. If omitted, the <code>time</code> defaults to midnight.
<code>int16Field = year_from_date(dateField)</code>	year from date
<code>int8Field=year_week_from_date (dateField)</code>	week of year from date

A date conversion to or from a numeric field can be specified with any WebSphere DataStage numeric data type. WebSphere DataStage performs the necessary modifications and either translates a numeric field to the source data type shown above or translates a conversion result to the numeric data type of the destination. For example, you can use the conversion `month_day_from_date` to convert a date to an `int8`, or to an `int16`, `int32`, `dfloat`, and so on

date formats

Four conversions, `string_from_date`, `ustring_from_date`, `date_from_string`, and `ustring_from_date`, take as a parameter of the conversion a date format or a date uformat. These formats are described below. The default format of the date contained in the string is yyyy-mm-dd.

The format string requires that you provide enough information for WebSphere DataStage to determine a complete date (either day, month, and year, or year and day of year).

date uformat

The date uformat provides support for international components in date fields. It's syntax is:

`String%macroString%macroString%macroString`

where `%macro` is a date formatting macro such as `%mmm` for a 3-character English month. See the following table for a description of the date format macros. Only the String components of date uformat can include multi-byte Unicode characters.

date format

The format string requires that you provide enough information for WebSphere DataStage to determine a complete date (either day, month, and year, or year and day of year).

The `format_string` can contain one or a combination of the following elements:

Table 33. Date format tags

Tag	Variable width availability	Description	Value range	Options
<code>%d</code>	import	Day of month, variable width	1...31	s

Table 33. Date format tags (continued)

Tag	Variable width availability	Description	Value range	Options
%dd		Day of month, fixed width	01...31	s
%ddd	with v option	Day of year	1...366	s, v
%m	import	Month of year, variable width	1...12	s
%mm		Month of year, fixed width	01...12	s
%mmm		Month of year, short name, locale specific	Jan, Feb ...	t, u, w
%mmmm	import/export	Month of year, full name, locale specific	January, February ...	t, u, w, -N, +N
%yy		Year of century	00...99	s
%yyyy		Four digit year	0001 ...9999	
%NNNNyy		Cutoff year plus year of century	yy = 00...99	s
%e		Day of week, Sunday = day 1	1...7	
%E		Day of week, Monday = day 1	1...7	
%eee		Weekday short name, locale specific	Sun, Mon ...	t, u, w
%eeee	import/export	Weekday long name, locale specific	Sunday, Monday ...	t, u, w, -N, +N
%W	import	Week of year (ISO 8601, Mon)	1...53	s
%WW		Week of year (ISO 8601, Mon)	01...53	s

When you specify a date format string, prefix each component with the percent symbol (%) and separate the string's components with a suitable literal character.

The default date_format is %yyyy-%mm-%dd.

Where indicated the tags can represent variable-width data elements. Variable-width date elements can omit leading zeroes without causing errors.

The following options can be used in the format string where indicated in the table:

s Specify this option to allow leading spaces in date formats. The s option is specified in the form:

`%(tag,s)`

Where *tag* is the format string. For example:

`%(m,s)`

indicates a numeric month of year field in which values can contain leading spaces or zeroes and be one or two characters wide. If you specified the following date format property:

`%(d,s)/%(m,s)/%yyyy`

Then the following dates would all be valid:

8/ 8/1958

08/08/1958

8/8/1958

- v Use this option in conjunction with the %ddd tag to represent day of year in variable-width format. So the following date property:

% (ddd, v)

represents values in the range 1 to 366. (If you omit the v option then the range of values would be 001 to 366.)

- u Use this option to render uppercase text on output.

- w Use this option to render lowercase text on output.

- t Use this option to render titlecase text (initial capitals) on output.

The u, w, and t options are mutually exclusive. They affect how text is formatted for output. Input dates will still be correctly interpreted regardless of case.

-N Specify this option to left justify long day or month names so that the other elements in the date will be aligned.

+N Specify this option to right justify long day or month names so that the other elements in the date will be aligned.

Names are left justified or right justified within a fixed width field of N characters (where N is between 1 and 99). Names will be truncated if necessary. The following are examples of justification in use:

%dd-%(mmmm,-5)-%yyyyy

21-Augus-2006

%dd-%(mmmm,-10)-%yyyyy

21-August -2005

%dd-%(mmmm,+10)-%yyyyy

21- August-2005

The locale for determining the setting of the day and month names can be controlled through the locale tag. This has the format:

% (L, 'locale')

Where *locale* specifies the locale to be set using the *language_COUNTRY.variant* naming convention supported by ICU. See *NLS Guide* for a list of locales. The default locale for month names and weekday names markers is English unless overridden by a %L tag or the APT_IMPEXP_LOCALE environment variable (the tag takes precedence over the environment variable if both are set).

Use the locale tag in conjunction with your time format, for example the format string:

% (L, 'es')%eeee, %dd %mmmm %yyyy

Specifies the Spanish locale and would result in a date with the following format:

miércoles, 21 septiembre 2005

The format string is subject to the restrictions laid out in the following table. A format string can contain at most one tag from each row. In addition some rows are mutually incompatible, as indicated in the 'incompatible with' column. When some tags are used the format string requires that other tags are present too, as indicated in the 'requires' column.

Table 34. Format tag restrictions

Element	Numeric format tags	Text format tags	Requires	Incompatible with
year	%yyyy, %yy, %[nnnn]yy	-	-	-
month	%mm, %m	%mmm, %mmmm	year	week of year
day of month	%dd, %d	-	month	day of week, week of year
day of year	%ddd		year	day of month, day of week, week of year
day of week	%e, %E	%eee, %eeee	month, week of year	day of year
week of year	%WW		year	month, day of month, day of year

When a numeric variable-width input tag such as %d or %m is used, the field to the immediate right of the tag (if any) in the format string cannot be either a numeric tag, or a literal substring that starts with a digit. For example, all of the following format strings are invalid because of this restriction:

%d%m-%yyyy

%d%mm-%yyyy

%(d)%(%m)-%yyyy

%h00 hours

The *year_cutoff* is the year defining the beginning of the century in which all two-digit years fall. By default, the year cutoff is 1900; therefore, a two-digit year of 97 represents 1997.

You can specify any four-digit year as the year cutoff. All two-digit years then specify the next possible year ending in the specified two digits that is the same or greater than the cutoff. For example, if you set the year cutoff to 1930, the two-digit year 30 corresponds to 1930, and the two-digit year 29 corresponds to 2029.

On import and export, the *year_cutoff* is the base year.

This property is mutually exclusive with days_since, text, and julian.

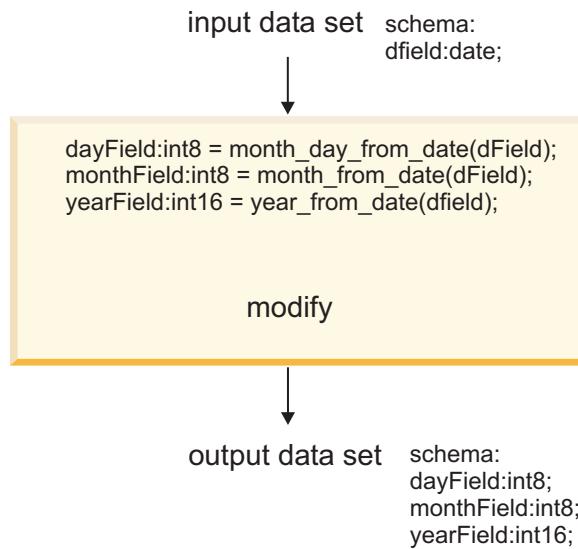
You can include literal text in your date format. Any Unicode character other than null, backslash, or the percent sign can be used (although it is better to avoid control codes and other non-graphic characters). The following table lists special tags and escape sequences:

Tag	Escape sequence
%%	literal percent sign
\%	literal percent sign
\n	newline
\t	horizontal tab
\\"	single backslash

For example, the format string %mm/%dd/%yyyy specifies that slashes separate the string's date components; the format %ddd-%yy specifies that the string stores the date as a value from 1 to 366, derives the year from the current year cutoff of 1900, and separates the two components with a dash (-).

The diagram shows the modification of a date field to three integers. The modify operator takes:

- The day of the month portion of a date field and writes it to an 8-bit integer
- The month portion of a date field and writes it to an 8-bit integer
- The year portion of a date field and writes it to a 16-bit integer



Use the following osh command:

```
$ osh "... | modify 'dayField = month_day_from_date(dField);  
monthField = month_from_date(dField);  
yearField = year_from_date(dField);' | ..."
```

Decimal field conversions

By default WebSphere DataStage converts decimal fields to and from all numeric data types and to and from string fields. The default rounding method of these conversion is truncate toward zero. However, the modify operator can specify a different rounding method. See “rounding type” on page 183.

The operator can specify fix_zero so that a source decimal containing all zeros (by default illegal) is treated as a valid decimal with a value of zero.

WebSphere DataStage does not perform range or representation checks of the fields when a source and destination decimal have the same precision and scale. However, you can specify the decimal_from_decimal conversion to force WebSphere DataStage to perform an explicit range and representation check. This conversion is useful when one decimal supports a representation of zeros in all its digits (normally illegal) and the other does not.

The following table lists the conversions involving decimal fields:

Conversion	Conversion Specification
decimal from decimal	decimalField = decimal_from_decimal[r_type](decimalField)
decimal from dfloat	decimalField = decimal_from_dfloat[r_type](dfloatField)

Conversion	Conversion Specification
decimal from string	<i>decimalField</i> = decimal_from_string[r_type](<i>stringField</i>)
decimal from ustring	<i>decimalField</i> = decimal_from_ustring[r_type](<i>ustringField</i>)
scaled decimal from int64	<i>target_field:decimal[p,s]</i> = scaled_decimal_from_int64 [no_warn] (<i>int64field</i>)
dfloat from decimal	<i>dfloatField</i> = dfloat_from_decimal[fix_zero] (<i>decimalField</i>)
dfloat from decimal	<i>dfloatField</i> = mantissa_from_decimal(<i>decimalField</i>)
dfloat from dfloat	<i>dfloatField</i> = mantissa_from_dfloat(<i>dfloatField</i>)
int32 from decimal	<i>int32Field</i> = int32_from_decimal[r_type, fix_zero] (<i>decimalField</i>)
int64 from decimal	<i>int64Field</i> = int64_from_decimal[r_type, fix_zero] (<i>decimalField</i>)
string from decimal	<i>stringField</i> = string_from_decimal [fix_zero] [suppress_zero] (<i>decimalField</i>)
ustring from decimal	<i>ustringField</i> = ustring_from_decimal [fix_zero] [suppress_zero] (<i>decimalField</i>)
uint64 from decimal	<i>uint64Field</i> = uint64_from_decimal[r_type, fix_zero] (<i>decimalField</i>)

A decimal conversion to or from a numeric field can be specified with any WebSphere DataStage numeric data type. WebSphere DataStage performs the necessary modification. For example, int32_from_decimal converts a decimal either to an int32 or to any numeric data type, such as int16, or uint32.

The scaled decimal from int64 conversion takes an integer field and converts the field to a decimal of the specified precision (*p*) and scale (*s*) by dividing the field by 10^2 . For example, the conversion:

```
Decfield:decimal[8,2]=scaled_decimal_from_int64(intfield)
```

where intfield = 12345678 would set the value of Decfield to 123456.78.

The fix_zero specification causes a decimal field containing all zeros (normally illegal) to be treated as a valid zero. Omitting fix_zero causes WebSphere DataStage to issue a conversion error when it encounters a decimal field containing all zeros. "Data Type Conversion Errors" discusses conversion errors.

The suppress_zero argument specifies that the returned string value will have no leading or trailing zeros. Examples:

000.100 -> 0.1; 001.000 -> 1; -001.100 -> -1.1

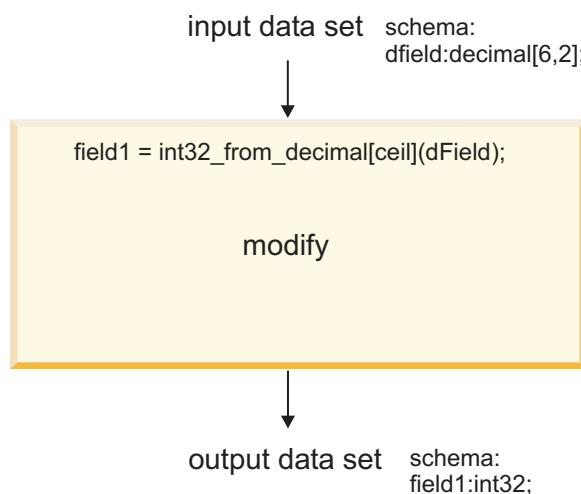
rounding type

You can optionally specify a value for the rounding type (*r_type*) of many conversions. The values of *r_type* are:

- ceil: Round the source field toward positive infinity. This mode corresponds to the IEEE 754 Round Up mode.
- Examples: 1.4 -> 2, -1.6 -> -1
- floor: Round the source field toward negative infinity. This mode corresponds to the IEEE 754 Round Down mode.
- Examples: 1.6 -> 1, -1.4 -> -2

- round_inf: Round or truncate the source field toward the nearest representable value, breaking ties by rounding positive values toward positive infinity and negative values toward negative infinity. This mode corresponds to the COBOL ROUNDED mode.
- Examples: 1.4 -> 1, 1.5 -> 2, -1.4 -> -1, -1.5 -> -2
- trunc_zero (default): Discard any fractional digits to the right of the right-most fractional digit supported in the destination, regardless of sign. For example, if the destination is an integer, all fractional digits are truncated. If the destination is another decimal with a smaller scale, round or truncate to the scale size of the destination decimal. This mode corresponds to the COBOL INTEGER-PART function.
- Examples: 1.6 -> 1, -1.6 -> -1

The diagram shows the conversion of a decimal field to a 32-bit integer with a rounding mode of ceil rather than the default mode of truncate to zero:



The osh syntax for this conversion is:

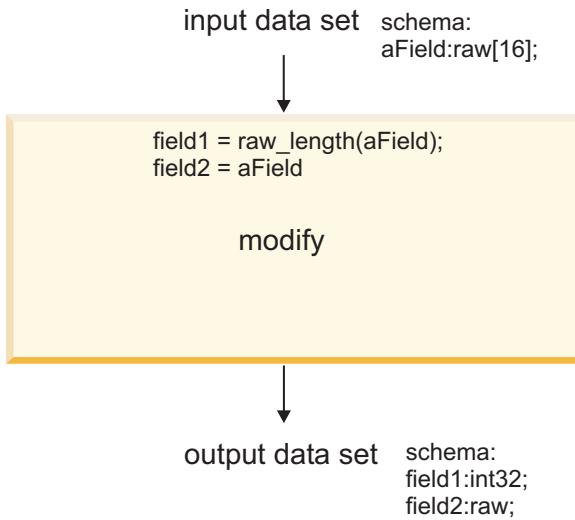
```
'field1 = int32_from_decimal[ceil,fix_zero] (dField);'
```

where fix_zero ensures that a source decimal containing all zeros is treated as a valid representation.

Raw field length extraction

Invoke the modify operator and the raw_length option to extract the length of a raw field. This specification returns an int32 containing the length of the raw field and optionally passes through the source field.

The diagram shows how to find the length of aField using the modify operator and the raw_length option:



Use the following osh commands to specify the raw_length conversion of a field:

```
$ modifySpec="field1 = raw_length(aField); field2 = aField;"  
$ osh " ... | modify '$modifySpec' |..."
```

Notice that a shell variable (modifySpec) has been defined containing the specifications passed to the operator.

Conversion Specification	Description
<i>rawField</i> = raw_from_string(<i>string</i>)	Returns <i>string</i> in raw representation.
<i>rawField</i> = u_raw_from_string(<i>ustring</i>)	Returns <i>ustring</i> in raw representation.
<i>int32Field</i> = raw_length(<i>raw</i>)	Returns the length of the <i>raw</i> field.

String and ustring field conversions

Use the modify operator to perform the following modifications involving string and ustring fields:

- Extract the length of a string.
- Convert long strings to shorter strings by string extraction.
- Convert strings to and from numeric values using lookup tables (see "String Conversions and Lookup Tables").

Conversion Specification	Description
<code>stringField=string_trim [character, direction, justify] (string)</code>	<p>You can use this function to remove the characters used to pad variable-length strings when they are converted to fixed-length strings of greater length. By default, these characters are retained when the fixed-length string is then converted back to a variable-length string.</p> <p>The <i>character</i> argument is the character to remove. It defaults to NULL. The value of the <i>direction</i> and <i>justify</i> arguments can be either begin or end; <i>direction</i> defaults to end, and <i>justify</i> defaults to begin. <i>justify</i> has no affect when the target string has variable length.</p> <p>Examples:</p> <pre>name:string = string_trim[NULL, begin](name)</pre> <p>removes all leading ASCII NULL characters from the beginning of name and places the remaining characters in an output variable-length string with the same name.</p> <pre>hue:string[10] = string_trim['Z', end, begin](color)</pre> <p>removes all trailing Z characters from color, and left justifies the resulting hue fixed-length string.</p>
<code>stringField=substring(string, starting_position, length)</code> <code>cstringField=u_substring(cstring, starting_position, length)</code>	<p>Copies parts of <i>strings</i> and <i>ustrings</i> to shorter strings by string extraction. The <i>starting_position</i> specifies the starting location of the substring; <i>length</i> specifies the substring length.</p> <p>The arguments <i>starting_position</i> and <i>length</i> are uint16 types and must be positive (≥ 0).</p>
<code>stringField=lookup_string_from_int16 [tableDefinition](int16Field)</code> <code>cstringField=lookup_cstring_from_int16 [tableDefinition](int16Field)</code>	<p>Converts numeric values to strings and ustrings by means of a lookup table.</p>
<code>int16Field=lookup_int16_from_string [tableDefinition](stringField)</code> <code>int16Field=lookup_int16_from_cstring [tableDefinition](cstringField)</code>	<p>Converts strings and ustrings to numeric values by means of a lookup table.</p>
<code>uint32 = lookup_uint32_from_string [tableDefinition](stringField)</code> <code>uint32 = lookup_uint32_from_cstring [tableDefinition](cstringField)</code>	
<code>stringField= lookup_string_from_uint32 [tableDefinition](uint32Field)</code> <code>cstringField=lookup_cstring_from_uint32 [tableDefinition](uint32Field)</code>	<p>Converts numeric values to strings and ustrings by means of a lookup table.</p>
<code>stringField = string_from_ushort(ushort)</code>	<p>Converts ustrings to strings.</p>
<code>cstringField = cstring_from_string(string)</code>	<p>Converts strings to ustrings.</p>
<code>decimalField = decimal_from_string(stringField)</code>	<p>Converts strings to decimals.</p>
<code>decimalField = decimal_from_ushort(ushortField)</code>	<p>Converts ustrings to decimals.</p>

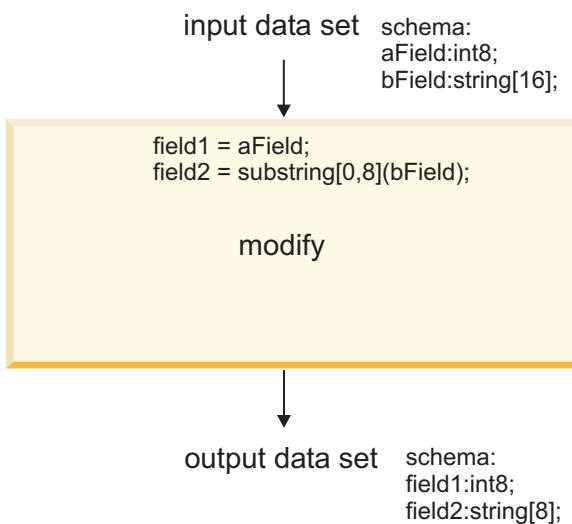
Conversion Specification	Description
<code>stringField = string_from_decimal[fix_zero] [suppress_zero] (decimalField)</code>	<p>Converts decimals to strings.</p> <p><code>fix_zero</code> causes a decimal field containing all zeros to be treated as a valid zero.</p> <p><code>suppress_zero</code> specifies that the returned <code>wstring</code> value will have no leading or trailing zeros. Examples: <code>000.100 -> 0.1; 001.000 -> 1; -001.100 -> -1.1</code></p>
<code>wstringField = wstring_from_decimal[fix_zero] [suppress_zero] (decimalField)</code>	<p>Converts decimals to <code>wstrings</code>.</p> <p>See <code>string_from_decimal</code> above for a description of the <code>fix_zero</code> and <code>suppress_zero</code> arguments.</p>
<code>dateField = date_from_string [date_format date_uformat] (stringField)</code> <code>dateField = date_from_wstring [date_format date_uformat] (wstringField)</code>	<p>date from string or <code>wstring</code></p> <p>Converts the string or <code>wstring</code> field to a date representation using the specified <code>date_format</code> or <code>date_uformat</code>.</p> <p>By default, the string format is <code>yyyy-mm-dd</code>. <code>date_format</code> and <code>date_uformat</code> are described in "date Formats".</p>
<code>stringField = string_from_date [date_format date_uformat] (dateField)</code> <code>wstringField = wstring_from_date [date_format date_uformat] (dateField)</code>	<p>strings and <code>wstrings</code> from date</p> <p>Converts the date to a string or <code>wstring</code> representation using the specified <code>date_format</code> or <code>date_uformat</code>.</p> <p>By default, the string format is <code>yyyy-mm-dd</code>. <code>date_format</code> and <code>date_uformat</code> are described in "date Formats".</p>
<code>int32Field=string_length(stringField)</code> <code>int32Field=wstring_length(wstringField)</code>	<p>Returns an <code>int32</code> containing the length of a string or <code>wstring</code>.</p>
<code>stringField=substring [startPosition,len] (stringField)</code> <code>wstringField=substring [startPosition,len] (wstringField)</code>	<p>Converts long strings/<code>wstrings</code> to shorter strings/<code>wstrings</code> by string extraction. The <code>startPosition</code> specifies the starting location of the substring; <code>len</code> specifies the substring length.</p> <p>If <code>startPosition</code> is positive, it specifies the byte offset into the string from the beginning of the string. If <code>startPosition</code> is negative, it specifies the byte offset from the end of the string.</p>
<code>stringField=uppercase_string (stringField)</code> <code>wstringField=uppercase_wstring (wstringField)</code>	<p>Convert strings and <code>wstrings</code> to all uppercase.</p> <p>Non-alphabetic characters are ignored in the conversion.</p>
<code>stringField=lowercase_string (stringField)</code> <code>wstringField=lowercase_wstring (wstringField)</code>	<p>Convert strings and <code>wstrings</code> to all lowercase.</p> <p>Non-alphabetic characters are ignored in the conversion.</p>
<code>stringField = string_from_time [time_format time_uformat]] (timeField)</code> <code>wstringField = wstring_from_time [time_format time_uformat] (timeField)</code>	<p>string and <code>wstring</code> from time</p> <p>Converts the time to a string or <code>wstring</code> representation using the specified <code>time_format</code> or <code>time_uformat</code>. The <code>time_format</code> options are described below.</p>

Conversion Specification	Description
<pre>stringField = string_from_timestamp [timestamp_format timestamp_ufORMAT] (tsField)</pre> <pre>ustringField = ustring_from_timestamp [timestamp_format timestamp_ufORMAT] (tsField)</pre>	strings and ustrings from timestamp Converts the timestamp to a string or ustring representation using the specified <i>timestamp_format</i> or <i>timestamp_ufORMAT</i> . By default, the string format is %yyyy-%mm-%dd hh:mm:ss. The <i>timestamp_format</i> and <i>timestamp_ufORMAT</i> options are described in "timestamp Formats".
<pre>tsField = timestamp_from_string [timestamp_format timestamp_ufORMAT] (stringField)</pre> <pre>tsField = timestamp_from_ustring [timestamp_format timestamp_ufORMAT] (ustringField)</pre>	timestamp from strings and ustrings Converts the string or ustring to a timestamp representation using the specified <i>timestamp_format</i> or <i>timestamp_ufORMAT</i> . By default, the string format is yyyy-mm-dd hh:mm:ss. The <i>timestamp_format</i> and <i>timestamp_ufORMAT</i> options are described "timestamp Formats".
<pre>timeField = time_from_string [time_format time_ufORMAT](stringField)</pre> <pre>timeField = time_from_ustring [time_format time_ufORMAT] (ustringField)</pre>	string and ustring from time Converts the time to a string or ustring representation using the specified <i>time_format</i> . The <i>time_ufORMAT</i> options are described below.

The following osh command converts a string field to lowercase:

```
osh "... | modify \"lname=lowercase_string(lname)\" | peek"
```

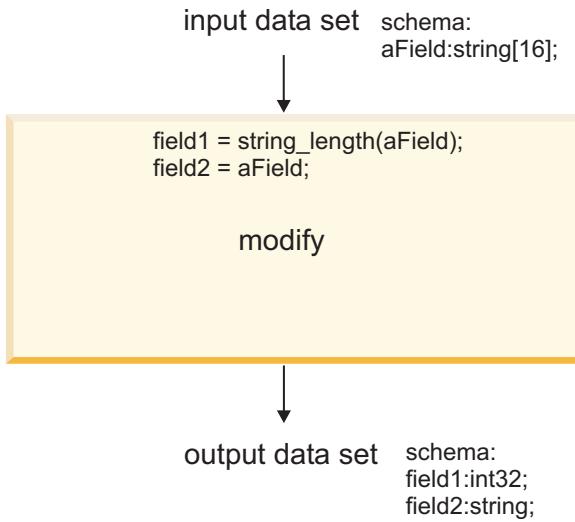
The diagram shows a modification that converts the name of aField to field1 and produces field2 from bField by extracting the first eight bytes of bField:



The following osh command performs the substring extraction:

```
modify 'field1 = aField; field2 = substring[0,8](bField);'
```

The diagram shows the extraction of the string_length of aField. The length is included in the output as field1.



The following osh commands extract the length of the string in aField and place it in field1 of the output:

```
$ modifySpec="field1 = string_length(aField); field2 = aField;"  
$ osh " ... | modify '$modifySpec' |..."
```

Notice that a shell variable (modifySpec) has been defined containing the specifications passed to the operator.

String conversions and lookup tables

You can construct a string lookup table to use when default conversions do not yield satisfactory results. A string lookup table is a table of two columns and as many rows as are required to perform a conversion to or from a string as shown in the following table:

Numeric Value	String or Ustring
numVal1	string1 ustring1
numVal2	string2 ustring1
...	...
numVal3	stringn ustringn

Each row of the lookup table specifies an association between a 16-bit integer or unsigned 32-bit integer value and a string or ustring. WebSphere DataStage scans the Numeric Value or the String or Ustring column until it encounters the value or string to be translated. The output is the corresponding entry in the row.

The numeric value to be converted might be of the int16 or the uint32 data type. WebSphere DataStage converts strings to values of the int16 or uint32 data type using the same table.

If the input contains a numeric value or string that is not listed in the table, WebSphere DataStage operates as follows:

- If a numeric value is unknown, an empty string is returned by default. However, you can set a default string value to be returned by the string lookup table.
- If a string has no corresponding value, 0 is returned by default. However, you can set a default numeric value to be returned by the string lookup table.

Here are the options and arguments passed to the modify operator to create a lookup table:

```

intField = lookup_int16_from_string[tableDefinition]
    (source_stringField); |
intField = lookup_int16_from_ushort[tableDefinition]
    (source_ushortField);

```

OR:

```

intField = lookup_uint32_from_string[tableDefinition]
    (source_stringField); |
intField = lookup_uint32_from_ushort[tableDefinition]
    (source_ushortField);
stringField = lookup_string_from_int16[tableDefinition](source_intField); |
ushortField = lookup_ushort_from_int16[tableDefinition](source_intField);

```

OR:

```

stringField = lookup_string_from_uint32[tableDefinition]
    (source_intField);
ushortField = lookup_ushort_from_uint32[tableDefinition]
    (source_intField);

```

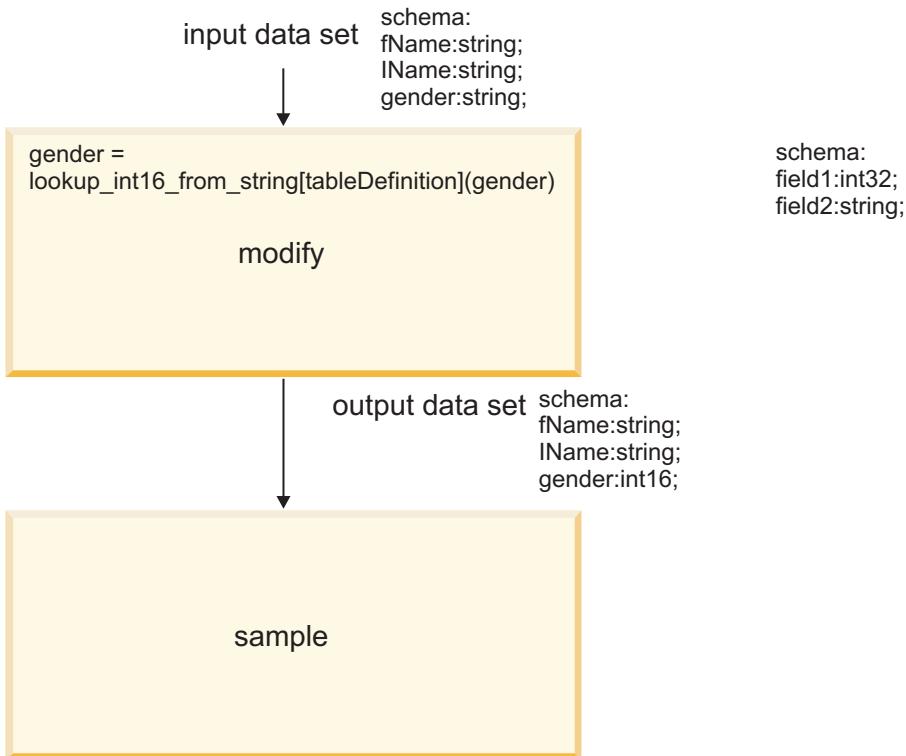
where:

tableDefinition defines the rows of a string or ushort lookup table and has the following form:
`{propertyList} ('string' | 'ushort' = value; 'string' | 'ushort'= value; ...)`

where:

- *propertyList* is one or more of the following options; the entire list is enclosed in braces and properties are separated by commas if there are more than one:
 - *case_sensitive*: perform a case-sensitive search for matching strings; the default is case-insensitive.
 - *default_value* = *defVal*: the default numeric value returned for a string that does not match any of the strings in the table.
 - *default_string* = *defString*: the default string returned for numeric values that do not match any numeric value in the table.
- *string* or *ushort* specifies a comma-separated list of strings or ushorts associated with *value*; enclose each string or ushort in quotes.
- *value* specifies a comma-separated list of 16-bit integer values associated with *string* or *ushort*.

The diagram shows an operator and data set requiring type conversion:



Whereas gender is defined as a string in the input data set, the SampleOperator defines the field as an 8:-bit integer. The default conversion operation cannot work in this case, because by default WebSphere DataStage converts a string to a numeric representation and gender does not contain the character representation of a number. Instead the gender field contains the string values "male", "female", "m", or "f". You must therefore specify a string lookup table to perform the modification.

The gender lookup table required by the example shown above is shown in the following table:

Numeric Value	String
0	"f"
0	"female"
1	"m"
1	"male"

The value f or female is translated to a numeric value of 0; the value m or male is translated to a numeric value of 1.

The following osh code performs the conversion:

```
modify 'gender = lookup_int16_from_string[{default_value = 2}  
('f' = 0; 'female' = 0; 'm' = 1; 'male' = 1;)] (gender);'
```

In this example, gender is the name of both the source and the destination fields of the translation. In addition, the string lookup table defines a default value of 2; if gender contains a string that is not one of "f", "female", "m", or "male", the lookup table returns a value of 2.

Time field conversions

WebSphere DataStage performs no automatic conversions to or from the time data type. You must invoke the modify operator if you want to convert a source or destination time field. Most time field conversions extract a portion of the time, such as hours or minutes, and write it into a destination field.

Conversion Specification	Description
<code>int8Field = hours_from_time(timeField)</code>	hours from time
<code>int32Field = microseconds_from_time(timeField)</code>	microseconds from time
<code>int8Field = minutes_from_time(timeField)</code>	minutes from time
<code>dfloatField = seconds_from_time(timeField)</code>	seconds from time
<code>dfloatField = midnight_seconds_from_time (timeField)</code>	seconds-from-midnight from time
<code>stringField = string_from_time [time_format time_uformat] (timeField)</code>	string and ustring from time Converts the time to a string or ustring representation using the specified <code>time_format</code> or <code>time_uformat</code> . The time formats are described below.
<code>ustringField = ustring_from_time [time_format time_uformat] (timeField)</code>	Converts the time to a string or ustring representation using the specified <code>time_format</code> or <code>time_uformat</code> . The time formats are described below.
<code>timeField = time_from_midnight_seconds (dfloatField)</code>	time from seconds-from-midnight
<code>timeField = time_from_string [time_format time_uformat] (stringField)</code>	time from string Converts the string or ustring to a time representation using the specified <code>time_format</code> or <code>time_uformat</code> .
<code>timeField = time_from_ustring [time_format time_uformat] (ustringField)</code>	The time format options are described below.
<code>timeField = time_from_timestamp(tsField)</code>	time from timestamp
<code>tsField = timestamp_from_time [date](timeField)</code>	timestamp from time The <code>date</code> argument is required. It specifies the date portion of the timestamp and must be in the form <code>yyyy-mm-dd</code> .

Time conversion to a numeric field can be used with any WebSphere DataStage numeric data type. WebSphere DataStage performs the necessary modifications to translate a conversion result to the numeric data type of the destination. For example, you can use the conversion `hours_from_time` to convert a time to an int8, or to an int16, int32, dfloat, and so on.

time Formats

Four conversions, `string_from_time`, `ustring_from_time`, `time_from_string`, and `ustring_from_time`, take as a parameter of the conversion a time format or a time uformat. These formats are described below. The default format of the time contained in the string is hh:mm:ss.

time Uformat

The time uformat date format provides support for international components in time fields. Its syntax is:

`String % macroString % macroString % macroString`

where `%macro` is a time formatting macro such as `%hh` for a two-digit hour. See “time Format” on page 193 below for a description of the time format macros. Only the String components of time uformat can include multi-byte Unicode characters.

time Format

The `string_from_time` and `time_from_string` conversions take a format as a parameter of the conversion. The default format of the time in the string is `hh:mm:ss`. However, you can specify an optional format string defining the time format of the string field. The format string must contain a specification for hours, minutes, and seconds.

The possible components of the `time_format` string are given in the following table:

Table 35. Time format tags

Tag	Variable width availability	Description	Value range	Options
<code>%h</code>	import	Hour (24), variable width	0...23	s
<code>%hh</code>		Hour (24), fixed width	0...23	s
<code>%H</code>	import	Hour (12), variable width	1...12	s
<code>%HH</code>		Hour (12), fixed width	01...12	s
<code>%n</code>	import	Minutes, variable width	0...59	s
<code>%nn</code>		Minutes, fixed width	0...59	s
<code>%s</code>	import	Seconds, variable width	0...59	s
<code>%ss</code>		Seconds, fixed width	0...59	s
<code>%s.N</code>	import	Seconds + fraction ($N = 0...6$)	-	s, c, C
<code>%ss.N</code>		Seconds + fraction ($N = 0...6$)	-	s, c, C
<code>%SSS</code>	with v option	Milliseconds	0...999	s, v
<code>%SSSSSS</code>	with v option	Microseconds	0...999999	s, v
<code>%aa</code>	German	am/pm marker, locale specific	am, pm	u, w

By default, the format of the time contained in the string is `%hh:%nn:%ss`. However, you can specify a format string defining the format of the string field.

You must prefix each component of the format string with the percent symbol. Separate the string's components with any character except the percent sign (%).

Where indicated the tags can represent variable-fields on import, export, or both. Variable-width date elements can omit leading zeroes without causing errors.

The following options can be used in the format string where indicated:

`s` Specify this option to allow leading spaces in time formats. The s option is specified in the form:
`%(tag,s)`

Where `tag` is the format string. For example:

`%(n,s)`

indicates a minute field in which values can contain leading spaces or zeroes and be one or two characters wide. If you specified the following date format property:

```
%(h,s):$(n,s):$(s,s)
```

Then the following times would all be valid:

20: 6:58

20:06:58

20:6:58

- v Use this option in conjunction with the %SSS or %SSSSSS tags to represent milliseconds or microseconds in variable-width format. So the time property:

```
%(SSS,v)
```

represents values in the range 0 to 999. (If you omit the v option then the range of values would be 000 to 999.)

- u Use this option to render the am/pm text in uppercase on output.
- w Use this option to render the am/pm text in lowercase on output.
- c Specify this option to use a comma as the decimal separator in the %ss.N tag.
- C Specify this option to use a period as the decimal separator in the %ss.N tag.

The c and C options override the default setting of the locale.

The locale for determining the setting of the am/pm string and the default decimal separator can be controlled through the locale tag. This has the format:

```
%(L,'locale')
```

Where *locale* specifies the locale to be set using the *language_COUNTRY.variant* naming convention supported by ICU. See *NLS Guide* for a list of locales. The default locale for am/pm string and separators markers is English unless overridden by a %L tag or the APT_IMPEXP_LOCALE environment variable (the tag takes precedence over the environment variable if both are set).

Use the locale tag in conjunction with your time format, for example:

```
%L('es')%HH:%nn %aa
```

Specifies the Spanish locale.

The format string is subject to the restrictions laid out in the following table. A format string can contain at most one tag from each row. In addition some rows are mutually incompatible, as indicated in the 'incompatible with' column. When some tags are used the format string requires that other tags are present too, as indicated in the 'requires' column.

Table 36. Format tag restrictions

Element	Numeric format tags	Text format tags	Requires	Incompatible with
hour	%hh, %h, %HH, %H	-	-	-
am/pm marker	-	%aa	hour (%HH)	hour (%hh)
minute	%nn, %n	-	-	-
second	%ss, %s	-	-	-
fraction of a second	%ss.N, %s.N, %SSS, %SSSSSS	-	-	-

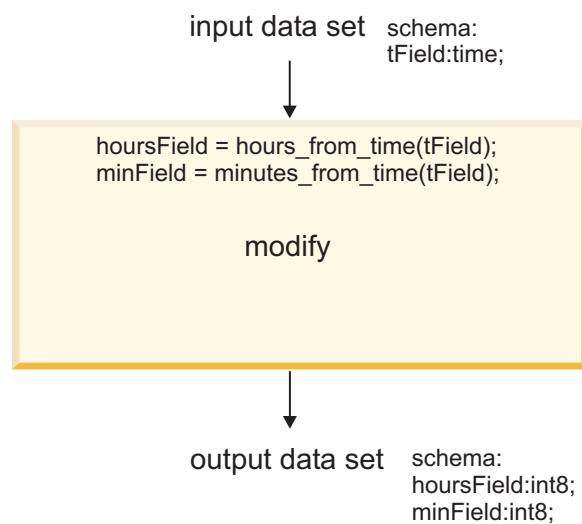
You can include literal text in your date format. Any Unicode character other than null, backslash, or the percent sign can be used (although it is better to avoid control codes and other non-graphic characters). The following table lists special tags and escape sequences:

Tag	Escape sequence
%%	literal percent sign
\%	literal percent sign
\n	newline
\t	horizontal tab
\\\	single backslash

Converting Time Fields to Integers Example

The following figure shows the conversion of time field to two 8-bit integers, where:

- The `hours_from_time` conversion specification extracts the hours portion of a time field and writes it to an 8-bit integer
- The `minutes_from_time` conversion specification extracts the minutes portion of a time field and writes it to an 8-bit integer.



The following osh code converts the hours portion of `tField` to the int8 `hoursField` and the minutes portion to the int8 `minField`:

```
modify 'hoursField = hours_from_time(tField);  
minField = minutes_from_time(tField);'
```

Timestamp field conversions

By default WebSphere DataStage converts a source timestamp field only to either a time or date destination field. However, you can invoke the modify operator to perform other conversions.

Conversion Specification	Description
<code>dfloatField = seconds_since_timestamp [timestamp](tsField)</code>	seconds_since from timestamp

Conversion Specification	Description
<code>tsField = timestamp_from_seconds_since [timestamp](dfloatField)</code>	timestamp from seconds_since
<code>stringField = string_from_timestamp [timestamp_format timestamp_uformat](tsField)</code>	strings and ustrings from timestamp
<code>ustringField = ustring_from_timestamp [timestamp_format timestamp_uformat](tsField)</code>	Converts the timestamp to a string or ustring representation using the specified <code>timestamp_format</code> or <code>timestamp_uformat</code> .
	By default, the string format is %yyyy-%mm-%dd hh:mm:ss. The <code>timestamp_format</code> and <code>timestamp_uformat</code> options are described in "timestamp Formats".
<code>int32Field = timet_from_timestamp(tsField)</code>	timet_from_timestamp <code>int32Field</code> contains a timestamp as defined by the UNIX timet representation.
<code>dateField = date_from_timestamp(tsField)</code>	date from timestamp Converts the timestamp to a date representation.
<code>tsField = timestamp_from_string [timestamp_format timestamp_uformat](stringField)</code>	timestamp from strings and ustrings
<code>tsField = timestamp_from_ustring [timestamp_format timestamp_uformat](usringField)</code>	Converts the string or ustring to a timestamp representation using the specified <code>timestamp_format</code> .
	By default, the string format is yyyy-mm-dd hh:mm:ss. The <code>timestamp_format</code> and <code>timestamp_uformat</code> options are described in "timestamp Formats".
<code>tsField = timestamp_from_timet(int32Field)</code>	timestamp from time_t <code>int32Field</code> must contain a timestamp as defined by the UNIX time_t representation.
<code>tsField = timestamp_from_date [time](dateField)</code>	timestamp from date The <code>time</code> argument optionally specifies the time to be used in building the timestamp result and must be in the form hh:mm:ss. If omitted, the time defaults to midnight.
<code>tsField = timestamp_from_time [date](timeField)</code>	timestamp from time The <code>date</code> argument is required. It specifies the date portion of the timestamp and must be in the form yyyy-mm-dd.
<code>tsField = timestamp_from_date_time (date , time)</code>	Returns a timestamp from <code>date</code> and <code>time</code> . The <code>date</code> specifies the date portion (yyyy-nn-dd) of the timestamp. The <code>time</code> argument specifies the time to be used when building the timestamp. The time argument must be in the hh:nn:ss format.
<code>timeField = time_from_timestamp(tsField)</code>	time from timestamp

Timestamp conversion of a numeric field can be used with any WebSphere DataStage numeric data type. WebSphere DataStage performs the necessary conversions to translate a conversion result to the numeric data type of the destination. For example, you can use the conversion `timet_from_timestamp` to convert a timestamp to an int32, dfloat, and so on.

timestamp formats

The `string_from_timestamp`, `cstring_from_timestamp`, `timestamp_from_string`, and `timestamp_from_cstring` conversions take a timestamp format or timestamp uformat argument. The default format of the timestamp contained in the string is `yyyy-mm-dd hh:mm:ss`. However, you can specify an optional format string defining the data format of the string field.

timestamp format

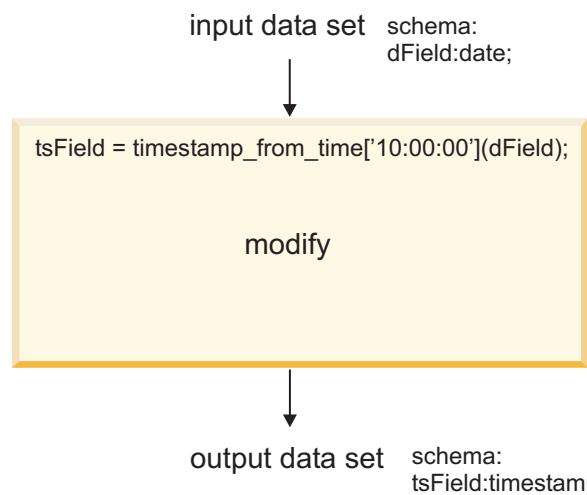
The format options of timestamp combine the formats of the date and time data types. The default timestamp format is as follows:

```
%yyyy-%mm-%dd %hh:%mm:%ss
```

timestamp uformat

For timestamp uformat, concatenate the date uformat with the time uformat. The two formats can be in any order, and their components can be mixed. These formats are described in "date Uformat" under "Date Field Conversions" and "time Uformat" under "Time Field Conversions" ..

The following diagram shows the conversion of a date field to a timestamp field. As part of the conversion, the operator sets the time portion of the timestamp to 10:00:00.



To specify the `timestamp_from_date` conversion and set the time to 10:00:00, use the following osh command:

```
modify 'tsField=timestamp_from_date['10:00:00'](dField);'
```

The modify operator and nulls

All WebSphere DataStage data types support nulls. As part of processing a record, an operator can detect a null and take the appropriate action, for example, it can omit the null field from a calculation or signal an error condition.

WebSphere DataStage represents nulls in two ways.

- It allocates a single bit to mark a field as null. This type of representation is called an out-of-band null.
- It designates a specific field value to indicate a null, for example a numeric field's most negative possible value. This type of representation is called an in-band null. In-band null representation can be disadvantageous because you must reserve a field value for nulls and this value cannot be treated as valid data elsewhere.

The modify operator can change a null representation from an out-of-band null to an in-band null and from an in-band null to an out-of-band null.

The record schema of an operator's input or output data set can contain fields defined to support out-of-band nulls. In addition, fields of an operator's interface might also be defined to support out-of-band nulls. The next table lists the rules for handling nullable fields when an operator takes a data set as input or writes to a data set as output.

Source Field	Destination Field	Result
not_nullable	not_nullable	Source value propagates to destination.
not_nullable	nullable	Source value propagates; destination value is never null.
nullable	not_nullable	If the source value is not null, the source value propagates. If the source value is null, a fatal error occurs, unless you apply the modify operator, as in "Out-of-Band to Normal Representation".
nullable	nullable	Source value or null propagates.

Out-of-band to normal representation

The modify operator can change a field's null representation from a single bit to a value you choose, that is, from an out-of-band to an in-band representation. Use this feature to prevent fatal data type conversion errors that occur when a destination field has not been defined as supporting nulls. See "Data Type Conversion Errors".

To change a field's null representation from a single bit to a value you choose, use the following osh syntax:

```
destField[: dataType] = handle_null ( sourceField , value )
```

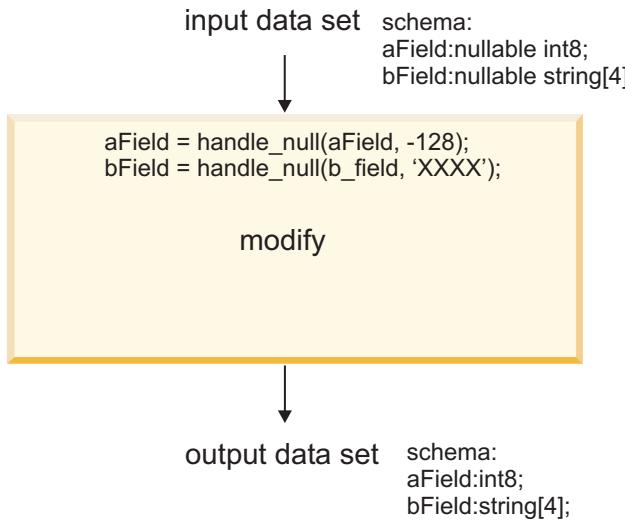
where:

- *destField* is the destination field's name.
- *dataType* is its optional data type; use it if you are also converting types.
- *sourceField* is the source field's name
- *value* is the value you wish to represent a null in the output. The *destField* is converted from an WebSphere DataStage out-of-band null to a value of the field's data type. For a numeric field *value* can be a numeric value, for decimal, string, time, date, and timestamp fields, *value* can be a string.

Conversion specifications are described in:

- "Date Field Conversions"
- "Decimal Field Conversions"
- "String and Ustring Field Conversions"
- "Time Field Conversions"
- "Timestamp Field Conversions"

For example, the diagram shows the modify operator converting the WebSphere DataStage out-of-band null representation in the input to an output value that is written when a null is encountered:



While in the input fields a null takes the WebSphere DataStage out-of-band representation, in the output a null in aField is represented by -128 and a null in bField is represented by ASCII XXXX (0x59 in all bytes).

To make the output aField contain a value of -128 whenever the input contains an out-of-band null, and the output bField contain a value of 'XXXX' whenever the input contains an out-of-band null, use the following osh code:

```
$ modifySpec = "aField = handle_null(aField, -128);  
              bField = handle_null(bField, 'XXXX');"  
$ osh " ... | modify '$modifySpec' | ... "
```

Notice that a shell variable (modifySpec) has been defined containing the specifications passed to the operator.

Normal to out-of-band representation

The modify operator can change a field's null representation from a normal field value to a single bit, that is, from an in-band to an out-of-band representation.

To change a field's null representation to out-of band use the following osh syntax:

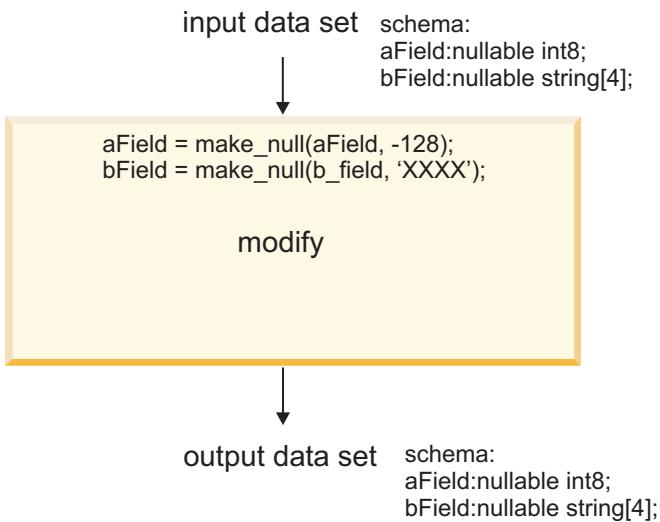
```
destField [: dataType ] = make_null(sourceField , value );
```

Where:

- *destField* is the destination field's name.
- *dataType* is its optional data type; use it if you are also converting types.
- *sourceField* is the source field's name.
- *value* is the value of the source field when it is null.

A conversion result of *value* is converted from a WebSphere DataStage out-of-band null to a value of the field's data type. For a numeric field *value* can be a numeric value, for decimal, string, time, date, and timestamp fields, *value* can be a string.

For example, the diagram shows a modify operator converting the value representing a null in an input field (-128 or 'XXXX') to the WebSphere DataStage single-bit null representation in the corresponding field of the output data set:



In the input a null value in *aField* is represented by -128 and a null value in *bField* is represented by ASCII XXXX, but in both output fields a null value is represented by WebSphere DataStage's single bit.

The following osh syntax causes the *aField* of the output data set to be set to the WebSphere DataStage single-bit null representation if the corresponding input field contains -128 (in-band-null), and the *bField* of the output to be set to WebSphere DataStage's single-bit null representation if the corresponding input field contains 'XXXX' (in-band-null).

```
$modifySpec = "aField = make_null(aField, -128);  
bField = make_null(bField, 'XXXX');"  
$ osh "... | modify '$modifySpec' | ... "
```

Notice that a shell variable (*modifySpec*) has been defined containing the specifications passed to the operator.

The null and notnull conversions

WebSphere DataStage supplies two other conversions to use with nullable fields, called *null* and *notnull*.

- The *null* conversion sets the destination field to 1 if the source field is null and to 0 otherwise.
- The *notnull* conversion sets the destination field to 1 if the source field is not null and to 0 if it is null.

In osh, define a *null* or *notnull* conversion as follows:

```
destField [:dataType] = null( sourceField ); destField [:dataType] = notnull( sourceField );
```

By default, the data type of the destination field is *int8*. Specify a different destination data type to override this default. WebSphere DataStage issues a warning if the source field is not nullable or the destination field is nullable.

The modify operator and partial schemas

You can invoke a *modify* operator to change certain characteristics of a data set containing a partial record schema. ("Complete and Partial Schemas" discusses partial schemas and their definition.) When the *modify* operator drops a field from the intact portion of the record, it drops only the field definition. The contents of the intact record are not altered. Dropping the definition means you can no longer access that portion of the intact record.

The modify operator and vectors

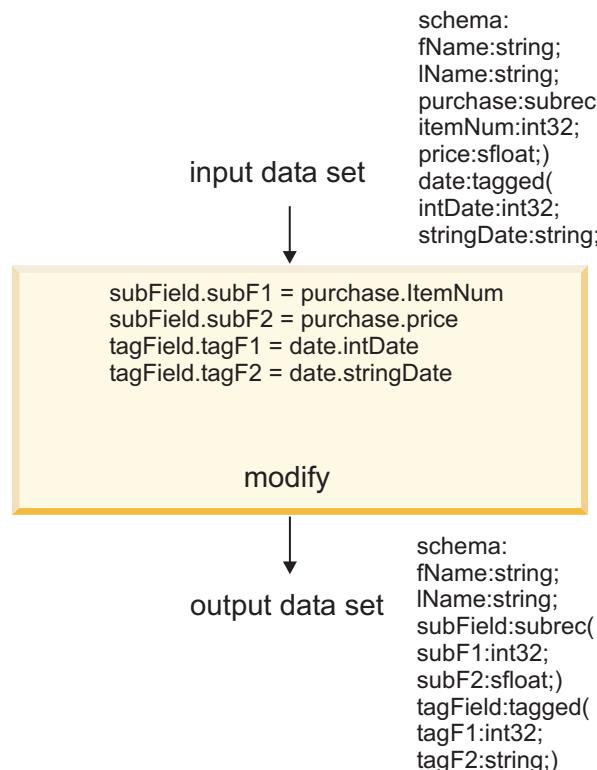
The modify operator cannot change the length of a vector or the vector's length type (fixed or variable). However, you can use the operator either to translate the name of a vector or to convert the data type of the vector elements.

The modify operator and aggregate schema components

Data set and operator interface schema components can contain aggregates (subrecords and tagged aggregates). You can apply modify adapters to aggregates, with these restrictions:

- Subrecords might be translated only to subrecords.
- Tagged fields might be translated only to tagged fields.
- Within subrecords and tagged aggregates, only elements of the same level can be bound by the operator.

The diagram shows an operation in which both the input data set and the output contain aggregates:



In this example, purchase contains an item number and a price for a purchased item; date contains the date of purchase represented as either an integer or a string. You must translate the aggregate purchase to the interface component subField and the tagged component date to tagField.

To translate aggregates:

1. Translate the aggregate of an input data set to an aggregate of the output.

To translate purchase, the corresponding output component must be a compatible aggregate type. The type is subrecord and the component is subField. The same principle applies to the elements of the subrecord.

2. Translate the individual fields of the data set's aggregate to the individual fields of the operator's aggregate.

If multiple elements of a tagged aggregate in the input are translated, they must all be bound to members of a single tagged component of the output's record schema. That is, all elements of *tagField* must be bound to a single aggregate in the input.

Here is the osh code to rename purchase.price to subField.subF2.

```
$ modifySpec = "subField = purchase;
    subField.subF1 = purchase.itemNum;
    subField.subF2 = purchase.price;
    tagField = date;
    tagField.tagF1 = date.intDate;
    tagField.tagF2 = date.stringDate; );
"
$ osh "... | modify '$modifySpec' | ..."
```

Notice that a shell variable (*modifySpec*) has been defined containing the specifications passed to the operator.

Aggregates might contain nested aggregates. When you translate nested aggregates, all components at one nesting level in an input aggregate must be bound to all components at one level in an output aggregate.

The table shows sample input and output data sets containing nested aggregates. In the input data set, the record *purchase* contains a subrecord *description* containing a description of the item:

Level	Schema 1 (for input data set)	Level	Schema 2 (for output data set)
0	purchase: subrec (...
1	itemNum: int32;	n	subField (
1	price: sfloat;	n + 1	subF1;
1	description: subrec; (n + 1	subF2:);
2	color: int32;		...
2	size:int8;);		
);		

Note that:

- itemNum and price are at the same nesting level in purchase.
- color and size are at the same nesting level in purchase.
- subF1 and subF2 are at the same nesting level in subField.

You can bind:

- purchase.itemNum and purchase.price (both level 1) to subField.subF1 and subField.subF2, respectively
- purchase.description.color and purchase.description.size (both level 2) to subField.subF1 and subField.subF2, respectively

You cannot bind two elements of purchase at different nesting levels to subF1 and subF2. Therefore, you cannot bind itemNum (level 1) to subF1 and size (level2) to subF2.

Note: WebSphere DataStage features several operators that modify the record schema of the input data set and the level of fields within records. Two of them act on tagged subrecords. See the topic on the restructure operators.

Allowed conversions

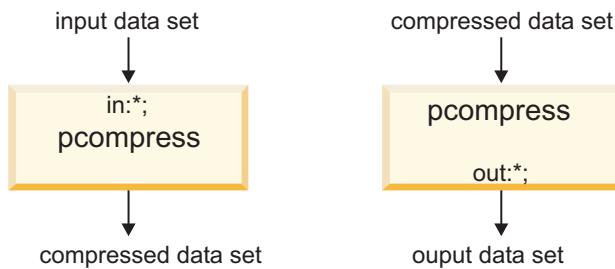
The table lists all allowed data type conversions arranged alphabetically. The form of each listing is:
conversion_name (*source_type* ,*destination_type*)

Conversion Specification
date_from_days_since (int32 , date)
date_from Julian_day (int32 , date)
date_from_string (string , date)
date_from_timestamp (timestamp , date)
date_from_ustring (ustring , date)
days_since_from_date (date , int32)
decimal_from_decimal (decimal , decimal)
decimal_from_dfloat (dfloat , decimal)
decimal_from_string (string , decimal)
decimal_from_ustring (ustring , decimal)
dfloat_from_decimal (decimal , dfloat)
hours_from_time (time , int8)
int32_from_decimal (decimal , int32)
int64_from_decimal (decimal , int64)
julian_day_from_date (date , uint32)
lookup_int16_from_string (string , int16)
lookup_int16_from_ustring (ustring , int16)
lookup_string_from_int16 (int16 , string)
lookup_string_from_uint32 (uint32 , string)
lookup_uint32_from_string (string , uint32)
lookup_uint32_from_ustring (ustring , uint32)
lookup_ustring_from_int16 (int16 , ustring)
lookup_ustring_from_int32 (uint32 , ustring)
lowercase_string (string , string)
lowercase_ustring (ustring , ustring)
mantissa_from_dfloat (dfloat , dfloat)
mantissa_from_decimal (decimal , dfloat)
microseconds_from_time (time , int32)
midnight_seconds_from_time (time , dfloat)
minutes_from_time (time , int8)
month_day_from_date (date , int8)
month_from_date (date , int8)
next_weekday_from_date (date , date)
notnull (any , int8)
null (any , int8)
previous_weekday_from_date (date , date)
raw_from_string (string , raw)

Conversion Specification
raw_length (raw , int32)
seconds_from_time (time , dfloat)
seconds_since_from_timestamp (timestamp , dfloat)
string_from_date (date , string)
string_from_decimal (decimal , string)
string_from_time (time , string)
string_from_timestamp (timestamp , string)
string_from_ushort (ushort , string)
string_length (string , int32)
substring (string , string)
time_from_midnight_seconds (dfloat , time)
time_from_string (string , time)
time_from_timestamp (timestamp , time)
time_from_ushort (ushort , time)
timestamp_from_date (date , timestamp)
timestamp_from_seconds_since (dfloat , timestamp)
timestamp_from_string (string , timestamp)
timestamp_from_time (time , timestamp)
timestamp_from_timet (int32 , timestamp)
timestamp_from_ushort (ushort , timestamp)
timet_from_timestamp (timestamp , int32)
uint64_from_decimal (decimal , uint64)
uppercase_string (string , string)
uppercase_ushort (ushort , ushort)
u_raw_from_string (ushort , raw)
ushort_from_date (date , ushort)
ushort_from_decimal (decimal , ushort)
ushort_from_string (string , ushort)
ushort_from_time (time , ushort)
ushort_from_timestamp (timestamp , ushort)
ushort_length (ushort , int32)
u_substring (ushort , ushort)
weekday_from_date (date , int8)
year_day_from_date (date , int16)
year_from_date (date , int16)
year_week_from_date (date , int8)

pcompress operator

The pcompress operator uses the UNIX compress utility to compress or expand a data set. The operator converts a WebSphere DataStage data set from a sequence of records into a stream of raw binary data; conversely, the operator reconverts the data stream into a WebSphere DataStage data set.



Data flow diagram

The mode of the pcompress operator determines its action. Possible values for the mode are:

- compress: compress the input data set
- expand: expand the input data set

pcompress: properties

Table 37. pcompress operator

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	mode = compress: in: *;
Output interface schema	mode = expand: out: *;
Transfer behavior	in -> out without record modification for a compress/decompress cycle
Execution mode	parallel (default) or sequential
Partitioning method	mode = compress: any mode = expand: same
Collection method	any
Preserve-partitioning flag in output data set	mode = compress: sets mode = expand: propagates
Composite operator	yes:APT_EncodeOperator

Pcompress: syntax and options

```
pcompress [-compress | -expand]  
[-command compress | gzip]
```

Table 38. Pcompress options

Option	Use
-compress	-compress This option is the default mode of the operator. The operator takes a data set as input and produces a compressed version as output.

Table 38. Pcompress options (continued)

Option	Use
-expand	<p>-expand</p> <p>This option puts the operator in expand mode. The operator takes a compressed data set as input and produces an uncompressed data set as output.</p>
-command	<p>-command "compress" "gzip"</p> <p>Optionally specifies the UNIX command to be used to perform the compression or expansion.</p> <p>When you specify "compress" the operator uses the UNIX command, compress -f, for compression and the UNIX command, uncompress, for expansion. When you specify "gzip", the operator uses the UNIX command, gzip -, for compression and the UNIX command, gzip -d -, for expansion.</p>

The default mode of the operator is -compress, which takes a data set as input and produces a compressed version as output. Specifying -expand puts the command in expand mode, which takes a compressed data set as input and produces an uncompressed data set as output.

Compressed data sets

Each record of an WebSphere DataStage data set has defined boundaries that mark its beginning and end. The pcompress operator invokes the UNIX compress utility to change a WebSphere DataStage data set, which is in record format, into raw binary data and vice versa.

Processing compressed data sets

A compressed data set is similar to a WebSphere DataStage data set. A compressed, persistent data set is represented on disk in the same way as a normal data set, by two or more files: a single descriptor file and one or more data files.

A compressed data set cannot be accessed like a standard WebSphere DataStage data set.

A compressed data set cannot be processed by most WebSphere DataStage operators until it is decoded, that is, until its records are returned to their normal WebSphere DataStage format.

Nonetheless, you can specify a compressed data set to any operator that does not perform field-based processing or reorder the records. For example, you can invoke the copy operator to create a copy of the compressed data set.

You can further encode a compressed data set, using an encoding operator, to create a compressed-encoded data set. (See "Encode Operator" .) You would then restore the data set by first decoding and then decompressing it.

Compressed data sets and partitioning

When you compress a data set, you remove its normal record boundaries. The compressed data set must not be repartitioned before it is expanded, because partitioning in WebSphere DataStage is performed record-by-record. For that reason, the pcompress operator sets the preserve-partitioning flag in the output data set. This prevents an WebSphere DataStage operator that uses a partitioning method of any from repartitioning the data set to optimize performance and causes WebSphere DataStage to issue a warning if any operator attempts to repartition the data set.

For an expand operation, the operator takes as input a previously compressed data set. If the preserve-partitioning flag in this data set is not set, WebSphere DataStage issues a warning message.

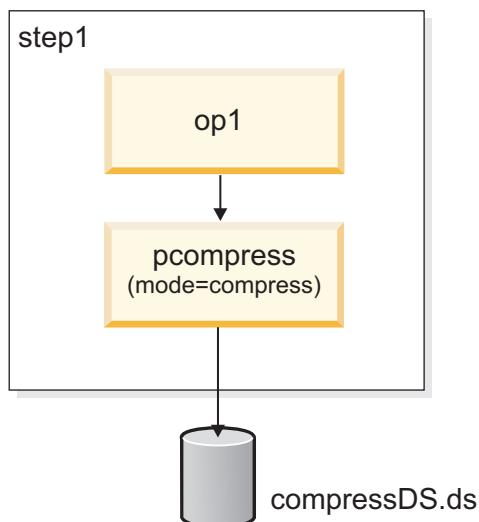
Using orchadmin with a compressed data set

The orchadmin utility manipulates persistent data sets. However, the records of a compressed data set are not in the normal form. For that reason, you can invoke only a subset of the orchadmin commands to manipulate a compressed data set. These commands are as follows:

- delete to delete a compressed data set
- copy to copy a compressed data set
- describe to display information about the data set before compression

Example

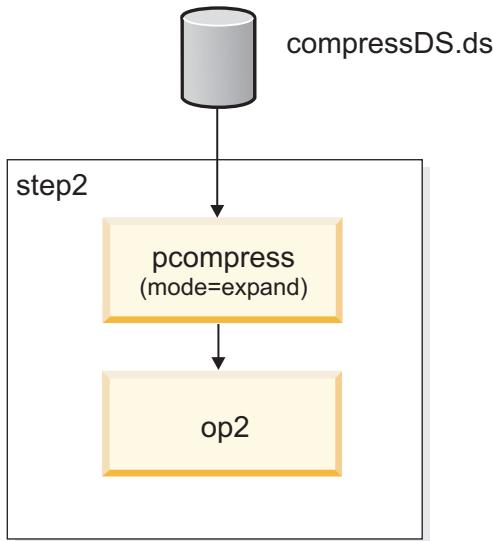
This example consists of two steps. In the first step, the pcompress operator compresses the output of the upstream operator before it is stored on disk:



In osh, the default mode of the operator is -compress, so you need not specify any option:

```
$ osh " ... op1 | pcompress > compressDS.ds "
```

In the next step, the pcompress operator expands the same data set so that it can be used by another operator.



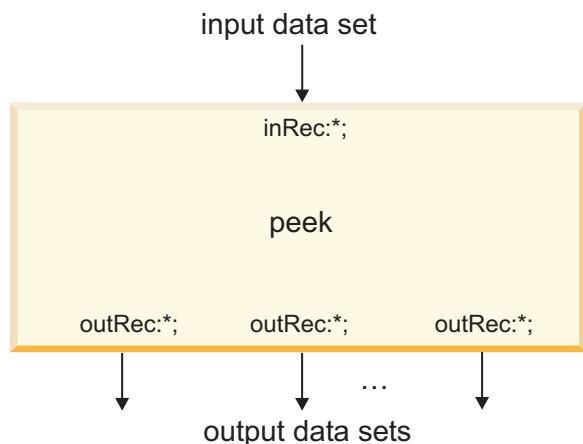
Use the osh command:

```
$ osh "pcompress -expand < compressDS.ds | op2 ... "
```

Peek operator

The peek operator lets you print record field values to the screen as the operator copies records from its input data set to one or more output data sets. This might be helpful for monitoring the progress of your job, or to diagnose a bug in your job.

Data flow diagram



peek: properties

Table 39. peek properties

Property	Value
Number of input data sets	1
Number of output data sets	N (set by user)
Input interface schema	inRec: $*$
Output interface schema	outRec: $*$

Table 39. peek properties (continued)

Property	Value
Transfer behavior	inRec -> outRec without record modification
Execution mode	parallel (default) or sequential
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	no

Peek: syntax and options

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
peek
[-all] | [-nrecs numrec]
[-dataset] [-delim string]
[-field fieldname ... ]
[-name]
[-part part_num]
[-period P]
[-skip N]
[-var input_schema_var_name]
```

There are no required options.

Table 40. Peek options

Option	Use
-all	-all Causes the operator to print all records. The default operation is to print 10 records per partition.
-dataset	-dataset Specifies to write the output to a data set. The record schema of the output data set is: record(rec:string);
-delim	-delim <i>string</i> Uses the string <i>string</i> as a delimiter on top-level fields. Other possible values for this are: n1 (newline), tab, and space. The default is the space character.
-field	-field <i>fieldname</i> Specifies the field whose values you want to print. The default is to print all field values. There can be multiple occurrences of this option.
-name	-name Causes the operator to print the field name, followed by a colon, followed by the field value. By default, the operator prints only the field value, followed by a space.

Table 40. Peek options (continued)

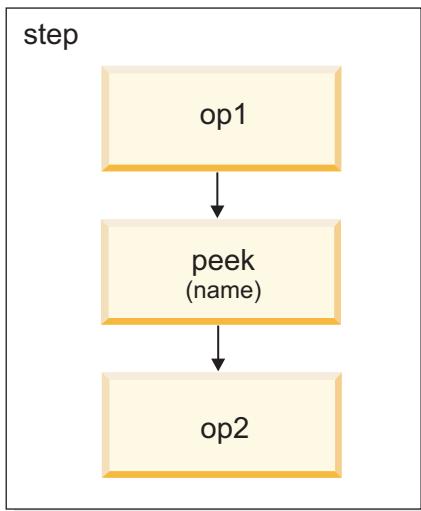
Option	Use
-nrecs	<p>-nrecs <i>numrec</i></p> <p>Specifies the number of records to print per partition. The default is 10.</p>
-period	<p>-period <i>p</i></p> <p>Cause the operator to print every <i>p</i>th record per partition, starting with first record. <i>p</i> must be ≥ 1.</p>
-part	<p>-part <i>part_num</i></p> <p>Causes the operator to print the records for a single partition number. By default, the operator prints records from all partitions.</p>
-skip	<p>-skip <i>n</i></p> <p>Specifies to skip the first <i>n</i> records of every partition. The default value is 0.</p>
-var	<p>-var <i>input_schema_var_name</i></p> <p>Explicitly specifies the name of the operator's input schema variable. This is necessary when your input data set contains a field named inRec.</p>

Using the operator

The peek operator reads the records from a single input data set and copies the records to zero or more output data sets. For a specified number of records per partition, where the default is 10, the record contents are printed to the screen.

By default, the value of all record fields is printed. You can optionally configure the operator to print a subset of the input record fields.

For example, the diagram shows the peek operator, using the -name option to dump both the field names and field values for ten records from every partition of its input data set, between two other operators in a data flow:



This data flow can be implemented with the osh command:

```
$ osh " ... op1 | peek -name | op2 ... "
```

The output of this example is similar to the following:

```
ORCHESTRATE VX.Y
16:30:49 00 APT configuration file: ./config.apt
From[1,0]: 16:30:58 00
Name:Mary Smith Age:33 Income:17345 Zip:02141 Phone:555-1212
From[1,1]: 16:30:58 00
Name:John Doe Age:34 Income:67000 Zip:02139 Phone:555-2121
16:30:59 00 Step execution finished with status = OK.
```

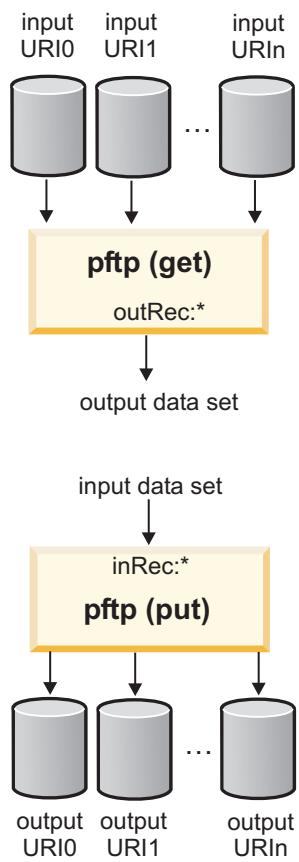
PFTP operator

The PFTP (parallel file transfer protocol) Enterprise operator transfers files to and from multiple remote hosts.

It works by forking an FTP client executable to transfer a file to or from multiple remote hosts using a URI (Uniform Resource Identifier).

This section describes the operator and also addresses issues such as restartability, describing how you can restart an ftp transfer from the point where it was stopped if it fails. The restart occurs at the file boundary.

Data flow diagram



Operator properties

Property	Value
Number of input data sets	$0 \leq N \leq 1$ (zero or one input data set in put mode)
Number of output data sets	$0 \leq N \leq 1$ (zero or one output data set in get mode)
Input interface schema	inputRec:* (in put mode)
Output interface schema	outputRec:* (in get mode)
Transfer behavior	inputRec:* is exported according to a user supplied schema and written to a pipe for ftp transfer. outputRec:* is imported according to a user supplied schema by reading a pipe written by ftp.
Default execution mode	Parallel
Input partitioning style	None
Output partitioning style	None
Partitioning method	None
Collection method	None
Preserve-partitioning flag in input data set	Not Propagated
Preserve-partitioning flag in output data set	Not Propagated
Restartable	Yes
Combinable operator	No
Consumption pattern	None

Property	Value
Composite Operator	No

Pftp: syntax and options

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes. The options within [] are optional.

```
pftp
  -mode put | get
  [-schema schema | -schemafile schemafile]
  -uri uri1 [-open_command cmd] [-uri uri2 [-open_command cmd]... ]
  [-ftp_call ftp_command]
  [-user user1 [-user user2...]]
  [-password password1 [-password password2...]]
  [-overwrite]
  [-transfer_type[ascii,binary]]
  [-xfer_mode[ftp,sftp]]
  [[[-restartable_transfer [-job_id job_id] [-checkpointdir checkpoint_dir]]]
  [-abandon_transfer [-job_id job_id] [-checkpointdir checkpoint_dir]]
  [-restart_transfer [-job_id job_id] [-checkpointdir checkpoint_dir]]]]
```

Table 41. Pftp2 options

Option	Use
-mode	-mode put -mode get put or get

Table 41. Pftp2 options (continued)

Option	Use
-uri	<p>-uri <i>uri1</i> [-uri <i>uri2...</i>]</p> <p>The URIs (Uniform Resource Identifiers) are used to transfer or access files from or to multiple hosts.</p> <p>There can be multiple URIs.</p> <p>You can specify one or more URIs, or a single URI with a wild card in the path to retrieve all the files using the wild character pointed by the URI. pftp collects all the retrieved file names before starting the file transfer. pftp supports limited wild carding. Get commands are issued in sequence for files when a wild card is specified. You can specify an absolute or a relative pathname.</p> <p>For put operations the syntax for a relative path is: <code>ftp://remotehost.domain.com/path/remotefilename</code></p> <p>Where <i>path</i> is the relative path of the user's home directory.</p> <p>For put operations the syntax for an absolute path is: <code>ftp://remotehost.domain.com//path/remotefilename</code></p> <p>While connecting to the mainframe system, the syntax for an absolute path is: <code>ftp://remotehost.domain.com/\'path.remotefilename\'</code></p> <p>Where <i>path</i> is the absolute path of the user's home directory.</p> <p>For get operations the syntax for a relative path is: <code>ftp://host/path/filename</code></p> <p>Where <i>path</i> is the relative path of the user's home directory.</p> <p>For get operations the syntax for an absolute path is: <code>ftp://host//path/filename</code></p> <p>While connecting to the mainframe system, the syntax for an absolute path is: <code>ftp://host//\'path.remotefilename\'</code></p> <p>Where <i>path</i> is the absolute path of the user's home directory.</p>
-open_command	<p>-open_command <i>cmd</i></p> <p>Needed only if any operations need to be performed besides navigating to the directory where the file exists</p> <p>This is a sub-option of the URI option. At most one open_command can be specified for an URI.</p> <p>Example:</p> <pre>-uri ftp://remotehost/fileremote1.dat -open_command verbose</pre>

Table 41. Pftp2 options (continued)

Option	Use
-user	<p><code>-user username1 [-user username2...]</code></p> <p>With each URI you can specify the User Name to connect to the URI. If not specified, the ftp will try to use the .netrc file in the user's home directory.</p> <p>There can be multiple user names.</p> <p>User1 corresponds to URI1. When the number of usernames is less than the number of URIs, the last username is set for the remaining URIs.</p> <p>Example:</p> <pre>-user User1 -user User2</pre>
-password	<p><code>-password password1 [-password password1]</code></p> <p>With each URI you can specify the Password to connect to the URI. If not specified, the ftp will try to use the .netrc file in the user's home directory.</p> <p>There can be multiple passwords.</p> <p>Password1 corresponds to URI1. When the number of passwords is less than the number of URIs, the last password is set for the remaining URIs.</p> <p>Note The number of passwords should be equal to the number of usernames.</p> <p>Example:</p> <pre>-password Secret1 -password Secret2</pre>
-schema	<p><code>-schema schema</code></p> <p>You can specify the schema for get or put operations.</p> <p>This option is mutually exclusive with -schemafile.</p> <p>Example:</p> <pre>-schema 'record(name:string;)'</pre>
-schemafile	<p><code>-schemafile schemafile</code></p> <p>You can specify the schema for get or put operations. in a schema file.</p> <p>This option is mutually exclusive with -schema.</p> <p>Example:</p> <pre>-schemafile file.schema</pre>
-ftp_call	<p><code>-ftp_call cmd</code></p> <p>The ftp command to call for get or put operations. The default is 'ftp'.</p> <p>You can include absolute path with the command.</p> <p>Example:</p> <pre>-ftp_call '/opt/gnu/bin/wuftp'.</pre>

Table 41. Pftp2 options (continued)

Option	Use
-force_config_file_parallelism	<p>-force_config_file_parallelism</p> <p>Optionally limits the number of pftp players via the APT_CONFIG_FILE configuration file.</p> <p>The operator executes with a maximum degree of parallelism as determined by the configuration file.</p> <p>The operator will execute with a lesser degree of parallelism if the number of get arguments is less than the number of nodes in the Configuration file.</p> <p>In some cases this might result in more than one file being transferred per player.</p>
-overwrite	<p>-overwrite</p> <p>Overwrites remote files in ftp put mode.</p> <p>When this option is not specified, the remote file is not overwritten.</p>

Table 41. Pftp2 options (continued)

Option	Use
-restartable_transfer -restart_transfer -abandon_transfer	<p>This option is used to initiate a restartable ftp transfer.</p> <p>The restartability option in get mode will reinitiate ftp transfer at the file boundary. The transfer of the files that failed half way is restarted from the beginning or zero file location. The file URIs that were transferred completely are not transferred again. Subsequently, the downloaded URIs are imported to the data set from the downloaded temporary folder path.</p> <ul style="list-style-type: none"> • A restartable pftp session is initiated as follows: <pre>osh "pftp -uri ftp://remotehost/file.dat -user user -password secret -restartable_transfer -jobid 100 -checkpointdir 'chkdir' -mode put < input.ds"</pre> <ul style="list-style-type: none"> • -restart_transfer :If the transfer fails, to restart the transfer again, the restartable pftp session is resumed as follows: <pre>osh "pftp -uri ftp://remotehost/file.dat -user user -password secret -restart_transfer -jobid 100 -checkpointdir 'chkdir' -mode put < input.ds"</pre> <ul style="list-style-type: none"> • -abandon_transfer : Used to abort the operation, the restartable pftp session is abandoned as follows: <pre>osh "pftp -uri ftp://remotehost/file.dat -user user -password secret -abandon_transfer -jobid 100 -checkpointdir 'chkdir' -mode put < input.ds"</pre>

Table 41. Pftp2 options (continued)

Option	Use
-job_id	This is an integer to specify job identifier of restartable transfer job. This is a dependent option of -restartable_transfer, -restart_transfer, or -abandon_transfer Example: <code>-job_id 101</code>
-checkpointdir	This is the directory name/path of location where pftp restartable job id folder can be created. The checkpoint folder must exist. Example: <code>-checkpointdir "/apt/linux207/orch_master/apt/folder"</code>
-transfer_type	This option is used to specify the data transfer type. You can either choose ASCII or Binary as the data transfer type. Example: <code>-transfer_type binary</code>
-xfer_mode	This option is used to specify data transfer protocol. You can either choose FTP or SFTP mode of data transfer. Example: <code>-xfer_mode sftp</code>

Restartability

You can specify that the FTP operation runs in restartable mode. To do this you:

1. Specify the -restartable_transfer option
2. Specify a unique job_id for the transfer
3. Optionally specify a checkpoint directory for the transfer using the -checkpointdir directory (if you do not specify a checkpoint directory, the current working directory is used)

When you run the job that performs the FTP operation, information about the transfer is written to a restart directory identified by the job id located in the checkpoint directory prefixed with the string "pftp_jobid_". For example, if you specify a job_id of 100 and a checkpoint directory of /home/bgamsworth/checkpoint the files would be written to /home/bgamsworth/checkpoint/pftp_jobid_100.

If the FTP operation does not succeed, you can rerun the same job with the option set to restart or abandon. For a production environment you could build a job sequence that performed the transfer, then tested whether it was successful. If it was not, another job in the sequence could use another PFTP operator with the restart transfer option to attempt the transfer again using the information in the restart directory.

For get operations, WebSphere DataStage reinitiates the FTP transfer at the file boundary. The transfer of the files that failed half way is restarted from the beginning or zero file location. The file URIs that were transferred completely are not transferred again. Subsequently, the downloaded URIs are imported to the data set from the temporary folder path.

If the operation repeatedly fails, you can use the abandon_transfer option to abandon the transfer and clear the temporary restart directory.

pivot operator

Use the Pivot Enterprise stage to pivot data horizontally.

The pivot operator maps a set of fields in an input row to a single column in multiple output records. This type of mapping operation is known as horizontal pivoting. The data output by the pivot operator usually has fewer fields, but more records than the input data. You can map several sets of input fields to several output columns. You can also output any of the fields in the input data with the output data.

You can generate a pivot index that will assign an index number to each record with a set of pivoted data.

Properties: pivot operator

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec:*
Output interface schema	outRec:*, pivotField:*, ..., pivotFieldn:*, pivotIndex:int;
Execution mode	parallel (default) or sequential
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	no
Combinable operator	yes

The pivot operator:

- Takes any single data set as input
- Has an input interface schema consisting of a single schema variable *inRec* .
- Copies the input data set to the output data set, pivoting data in multiple input fields to single output fields in multiple records.

Pivot: syntax and options

```
pivot -horizontal  
-derive field_name  
-from field_name [-from field_name]...  
-type type  
[-derive field_name  
-from field_name [-from field_name]...  
-type type]...  
[-index field_name]
```

Table 42. Pivot options

Option	Use
-horizontal	-horizontal Specifies that the operator will perform a horizontal pivot operation.

Table 42. Pivot options (continued)

Option	Use
-derive	-derive <i>field_name</i> Specifies a name for an output field.
-from	-from <i>field_name</i> Specifies the name of the input field from which the output field is derived.
-type	-type <i>type</i> Specifies the type of the output field.
-index	-index <i>field_name</i> Specifies that an index field will be generated for pivotted data.

Pivot: examples

In this example you use the pivot operator to pivot the data shown in the first table to produce the data shown in the second table. This example has a single pivotted output field and a generated index added to the data.

Table 43. Simple pivot operation - input data

REPID	last_name	Jan_sales	Feb_sales	Mar_sales
100	Smith	1234.08	1456.80	1578.00
101	Yamada	1245.20	1765.00	1934.22

Table 44. Simple pivot operation - output data

REPID	last_name	Q1sales	Pivot_index
100	Smith	1234.08	0
100	Smith	1456.80	1
100	Smith	1578.00	2
101	Yamada	1245.20	0
101	Yamada	1765.00	1
101	Yamada	1934.22	2

The osh command is:

```
$ osh "pivot -horizontal
-derive 'REPID' -from 'REPID' -type 'string'-index 'pivot_index'
-derive 'last_name' -from 'last_name' -type 'string'
-derive 'Q1sales' -from 'Jan_sales' -from 'Feb_sales'
-from 'Mar_sales'-type 'decimal[10,2]"'
```

In this example, you use the pivot operator to pivot the data shown in the first table to produce the data shown in the second table. This example has multiple pivotted output fields.

Table 45. Pivot operation with multiple pivot columns - input data

REPID	last_name	Q1sales	Q2sales	Q3sales	Q4sales
100	Smith	4268.88	5023.90	4321.99	5077.63

Table 45. Pivot operation with multiple pivot columns - input data (continued)

REPID	last_name	Q1sales	Q2sales	Q3sales	Q4sales
101	Yamada	4944.42	5111.88	4500.67	4833.22

Table 46.

REPID	last_name	halfyear1	halfyear2
100	Smith	4268.88	4321.99
100	Smith	5023.90	5077.63
101	Yamada	4944.42	4500.67
101	Yamada	5111.88	4833.22

The osh command is:

```
$ osh "pivot -horizontal  
-derive 'REPID' -from 'REPID' -type 'string'  
-derive 'last_name' -from 'last_name' -type 'string'  
-derive 'halfyear1' -from 'Q1sales' -from 'Q2sales'  
-type 'decimal[10,2]  
-derive 'halfyear2' -from 'Q3sales' -from 'Q4sales'  
-type 'decimal[10,2]"
```

Remdup operator

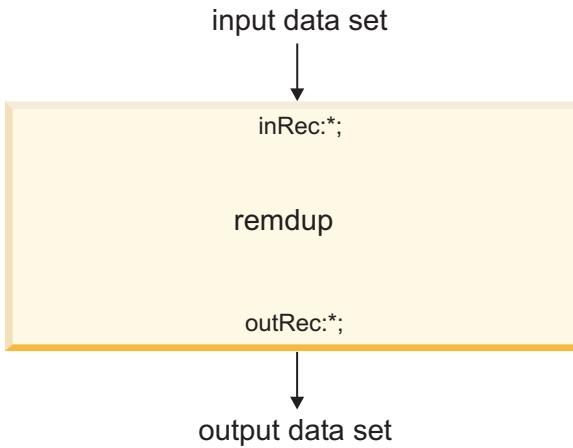
The remove-duplicates operator, remdup, takes a single sorted data set as input, removes all duplicate records, and writes the results to an output data set. Removing duplicate records is a common way of cleansing a data set before you perform further processing.

Two records are considered duplicates if they are adjacent in the input data set and have identical values for the key field(s). A key field is any field you designate to be used in determining whether two records are identical.

For example, a direct mail marketer might use remdup to aid in householding, the task of cleansing a mailing list to prevent multiple mailings going to several people in the same household.

The input data set to the remove duplicates operator must be sorted so that all records with identical key values are adjacent. By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the remdup operator and other operators.

Data flow diagram



remdup: properties

Table 47. remdup properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec -> outRec without record modification
Execution mode	parallel (default) or sequential
Input partitioning style	keys in same partition
Partitioning method	same (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Restartable	yes
Composite operator	no

Remdup: syntax and options

```

remdup
[-key field [-cs | -ci] [-ebcdic] [-hash] [-param params] ...] [-collation_sequence locale |
collation_file_pathname | OFF][-first | -last]

```

Table 48. remdup options

Option	Use
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> Specify a predefined IBM ICU locale Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu /userguide/Collate_Intro.htm</p>
-first	<p>-first</p> <p>Specifies that the first record of the duplicate set is retained. This is the default.</p>
-last	<p>-last</p> <p>Specifies that the last record of the duplicate set is retained. The options -first and -last are mutually exclusive.</p>
-key	<p>-key <i>field</i> [-cs -ci] [-ebcdic] [-hash] [-param <i>params</i>]</p> <p>Specifies the name of a key field. The -key option might be repeated for as many fields as are defined in the input data set's record schema.</p> <p>The -cs option specifies case-sensitive comparison, which is the default. The -ci option specifies a case-insensitive comparison of the key fields.</p> <p>By default data is represented in the ASCII character set. To represent data in the EBCDIC character set, specify the -ebcdic option.</p> <p>The -hash option specifies hash partitioning using this key.</p> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property =value</i> pairs separated by commas.</p>

Removing duplicate records

The remove duplicates operator determines if two adjacent records are duplicates by comparing one-or-more fields in the records. The fields used for comparison are called key fields. When using this operator, you specify which of the fields on the record are to be used as key fields. You can define only

one key field or as many as you need. Any field on the input record might be used as a key field. The determination that two records are identical is based solely on the key field values and all other fields on the record are ignored.

If the values of all of the key fields for two adjacent records are identical, then the records are considered duplicates. When two records are duplicates, one of them is discarded and one retained. By default, the first record of a duplicate pair is retained and any subsequent duplicate records in the data set are discarded. This action can be overridden with an option to keep the last record of a duplicate pair.

In order for the operator to recognize duplicate records as defined by the key fields, the records must be adjacent in the input data set. This means that the data set must have been hash partitioned, then sorted, using the same key fields for the hash and sort as you want to use for identifying duplicates. By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the remdup operator and other operators.

For example, suppose you want to sort the data set first by the Month field and then by the Customer field and then use these two fields as the key fields for the remove duplicates operation.

Use the following osh command:

```
$ osh "remdup -key Month -key Customer < inDS.ds > outDS.ds"
```

In this example, WebSphere DataStage-inserted partition and sort components guarantees that all records with the same key field values are in the same partition of the data set. For example, all of the January records for Customer 86111 are processed together as part of the same partition.

Using options to the operator

By default, the remdup operator retains the first record of a duplicate pair and discards any subsequent duplicate records in the data set. Suppose you have a data set which has been sorted on two fields: Month and Customer. Each record has a third field for the customer's current Balance and the data set can contain multiple records for a customers balance for any month.

When using the remdup operator to cleanse this data set, by default, only the first record is retained for each customer and all the others are discarded as duplicates. For example, if the records in the data set are:

Month	Customer	Balance [®]
Apr	86111	787.38
Apr	86111	459.32
Apr	86111	333.21
May	86111	134.66
May	86111	594.26

The default result of removing duplicate records on this data set is:

Month	Customer	Balance
Apr	86111	787.38
May	86111	134.66

Using the -last option, you can specify that the last duplicate record is to be retained rather than the first. This can be useful if you know, for example, that the last record in a set of duplicates is always the most recent record.

For example, if the osh command is:

```
$ osh "remdup -key Month -key Customer -last < inDS.ds > outDS.ds"
```

the output would given by:

Month	Customer	Balance
Apr	86111	333.21
May	86111	594.26

If a key field is a string, you have a choice about how the value from one record is compared with the value from the next record. The default is that the comparison is case sensitive. If you specify the -ci options the comparison is case insensitive. In osh, specify the -key option with the command:

```
$osh "remdup -key Month -ci < inDS.ds > outDS.ds"
```

With this option specified, month values of "JANUARY" and "January" match, whereas without the case-insensitive option they do not match.

For example, if your input data set is:

Month	Customer	Balance
Apr	59560	787.38
apr	43455	459.32
apr	59560	333.21
May	86111	134.66
might	86111	594.26

The output from a case-sensitive sort is:

Month	Customer	Balance
Apr	59560	787.38
May	86111	134.66
apr	43455	459.32
apr	59560	333.21
might	86111	594.26

Thus the two April records for customer 59560 are not recognized as a duplicate pair because they are not adjacent to each other in the input.

To remove all duplicate records regardless of the case of the *Month* field, use the following statement in osh:

```
$ osh "remdup -key Month -ci -key Customer < inDS.ds > outDS.ds"
```

This causes the result of sorting the input to be:

Month	Customer	Balance
apr	43455	459.32
Apr	59560	787.38
apr	59560	333.21

Month	Customer	Balance
May	86111	134.66
might	86111	594.26

The output from the remdup operator will then be:

Month	Customer	Balance
apr	43455	459.32
Apr	59560	787.38
May	86111	134.66

Using the operator

The remdup operator takes a single data set as input, removes all duplicate records, and writes the results to an output data set. As part of this operation, the operator copies an entire record from the input data set to the output data without altering the record. Only one record is output for all duplicate records.

Example 1: using remdup

The following is an example of use of the remdup operator. Use the osh command:

```
$ osh "remdup -key Month < indDS.ds > outDS.ds"
```

This example removes all records in the same month except the first record. The output data set thus contains at most 12 records.

Example 2: using the -last option

In this example, the last record of each duplicate pair is output rather than the first, because of the -last option. Use the osh command:

```
$ osh "remdup -key Month -last < indDS.ds > outDS.ds"
```

Example 3: case-insensitive string matching

This example shows use of case-insensitive string matching.

Use the osh command:

```
$ osh "remdup -key Month -ci -last < indDS.ds > outDS.ds"
```

The results differ from those of the previous example if the Month field has mixed-case data values such as "May" and "MAY". When the case-insensitive comparison option is used these values match and when it is not used they do not.

Example 4: using remdup with two keys

This example retains the first record in each month for each customer. Therefore there are no more than 12 records in the output for each customer. Use the osh command:

```
$ osh "remdup -key Month -ci -key Customer < indDS.ds > outDS.ds"
```

Sample operator

The sample operator is useful when you are building, testing, or training data sets for use with the WebSphere DataStage data-modeling operators.

The sample operator allows you to:

- Create disjoint subsets of an input data set by randomly sampling the input data set to assign a percentage of records to output data sets. WebSphere DataStage uses a pseudo-random number generator to randomly select, or sample, the records of the input data set to determine the destination output data set of a record.

You supply the initial seed of the random number generator. By changing the seed value, you can create different record distributions each time you sample a data set, and you can recreate a given distribution by using the same seed value.

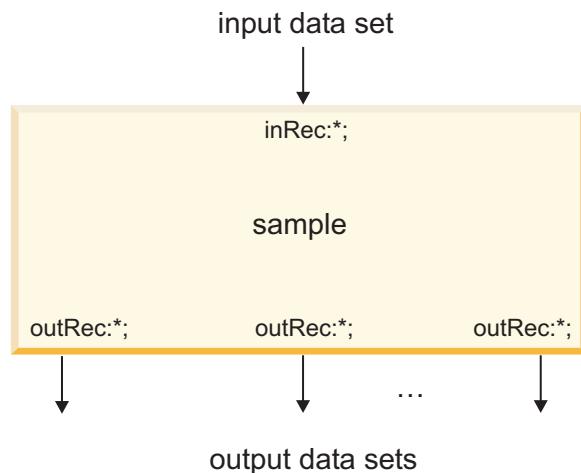
A record distribution is repeatable if you use the same:

- Seed value
- Number of output data sets
- Percentage of records assigned to each data set

No input record is assigned to more than one output data set. The sum of the percentages of all records assigned to the output data sets must be less than or equal to 100%

- Alternatively, you can specify that every *n*th record be written to output data set 0.

Data flow diagram



sample: properties

Table 49. sample properties

Property	Value
Number of input data sets	1
Number of output data sets	N (set by user)
Input interface schema	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec -> outRec without record modification
Execution mode	parallel (default) or sequential

Table 49. sample properties (continued)

Property	Value
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	no

Sample: syntax and options

```
sample
-percent percent output_port_num
[-percent percent output_port_num ... ]
|
-sample sample [-maxoutputrows maxout] [-seed seed_val ]
```

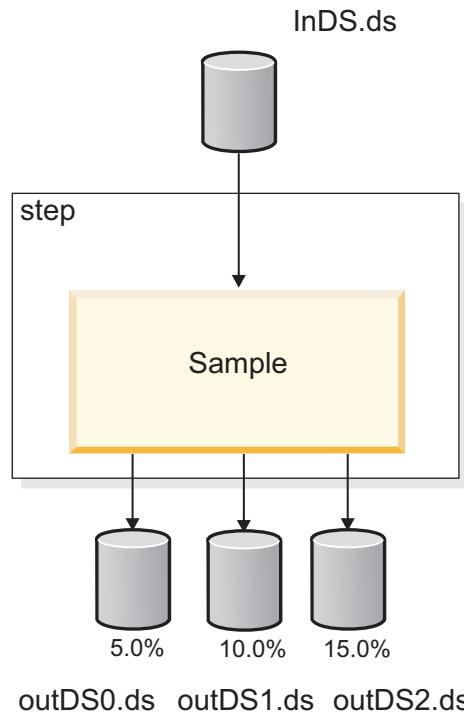
Either the -percent option must be specified for each output data set or the -sample option must be specified.

Table 50. Sample options

Option	Use
maxoutputrows	-maxoutputrows <i>maxout</i> Optionally specifies the maximum number of rows to be output per process. Supply an integer ≥ 1 for <i>maxout</i> .
-percent	-percent <i>percent output_port_num</i> Specifies the sampling percentage for each output data set. You specify the percentage as an integer value in the range of 0, corresponding to 0.0%, to 100, corresponding to 100.0%. The sum of the percentages specified for all output data sets cannot exceed 100.0%. The <i>output_port_num</i> following percent is the output data set number. The -percent and -sample options are mutually exclusive. One must be specified.
-sample	-sample <i>sample</i> Specifies that each <i>n</i> th record is written to output 0. Supply an integer ≥ 1 for <i>sample</i> to indicate the value for <i>n</i> . The -sample and -percent options are mutually exclusive. One must be specified.
-seed	-seed <i>seed_val</i> Initializes the random number generator used by the operator to randomly sample the records of the input data set. <i>seed_val</i> must be a 32-bit integer. The operator uses a repeatable random number generator, meaning that the record distribution is repeatable if you use the same <i>seed_val</i> , number of output data sets, and percentage of records assigned to each data set.

Example sampling of a data set

This example configures the sample operator to generate three output data sets from an input data set. The first data set receives 5.0% of the records of the input data set 0, data set 1 receives 10.0%, and data set 2 receives 15.0%.



Use this osh command to implement this example:

```
$ osh "sample -seed 304452 -percent 5 0 -percent 10 1 -percent 15 2 < inDS.ds > outDS0.ds > outDS1.ds > outDS2.ds"
```

In this example, you specify a seed value of 304452, a sampling percentage for each output data set, and three output data sets.

Sequence operator

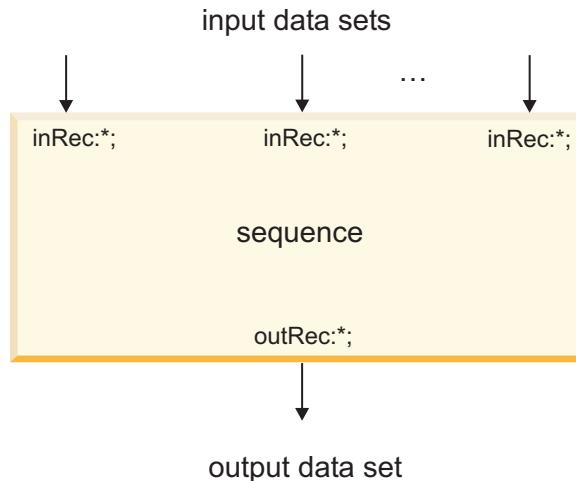
Using the sequence operator, you can copy multiple input data sets to a single output data set. The sequence operator copies all records from the first input data set to the output data set, then all the records from the second input data set, and so forth. This operation is useful when you want to combine separate data sets into a single large data set. This topic describes how to use the sequence operator.

The sequence operator takes one or more data sets as input and copies all input records to a single output data set. The operator copies all records from the first input data set to the output data set, then all the records from the second input data set, and so on. The record schema of all input data sets must be identical.

You can execute the sequence operator either in parallel (the default) or sequentially. Sequential mode allows you to specify a collection method for an input data set to control how the data set partitions are combined by the operator.

This operator differs from the funnel operator, described in "Funnel Operators" , in that the funnel operator does not guarantee the record order in the output data set.

Data flow diagram



sequence: properties

Table 51. sequence properties

Property	Value
Number of input data sets	N (set by user)
Number of output data sets	1
Input interface schema	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec -> outRec without record modification
Execution mode	parallel (default) or sequential
Partitioning method	round robin (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	no

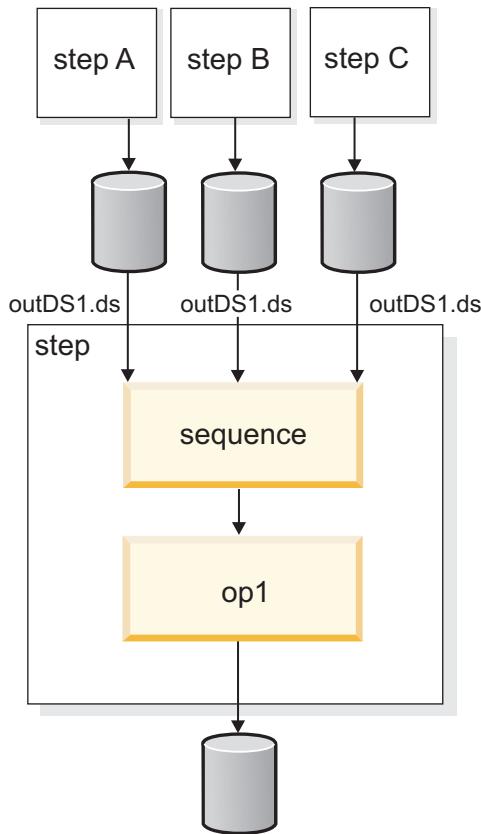
Sequence: syntax and options

The syntax for the sequence operator in an osh command is simply:
sequence

It has no operator-specific options.

Example of Using the sequence Operator

This example uses the sequence operator to combine multiple data sets created by multiple steps, before passing the combined data to another operator op1. The diagram shows data flow for this example:



The following osh commands create the data sets:

```
$ osh "... > outDS0.ds"
$ osh "... > outDS1.ds"
$ osh "... > outDS2.ds"
```

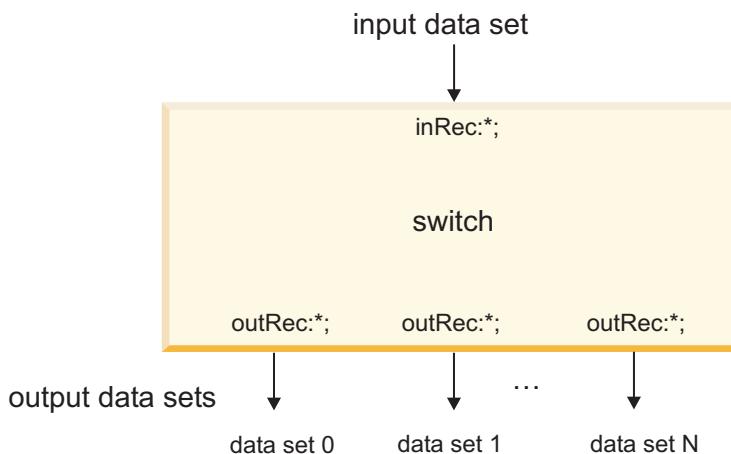
The osh command for the step beginning with the sequence operator is:

```
$ osh "sequence < outDS0.ds < outDS1.ds < outDS2.ds | op1 ... rees"
```

Switch operator

The switch operator takes a single data set as input. The input data set must have an integer field to be used as a selector, or a selector field whose values can be mapped, implicitly or explicitly, to int8. The switch operator assigns each input record to one of multiple output data sets based on the value of the selector field.

Data flow diagram



The switch operator is analogous to a C switch statement, which causes the flow of control in a C program to branch to one of several cases based on the value of a selector variable, as shown in the following C program fragment.

```

switch (selector)
{
    case 0: // if selector = 0,
              // write record to output data set 0
        break;

    case 1: // if selector = 1,
              // write record to output data set 1
        break;

    .
    .
    .
    case discard: // if selector = discard value
                   // skip record
        break;

    case default:// if selector is invalid,
                  // abort operator and end step
};


```

You can attach up to 128 output data sets to the switch operator corresponding to 128 different values for the selector field.

Note that the selector value for each record must be in or be mapped to the range 0 to $N-1$, where N is the number of data sets attached to the operator, or be equal to the discard value. Invalid selector values normally cause the switch operator to terminate and the step containing the operator to return an error. However, you might set an option that allows records whose selector field does not correspond to that range to be either dropped silently or treated as allowed rejects.

You can set a discard value for the selector field. Records whose selector field contains or is mapped to the discard value is dropped, that is, not assigned to any output data set.

switch: properties

Table 52. switch properties

Property	Value
Number of input data sets	1
Number of output data sets	$1 \leq N \leq 128$

Table 52. switch properties (continued)

Property	Value
Input interface schema	<i>selector field:any data type;inRec:*</i>
Output interface schema	<i>outRec:*</i>
Preserve-partitioning flag in output data set	propagated

Switch: syntax and options

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
switch
  [-allowRejects]
  |
  [-ifNotFound ignore | allow | fail]
  |
  [-ignoreRejects] | [-hashSelector]
  [-case "selector_value = output_ds"]
  [-collation_sequence locale | collation_file_pathname | OFF]
  [-discard discard_value]
  [-key field_name [-cs | -ci]]
  [-param params]]
```

If the selector field is of type integer and has no more than 128 values, there are no required options, otherwise you must specify a mapping using the -case option.

Table 53. switch options

Option	Use
-allowRejects	<p>-allowRejects</p> <p>Rejected records (whose selector value is not in the range 0 to N-1, where N is the number of data sets attached to the operator, or equal to the discard value) are assigned to the last output data set. This option is mutually exclusive with the -ignoreRejects, -ifNotFound, and -hashSelector options.</p>
-case	<p>-case <i>mapping</i></p> <p>Specifies the mapping between actual values of the selector field and the output data sets. <i>mapping</i> is a string of the form "<i>selector_value = output_ds</i>", where <i>output_ds</i> is the number of the output data set to which records with that selector value should be written (<i>output_ds</i> can be implicit, as shown in the example below). You must specify an individual mapping for each value of the selector field you want to direct to one of the output data sets, thus -case is invoked as many times as necessary to specify the complete mapping.</p> <p>Multi-byte Unicode character data is supported for ustring selector values.</p> <p>Note: This option is incompatible with the -hashSelector option.</p>

Table 53. switch options (continued)

Option	Use
-collation_sequence	<p><code>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</code></p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> • Specify a predefined IBM ICU locale • Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> • Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.htm</p>
-discard	<p><code>-discard <i>discard_value</i></code></p> <p>Specifies an integer value of the selector field, or the value to which it was mapped using case, that causes a record to be discarded by the operator.</p> <p>Note that <i>discard_value</i> must be outside the range 0 to <i>N</i>-1, where <i>N</i> is the number of data sets attached to the operator.</p> <p>Note: This option is mutually exclusive with -hashSelector.</p>
-hashSelector	<p><code>-hashSelector</code></p> <p>A boolean; when this is set, records are hashed on the selector field modulo the number of output data sets and assigned to an output data set accordingly. The selector field must be of a type that is convertible to uint32 and might not be nullable.</p> <p>Note: This option is incompatible with the -case, -discard, -allowRejects, -ignoreRejects, and -ifNotFound options.</p>

Table 53. switch options (continued)

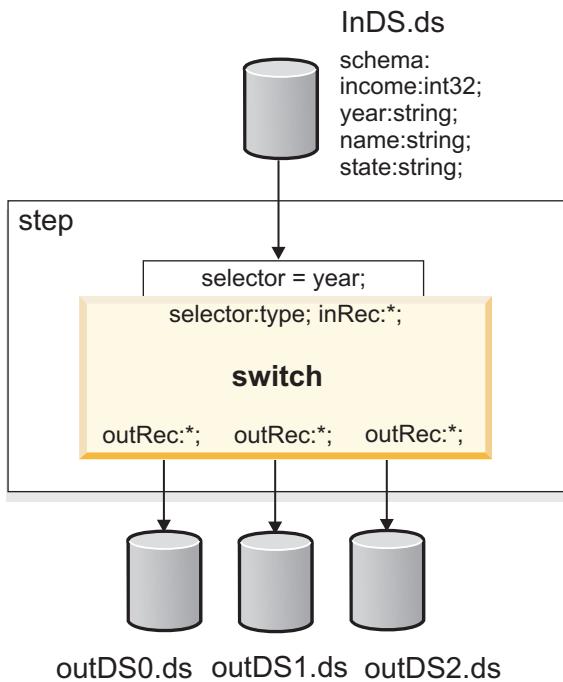
Option	Use
-ifNotFound	<p><code>-ifNotFound {allow fail ignore}</code></p> <p>Specifies what the operator should do if a data set corresponding to the selector value does not exist:</p> <ul style="list-style-type: none"> allow Rejected records (whose selector value is not in the range 0 to N-1 or equal to the discard value) are assigned to the last output data set. If this option value is used, you might not explicitly assign records to the last data set. fail When an invalid selector value is found, return an error and terminate. This is the default. ignore Drop the record containing the out-of-range value and continue. <p>Note. This option is incompatible with -allowRejects, -ignoreRejects, and -hashSelector options.</p>
-ignoreRejects	<p><code>-ignoreRejects</code></p> <p>Drop the record containing the out-of-range value and continue.</p> <p>Note. This option is mutually exclusive with the -allowRejects, -ifNotFound, and -hashSelector options.</p>
-key	<p><code>-key field_name [-cs -ci]</code></p> <p>Specifies the name of a field to be used as the selector field. The default field name is selector. This field can be of any data type that can be converted to int8, or any non-nullable type if case options are specified. Field names can contain multi-byte Unicode characters.</p> <p>Use the -ci flag to specify that field_name is case-insensitive. The -cs flag specifies that field_name is treated as case sensitive, which is the default.</p>

In this example, you create a switch operator and attach three output data sets numbered 0 through 2. The switch operator assigns input records to each output data set based on the selector field, whose year values have been mapped to the numbers 0 or 1 by means of the -case option. A selector field value that maps to an integer other than 0 or 1 causes the operator to write the record to the last data set. You might not explicitly assign input records to the last data set if the -ifNotFound option is set to allow.

With these settings, records whose year field has the value 1990, 1991, or 1992 go to outDS0.ds. Those whose year value is 1993 or 1994 go to outDS1.ds. Those whose year is 1995 are discarded. Those with any other year value are written to outDS2.ds, since rejects are allowed by the -ifNotFound setting. Note that because the -ifNotFound option is set to allow rejects, switch does not let you map any year value explicitly to the last data set (outDS2.ds), as that is where rejected records are written.

Note also that it was unnecessary to specify an output data set for 1991 or 1992, since without an explicit mapping indicated, case maps values across the output data sets, starting from the first (outDS0.ds). You might map more than one selector field value to a given output data set.

The operator also verifies that if a -case entry maps a selector field value to a number outside the range 0 to $N-1$, that number corresponds to the value of the -discard option.



In this example, all records with the selector field mapped to:

- 0 are written to outDS0.ds
- 1 is written to outDS1.ds
- 5 are discarded
- any other values are treated as rejects and written to outDS2.ds.

In most cases, your input data set does not contain an 8-bit integer field to be used as the selector; therefore, you use the -case option to map its actual values to the required range of integers. In this example, the record schema of the input data set contains a string field named year, which you must map to 0 or 1.

Specify the mapping with the following osh code:

```
$ osh "switch -case 1990=0
      -case 1991
      -case 1992
      -case 1993=1
      -case 1994=1
      -case 1995=5
      -discard 5
      -ifNotFound allow
      -key year < inDS.ds
      > outDS0.ds
      > outDS1.ds
      > outDS2.ds "
```

Note that by default output data sets are numbered starting from 0. You could also include explicit data set numbers, as shown below:

```
$ osh "switch -discard 3 < inDS.ds
      0> outDS0.ds
      1> outDS1.ds
      2> outDS2.ds "
```

Job monitoring information

The switch operator reports business logic information which can be used to make decisions about how to process data. It also reports summary statistics based on the business logic.

The business logic is included in the metadata messages generated by WebSphere DataStage as custom information. It is identified with:

```
name="BusinessLogic"
```

The output summary per criterion is included in the summary messages generated by WebSphere DataStage as custom information. It is identified with:

```
name="CriterionSummary"
```

The XML tags criterion, case and where are used by the switch operator when generating business logic and criterion summary custom information. These tags are used in the example information below.

Example metadata and summary messages

```
<response type="metadata">
  <component ident="switch">
    <componentstats startTime="2002-08-08 14:41:56"/>
    <linkstats portNum="0" portType="in"/>
    <linkstats portNum="0" portType="out"/>
    <linkstats portNum="1" portType="out"/>
    <linkstats portNum="2" portType="out"/>
    <custom_info Name="BusinessLogic" Desc="User-supplied logic to switch operator">
      <criterion name="key">tfield</criterion>
      <criterion name="case">
        <case value=" 0" output_port="0"></case>
        <case value=" 1" output_port="1"></case>
        <case value=" 2" output_port="2"></case>
      </criterion>
    </custom_info>
  </component>
</response>
<response type="summary">
  <component ident="switch" pid="2239">
    <componentstats startTime="2002-08-08 14:41:59" stopTime=
      "2002-08-08 14:42:40" percentCPU="99.5"/>
    <linkstats portNum="0" portType="in" recProcessed="1000000"/>
    <linkstats portNum="0" portType="out" recProcessed="250000"/>
    <linkstats portNum="1" portType="out" recProcessed="250000"/>
    <linkstats portNum="2" portType="out" recProcessed="250000"/>
    <custom_info Name="CriterionSummary" Desc=
      "Output summary per criterion">
      <case value=" 0" output_port="0" recProcessed="250000"/>
      <case value=" 1" output_port="1" recProcessed="250000"/>
      <case value=" 2" output_port="2" recProcessed="250000"/>
    </custom_info>
  </component>
</response>
```

Customizing job monitor messages

WebSphere DataStage specifies the business logic and criterion summary information for the switch operator using the functions addCustomMetadata() and addCustomSummary(). You can also use these functions to generate this kind of information for the operators you write.

Tail operator

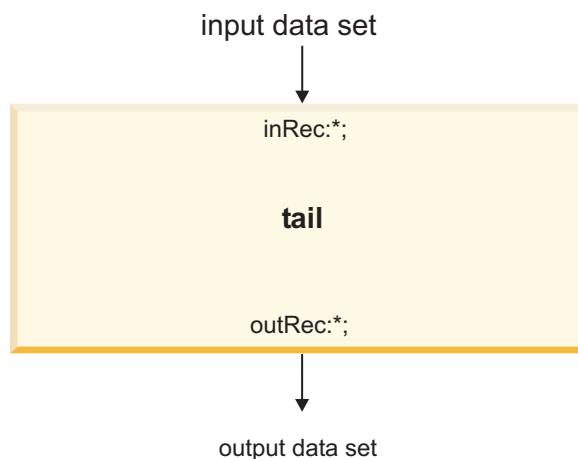
The tail operator copies the last N records from each partition of its input data set to its output data set. By default, N is 10 records. However, you can determine the following by means of options:

- The number of records to copy
- The partition from which the records are copied

This control is helpful in testing and debugging jobs with large data sets.

The head operator performs a similar operation, copying the first N records from each partition. See "Head Operator".

Data flow diagram



tail: properties

Table 54. tail properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec -> outRec without record modification

Tail: syntax and options

```
tail  
[-nrecs count] [-part partition_number]
```

Table 55. Tail options

Option	Use
-nrecs	<p>-nrecs count</p> <p>Specify the number of records (<i>count</i>) to copy from each partition of the input data set to the output data set. The default value of <i>count</i> is 10.</p>

Table 55. Tail options (continued)

Option	Use
-part	<p>-part <i>partition_number</i></p> <p>Copy records only from the indicated partition. By default, the operator copies records from all partitions. You can specify -part multiple times to specify multiple partition numbers. Each time you do, specify the option followed by the number of the partition.</p>

Tail example 1: tail operator default behavior

In this example, no options have been specified to the tail operator. The input data set consists of 60 sorted records (positive integers) hashed into four partitions. The output data set consists of the last ten records of each partition. The osh command for the example is:

```
$osh "tail < in.ds > out.ds"
```

Table 56. tail Operator Input and Output for Example 1

Partition 0		Partition 1		Partition 2		Partition 3	
Input	Output	Input	Output	Input	Output	Input	Output
0 9 18 19 23	9 18 19 23	3 5 11 12 13	16 17 35 42	6 7 8 22 29	30 33 41 43	1 2 4 10 20	26 27 28 31
25 36 37 40	25 36 37 40	14 15 16 17	46 49 50 53	30 33 41 43	44 45 48 55	21 24 26 27	32 34 38 39
47 51	47 51	35 42 46 49	57 59	44 45 48 55	56 58	28 31 32 34	52 54
		50 53 57 59		56 58		38 39 52 54	

Example 2: tail operator with both options

In this example, both the -nrecs and -part options are specified to the tail operator to request that the last 3 records of Partition 2 be output. The input data set consists of 60 sorted records (positive integers) hashed into four partitions. The output data set contains only the last three records of Partition 2. Table 57 shows the input and output data.

The osh command for this example is:

```
$ osh "tail -nrecs 3 -part 2 < in.ds > out0.ds"
```

Table 57. tail Operator Input and Output for Example 2

Partition 0		Partition 1		Partition 2		Partition 3	
Input	Output	Input	Output	Input	Output	Input	Output
0 9 18 19 23		3 5 11 12 13		6 7 8 22 29	55 56 58	1 2 4 10 20	
25 36 37 40		14 15 16 17		30 33 41 43		21 24 26 27	
47 51		35 42 46 49		44 45 48 55		28 31 32 34	
		50 53 57 59		56 58		38 39 52 54	

Transform operator

The transform operator modifies your input records, or transfers them unchanged, guided by the logic of the transformation expression you supply. You build transformation expressions using the Transformation Language, which is the language that defines expression syntax and provides built-in functions.

By using the Transformation Language with the transform operator, you can:

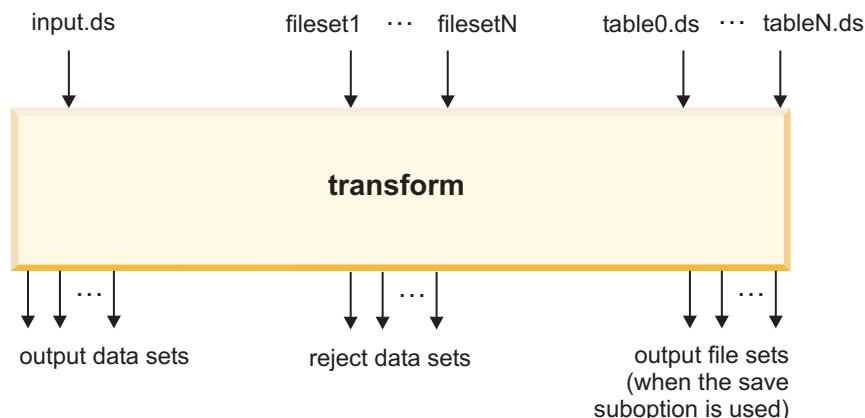
- Transfer input records to one or more outputs
- Define output fields and assign values to them based on your job logic
- Use local variables and input and output fields to perform arithmetic operations, make function calls, and construct conditional statements
- Control record processing behavior by explicitly calling `writerecord dataset_number`, `droprecord`, and `rejectrecord` as part of a conditional statement

Running your job on a non-NFS MPP

At run time, the transform operator distributes its shared library to remote nodes on non-NFS MPP systems. To prevent your job from aborting, these three conditions must be satisfied:

1. The `APT_COPY_TRANSFORM_OPERATOR` environment variable must be set.
2. Users must have create privileges on the project directory paths on all remote nodes at runtime. For example, the transform library `trx.so` is created on the conductor node at this location:
`/opt/IBM/InformationServer/Server/Projects/simple/RT_BP1.O`
3. Rename `$APT_ORCHHOME/etc/distribute-component.example` to `$APT_ORCHHOME/etc/distribute-component` and make the file executable:
`chmod 755 $APT_ORCHHOME/etc/distribute-component`

Data flow diagram



transform: properties

Table 58. *transform* properties

Property	Value
Number of input data sets	1 plus the number of lookup tables specified on the command line.
Number of output data sets	1 or more and, optionally, 1 or more reject data sets
Transfer behavior	See "Transfer Behavior"

Table 58. transform properties (continued)

Property	Value
Execution mode	parallel by default, or sequential
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	propagated
Composite operator	yes
Combinable operator	yes

Transform: syntax and options

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
transform
-fileset fileset_description
-table -key field [ci | cs]
[-key field [ci | cs] ...]
[-allow_dups]
[-save fileset_descriptor]
[-diskpool pool]
[-schema schema | -schemafile schema_file]
[-argvalue job_parameter_name=job_parameter_value ...] [-collation_sequence locale |
collation_file_pathname | OFF]
[-expression expression_string | -expressionfile expressionfile_path ]
[-maxrejectlogs integer]
[-sort [-input | -output [ port ] -key field_name
sort_key_suboptions ...]
[-part [-input | -output [port] -key field_name part_key_suboptions ...]
[-flag {compile | run | compileAndRun} [ flag_compilation_options ]]
[-inputschema schema | -inputschemafайл schema_file ]
[-outputschema schema | -outputschemafайл schema_file ]
[-reject [-rejectinfo reject_info_column_name_string]]
```

Where:

sort_key_suboptions are:

```
[-ci | -cs] [-asc | -desc] [-nulls {first | last}] [-param params ]
```

part_key_options are:

```
[-ci | -cs] [-param params ]
```

flag_compilation_options are:

```
[-dir dir_name_for_compilation] [-name library_path_name ]
[-optimize | -debug] [-verbose] [-compiler cpath ]
[-staticobj absolute_path_name] [-sharedobj absolute_path_name] [-t options ]
[compileopt options] [-linker lpath] [-linkopt options ]
```

The -table and -fileset options allow you to use conditional lookups.

Note: The following option values can contain multi-byte Unicode values:

- the field names given to the -inputschema and -outputschema options and the ustring values
- -inputschemafайл and -outputschemafайл files
- -expression option string and the -expressionfile option filepath
- -sort and -part key-field names

- -compiler, -linker, and -dir pathnames
- -name file name
- -staticobj and -sharedobj pathnames
- -compileopt and -linkopt pathnames

Option	Use
-argvalue	<p>-argvalue <i>job_parameter_name</i> = <i>job_parameter_value</i></p> <p>This option is similar to the -params top-level osh option, but the initialized variables apply to a transform operator rather than to an entire job. The global variable given by <i>job_parameter_name</i> is initialized with the value given by <i>job_parameter_value</i>.</p> <p>In your osh script, you reference the <i>job_parameter_value</i> with [& <i>job_parameter_name</i>] where the <i>job_parameter_value</i> component replaces the occurrence of [& <i>job_parameter_name</i>].</p>
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> • Specify a predefined IBM ICU locale • Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> • Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.htm</p>
-expression	<p>-expression <i>expression_string</i></p> <p>This option lets you specify expressions written in the Transformation Language. The expression string might contain multi-byte Unicode characters.</p> <p>Unless you choose the -flag option with run, you must use either the -expression or -expressionfile option.</p> <p>The -expression and -expressionfile options are mutually exclusive.</p>

Option	Use
-expressionfile	<p data-bbox="833 228 1144 259">-expressionfile <i>expression_file</i></p> <p data-bbox="833 283 1454 508">This option lets you specify expressions written in the Transformation Language. The expression must reside in an <i>expression_file</i>, which includes the name and path to the file which might include multi-byte Unicode characters. Use an absolute path, or by default the current UNIX directory. Unless you choose the -flag option with run, you must choose either the -expression or -expressionfile option.</p> <p data-bbox="833 532 1454 585">The -expressionfile and -expression options are mutually exclusive.</p>

Option	Use
<p>-flag</p>	<p>-flag {compile run compileAndRun} <i>suboptions</i></p> <p>compile: This option indicates that you wish to check the Transformation Language expression for correctness, and compile it. An appropriate version of a C++ compiler must be installed on your computer. Field information used in the expression must be known at compile time; therefore, input and output schema must be specified.</p> <p>run: This option indicates that you wish to use a pre-compiled version of the Transformation Language code. You do not need to specify input and output schemas or an expression because these elements have been supplied at compile time. However, you must add the directory containing the pre-compiled library to your library search path. This is not done by the transform operator. You must also use the -name suboption to provide the name of the library where the pre-compiled code resides.</p> <p>compileAndRun: This option indicates that you wish to compile and run the Transformation Language expression. This is the default value. An appropriate version of a C++ compiler must be installed on your computer.</p> <p>You can supply schema information in the following ways:</p> <ul style="list-style-type: none"> • You can omit all schema specifications. The transform operator then uses the up-stream operator's output schema as its input schema, and the schema for each output data set contains all the fields from the input record plus any new fields you create for a data set. • You can omit the input data set schema, but specify schemas for all output data sets or for selected data sets. The transform operator then uses the up-stream operator's output schema as its input schema. Any output schemas specified on the command line are used unchanged, and output data sets without schemas contain all the fields from the input record plus any new fields you create for a data set. • You can specify an input schema, but omit all output schemas or omit some output schemas. The transform operator then uses the input schema as specified. Any output schemas specified on the command line are used unchanged, and output data sets without schemas contain all the fields from the input record plus any new fields you create for a data set.

Option	Use
-flag (continued)	<ul style="list-style-type: none"> • The flag option has the following suboptions: <ul style="list-style-type: none"> -dir <i>dir_name</i> lets you specify a compilation directory. By default, compilation occurs in the TMPDIR directory or, if this environment variable does not point to an existing directory, to the /tmp directory. Whether you specify it or not, you must make sure the directory for compilation is in the library search path. -name <i>file_name</i> lets you specify the name of the file containing the compiled code. If you use the -dir <i>dir_name</i> suboption, this file is in the <i>dir_name</i> directory. • The following examples show how to use the -dir and -name options in an osh command line: <p>For development:</p> <pre>osh "transform -inputschema schema -outputschema schema -expression expression -flag compile - dir dir_name -name file_name "</pre> <p>For your production machine:</p> <pre>osh "... transform -flag run -name file_name ..."</pre> <p>The library file must be copied to the production machine.</p> <p>-flag compile and -flag compileAndRun have these additional suboptions:</p> <ul style="list-style-type: none"> -optimize specifies the optimize mode for compilation. -debug specifies the debug mode for compilation. • -verbose causes verbose messages to be output during compilation. -compiler <i>cpath</i> lets you specify the compiler path when the compiler is not in the default directory. The default compiler path for each operating system is: <ul style="list-style-type: none"> Solaris: /opt/SUNPRO6/SUNWspro/bin/CC AIX: /usr/vacpp/bin/xlC_r Tru64: /bin/cxx HP-UX: /opt/aCC/bin/aCC -staticobj <i>absolute_path_name</i> -sharedobj <i>absolute_path_name</i> These two suboptions specify the location of your static and dynamic-linking C-object libraries. The file suffix can be omitted. See "External Global C-Function Support" for details. -compileopt <i>options</i> lets you specify additional compiler options. These options are compiler-dependent. Pathnames might contain multi-byte Unicode characters. -linker <i>lpath</i> lets you specify the linker path when the linker is not in the default directory. The default linker path of each operating system is the same as the default compiler path listed above. -linkopt options lets you specify link options to the compiler. Pathnames might contain multi-byte Unicode characters.

Option	Use
-inputschema	<p>-inputschema <i>schema</i></p> <p>Use this option to specify an input schema. The schema might contain multi-byte Unicode characters. An error occurs if an expression refers to an input field not in the input schema.</p> <p>The -inputschema and the -inputschemafile options are mutually exclusive.</p> <p>The -inputschema option is not required when you specify <code>compileAndRun</code> or <code>run</code> for the -flag option; however, when you specify <code>compile</code> for the -flag option, you must include either the -inputschema or the -inputschemafile option. See the -flag option description in this table for information on the -compile suboption.</p>
-inputschemafile	<p>-inputschemafile <i>schema_file</i></p> <p>Use this option to specify an input schema. An error occurs if an expression refers to an input field not in the input schema. To use this option, the input schema must reside in a <i>schema_file</i>, where <i>schema_file</i> is the name and path to the file which might contain multi-byte Unicode characters. You can use an absolute path, or by default the current UNIX directory.</p> <p>The -inputschemafile and the -inputschema options are mutually exclusive.</p> <p>The -inputschemafile option is not required when you specify <code>compileAndRun</code> or <code>run</code> for the -flag option; however, when you specify <code>compile</code> for the -flag option, you must include either the -inputschema or the -inputschemafile option. See the -flag option description in this table for information on the -compile suboption.</p>
-maxrejectlogs	<p>-maxrejectlogs <i>integer</i></p> <p>An information log is generated every time a record is written to the reject output data set. Use this option to specify the maximum number of output reject logs the transform option generates. The default is 50. When you specify -1 to this option, an unlimited number of information logs are generated.</p>

Option	Use
<code>-outputschema</code>	<p><code>-outputschema schema</code></p> <p>Use this option to specify an output schema. An error occurs if an expression refers to an output field not in the output schema.</p> <p>The <code>-outputschema</code> and <code>-outputschemaf</code> options are mutually exclusive.</p> <p>The <code>-outputschema</code> option is not required when you specify <code>compileAndRun</code> or <code>run</code> for the <code>-f</code> option; however, when you specify <code>compile</code> for the <code>-f</code> option, you must include either the <code>-outputschema</code> or the <code>-outputschemaf</code> option. See the <code>-f</code> option description in this table for information on the <code>-compile</code> suboption. For multiple output data sets, repeat the <code>-outputschema</code> or <code>-outputschemaf</code> option to specify the schema for all output data sets.</p>
<code>-outputschemaf</code>	<p><code>-outputschemaf schema_file</code></p> <p>Use this option to specify an output schema. An error occurs if an expression refers to an output field not in the output schema. To use this option, the output schema must reside in a <i>schema_file</i> which includes the name and path to the file. You can use an absolute path, or by default the current UNIX directory.</p> <p>The <code>-outputschemaf</code> and the <code>-outputschema</code> options are mutually exclusive.</p> <p>The <code>-outputschemaf</code> option is not required when you specify <code>compileAndRun</code> or <code>run</code> for the <code>-f</code> option; however, when you specify <code>compile</code> for the <code>-f</code> option, you must include either the <code>-outputschema</code> or the <code>-outputschemaf</code> option. See the <code>-f</code> option description in this table for information on the <code>-compile</code> suboption. For multiple output data sets, repeat the <code>-outputschema</code> or <code>-outputschemaf</code> option to specify the schema for all output data sets.</p>
<code>-part</code>	<p><code>-part {-input -output[port]} -key field_name [-ci -cs] [-param params]</code></p> <p>You can use this option 0 or more times. It indicates that the data is hash partitioned. The required <code>field_name</code> is the name of a partitioning key.</p> <p>Exactly one of the suboptions <code>-input</code> and <code>-output[port]</code> must be present. These suboptions determine whether partitioning occurs on the input data or the output data. The default for <code>port</code> is 0. If <code>port</code> is specified, it must be an integer which represents an output data set where the data is partitioned.</p> <p>The suboptions to the <code>-key</code> option are <code>-ci</code> for case-insensitive partitioning, or <code>-cs</code> for a case-sensitive partitioning. The default is case-sensitive. The <code>-params</code> suboption is to specify any <i>property=value</i> pairs. Separate the pairs by commas (,).</p>

Option	Use
-reject	<p>-reject [-rejectinfo <i>reject_info_column_name_string</i>]</p> <p>This is optional. You can use it only once.</p> <p>When a null field is used in an expression, this option specifies that the input record containing the field is not dropped, but is sent to the output reject data set.</p> <p>The -rejectinfo suboption specifies the column name for the reject information.</p>
-sort	<p>-sort {-input -output [<i>port</i>]} -key <i>field_name</i> [-ci -cs] [-asc -desc] [-nulls {first last}] [-param <i>params</i>]</p> <p>You can use this option 0 or more times. It indicates that the data is sorted for each partition. The required <i>field_name</i> is the name of a sorting key.</p> <p>Exactly one of the suboptions -input and -output[<i>port</i>] must be present. These suboptions determine whether sorting occurs on the input data or the output data. The default for <i>port</i> is 0. If <i>port</i> is specified, it must be an integer that represents the output data set where the data is sorted.</p> <p>You can specify -ci for a case-insensitive sort, or -cs for a case-sensitive sort. The default is case-sensitive.</p> <p>You can specify -asc for an ascending order sort or -desc for a descending order sort. The default is ascending.</p> <p>You can specify -nulls {first last} to determine where null values should sort. The default is that nulls sort first.</p> <p>You can use -param <i>params</i> to specify any <i>property = value</i> pairs. Separate the pairs by commas (,).</p>

Option	Use
-table	<p>-table -key <i>field</i> [ci cs] [-key <i>field</i> [ci cs] ...] [-allow_dups] [-save <i>fileset_descriptor</i>] [-diskpool <i>pool</i>] [-schema <i>schema</i> -schemafile <i>schema_file</i>]</p> <p>Specifies the beginning of a list of key fields and other specifications for a lookup table. The first occurrence of -table marks the beginning of the key field list for lookup table1; the next occurrence of -table marks the beginning of the key fields for lookup table2, and so on. For example:</p> <pre>lookup -table -key field -table -key field</pre> <p>The -key option specifies the name of a lookup key field. The -key option must be repeated if there are multiple key fields. You must specify at least one key for each table. You cannot use a vector, subrecord, or tagged aggregate field as a lookup key.</p> <p>The -ci suboption specifies that the string comparison of lookup key values is to be case insensitive; the -cs option specifies case-sensitive comparison, which is the default.</p> <p>In create-only mode, the -allow_dups option causes the operator to save multiple copies of duplicate records in the lookup table without issuing a warning. Two lookup records are duplicates when all lookup key fields have the same value in the two records. If you do not specify this option, WebSphere DataStage issues a warning message when it encounters duplicate records and discards all but the first of the matching records.</p> <p>In normal lookup mode, only one lookup table (specified by either -table or -fileset) can have been created with -allow_dups set.</p> <p>The -save option lets you specify the name of a fileset to write this lookup table to; if -save is omitted, tables are written as scratch files and deleted at the end of the lookup. In create-only mode, -save is, of course, required.</p> <p>The -diskpool option lets you specify a disk pool in which to create lookup tables. By default, the operator looks first for a "lookup" disk pool, then uses the default pool (""). Use this option to specify a different disk pool to use.</p> <p>The -schema suboption specifies the schema that interprets the contents of the string or raw fields by converting them to another data type. The -schemafile suboption specifies the name of a file containing the schema that interprets the content of the string or raw fields by converting them to another data type. You must specify either -schema or -schemafile. One of them is required if the -compile option is set, but are not required for -compileAndRun or -run.</p>

Option	Use
-fileset	<p>[<code>-fileset fileset_descriptor ...</code>]</p> <p>Specify the name of a fileset containing one or more lookup tables to be matched.</p> <p>In lookup mode, you must specify either the <code>-fileset</code> option, or a table specification, or both, in order to designate the lookup table(s) to be matched against. There can be zero or more occurrences of the <code>-fileset</code> option. It cannot be specified in create-only mode.</p> <p>Warning: The fileset already contains key specifications. When you follow <code>-fileset fileset_descriptor</code> by <code>key_specifications</code>, the keys specified do not apply to the fileset; rather, they apply to the first lookup table. For example, <code>lookup -fileset file -key field</code>, is the same as:</p> <p><code>lookup -fileset file1 -table -key field</code></p>

Transfer behavior

You can transfer your input fields to your output fields using any one of the following methods:

- Set the value of the `-flag` option to `compileAndRun`. For example:

```
osh "..." | transform -expression expression
-flacompileAndRun
-dir dir_name
-name file_name | ..."
```
- Use schema variables as part of the schema specification. A partial schema might be used for both the input and output schemas.

This example shows a partial schema in the output:

```
osh "transform
    -expression expression
    -inputschema record(a:int32;b:string[5];c:time)
    -outputschema record(d:dfloat:outRec:*)
    -flag compile ..."
```

where the schema for output 0 is:

```
record(d:dfloat;a:int32;b:string[5];c:time)
```

This example shows partial schemas in the input and the output:

```
osh "transform
    -expression expression
    -inputschema record(a:int32;b:string[5];c:time;Inrec:*)
    -outputschema record(d:dfloat:outRec:*)
    -flag compile ..."
osh "... | transform -flag run ... | ..."
```

Output 0 contains the fields d, a, b, and c, plus any fields propagated from the up-stream operator.

- Use name matching between input and output fields in the schema specification. When input and output field names match and no assignment is made to the output field, the input field is transferred to the output data set unchanged. Any input field which doesn't have a corresponding output field is dropped. For example:

```
osh "transform
    -expression expression
    -inputschema record(a:int32;b:string[5];c:time)
    -outputschema record(a:int32;)
    -outputschema record(a:int32;b:string[5];c:time)
    -flag compile ..."
```

Field a is transferred from input to output 0 and output 1. Fields b and c are dropped in output 0, but are transferred from input to output 1.

- Specify a reject data set. In the Transformation Language, it is generally illegal to use a null field in expressions except in the following cases:

- In function calls to `notnull(field_name)` and `null(fieldname)`
- In an assignment statement of the form `a=b` where a and b are both nullable and b is null
- In these expressions:

```
if (null(a))
  b=a
else
  b=a+1
if (notnull(a))
  b=a+1
else
  b=a
b=null (a)?a:a +1;
b=notnull(a)?a+1:a;
```

If a null field is used in an expression in other than these cases and a reject set is specified, the whole input record is transferred to the reject data set.

The transformation language

The Transformation Language is a subset of C, with extensions specific to dealing with records.

General structure

As in C, statements must be terminated by semi-colons and compound statements must be grouped by braces. Both C and C++ style comments are allowed.

Names and keywords

Names of fields in records, local variable names, and language keywords can consist of alphanumeric characters plus the underscore character. They cannot begin with a numeric character. Names in the Transformation Language are case-sensitive but keywords are case-insensitive.

The available keywords fall into five groups:

- The keyword `extern` is used to declare global C functions. See "External Global C-Function Support" below.
- The keywords `global`, `initialize`, `mainloop`, and `finish` mark blocks of code that are executed at different stages of record processing. An explanation of these keywords are in "Code Segmentation Keywords".
- The keywords `droprecord`, `writerecord`, and `rejectrecord` control record processing. See "Record Processing Control".
- The keywords `inputname` and `outputname` are used to declare data set aliases. See "Specifying Data Sets".
- The `tablename` keyword is used to identify lookup tables by name. See "Specifying Lookup Tables".

External global C-function support

Standard C functions are supported in the Transformation Language. Declare them in your expression file using the `extern` keyword and place them before your code segmentation keywords. The syntax for an external C function declaration is:

```
extern return_type function_name ([ argument_type , argument_name ...]);
```

Here is an expression file fragment that incorporates external C-function declarations:

```

// externs this C function: int my_times(int x, int y) { ... }
extern int32 my_times(int32 x, int32 y);
// externs this C function: void my_print_message(char *msg) { ... }
extern void my_print_message(string msg);
inputname 0 in0;
outputname 0 out0;
mainloop
{ ... }

```

C function schema types and associated C types

The C function return and argument types can be any of the WebSphere DataStage schema types listed below with their associated C types.

Schema Type	Associated Native C Type
int8	signed char
uint8	unsigned char
int16	short
uint16	unsigned short
int32	int
uint32	unsigned int
int64	long long for Solaris and AIX
uint64	unsigned long long for Solaris and AIX
sfloat	float
dfloat	double
string	char *
void	void

Specifying the location of your C libraries

To specify the locations of your static and dynamically-linked libraries, use the `-staticobj` and `-sharedobj` suboptions of the `-flag` option. These two suboptions take absolute path names as values. The file suffix is optional.

The syntax is:

```
-staticobj absolute_path_name -sharedobj absolute_path_name
```

An example static library specification is:

```
-flag compile
  -name generate_statistics
  -staticobj /external_functions/static/part_statistics.o
```

An example dynamic library specification is:

```
-flag compile
  ...
  -sharedobj /external_functions/dynamic/generate
```

The shared object file name has `lib` prepended to it and has a platform-dependent object-file suffix: `.so` for Sun Solaris and Linux; `.sl` for HP-UX, and `.o` for AIX. The file must reside in this directory:
`/external-functions/dynamic`

For this example, the object filepath on Solaris is:

/external-functions/dynamic/libgenerate.so

Dynamically-linked libraries must be manually deployed to all running nodes. Add the library-file locations to your library search path.

See "Example 8: External C Function Calls" for an example job that includes C header and source files, a Transformation Language expression file with calls to external C functions, and an osh script.

Code segmentation keywords

The Transformation Language provides keywords to specify when code is executed. Refer to "Example 1: Student-Score Distribution" for an example of how to use of these keywords.

- `global {job_parameters}`

Use this syntax to declare a set of global variables whose values are supplied by osh parameters. Values cannot be set with the Transformation Language. A warning message is issued if a value is missing.

- `initialize {statements}`

Use this syntax to mark a section of code you want to be executed once before the main record loop starts. Global variables whose values are not given through osh parameters should be defined in this segment.

- `mainloop {statements}`

Use this syntax to indicate the main record loop code. The `mainloop` segments is executed once for each input record.

- `finish {statements}`

Use this syntax to mark a section of code you want to be executed once after the main record loop terminates.

Record processing control

The transform operator processes one input record at a time, generating zero or any number of output records and zero or one reject record for each input record, terminating when there are no more input records to process.

The transform operator automatically reads records by default. You do not need to specify this actions.

The Transformation Language lets you control the input and output of records with the following keywords.

- `writerecord n;`

Use this syntax to force an output record to be written to the specific data set whose port number is n.

- `droprecord;`

Use this syntax to prevent the current input record from being written.

- `rejectcord;`

If you declare a reject data set, you can use this syntax to direct the current input record to it. You should only send a record to the reject data set if it is not going to another output data set.

Note: Processing exceptions, such as null values for non-nullable fields, cause a record to be written to the reject data set if you have specified one. Otherwise the record is simply dropped.

Specifying lookup tables

You specify a lookup table using the tablename keyword. This name corresponds to a lookup table object of the same name. A lookup table can be from an input to the operator or from a fileset, therefore, the order of parameters in the command line is be used to determine the number associated with the table.

The name of any field in the lookup schema, other than key fields, can be used to access the field value, such as table1.field1. If a field is accessed when `is_match()` returns false, the value of the field is null if it is nullable or it has its default value.

Here is an example of lookup table usage:

```
transform -expressionfile trx1 -table -key a -fileset sKeyTable.fs < dataset.v < table.v > target.v
trx1:
inputname 0 in1;
outputname 0 out0;
tablename 0 tbl1;
tablename 1 sKeyTable;
mainloop
{
// This code demonstrates the interface without doing anything really // useful
int nullCount;
nullCount = 0;
lookup(sKeyTable);
if (is_match(sKeyTable)) // if there's no match
{
    lookup(tbl1);
    if (!is_match(tbl1))
    {
        out0.field2 = "missing";
    }
}
else
{
    // Loop through the results
    while (is_match(sKeyTable))
    {
        if (is_null(sKeyTable.field1))
        {
            nullCount++;
        }
        next_match(sKeyTable);
    }
}
writerecord 0;
}
```

Specifying data sets

By default, the transform operator supports a single input data set, one or more output data sets, and a single optional reject data set. There is no default correspondence between input and output. You must use `writerecord port` to specify where you want your output sent. You can assign a name to each data set for unambiguous reference, using this syntax:

```
inputname 0 input-dataset-name; outputname n output-dataset-name;
```

Because the transform operator accepts only a single input data set, the data set number for `inputname` is 0. You can specify 0 through (the number of output data sets - 1) for the `outputname` data set number. For example:

```
inputname 0 input-grades;
outputname 0 output-low-grades;
outputname 1 output-high-grades;
```

Data set numbers cannot be used to qualify field names. You must use the `inputname` and `outputname` data set names to qualify field names in your Transformation Language expressions. For example:

```
output-low-grades.field-a = input-grades.field-a + 10;
output-high-grades.field-a = output-low-grades.field-a - 10;
```

Field names that are not qualified by a data set name always default to output data set 0. It is good practice to use the `inputname` data set name to qualify input fields in expressions, and use the

`outputname` data set name to qualify output fields even though these fields have unique names among all data sets. The Transformation Language does not attempt to determine if an unqualified, but unique, name exists in another data set.

The `inputname` and `outputname` statements must appear first in your Transformation Language code. For an example, see the Transformation Language section of "Example 2: Student-Score Distribution With a Letter Grade Added to Example 1".

Data types and record fields

The Transformation Language supports all legal WebSphere DataStage schemas and all record types. The table lists the simple field types. The complex field types follow that table.

Input and output fields can only be defined within the input/output schemas. You must define them using the operator options, not through transformation expressions. Refer to "Syntax and Options" for the details of the transform operator options.

You can reference input and output data set fields by name. Use the normal WebSphere DataStage dot notation (for example, `s.field1`) for references to subrecord fields. Note that language keywords are not reserved, so field names can be the same as keywords if they are qualified by data set names, `in0.fielda`.

Fields might appear in expressions. Fields that appear on the left side of an assignment statement must be output fields. New values might not be assigned to input fields.

The fieldtype, or data type of a field, can be any legal WebSphere DataStage data type. Fieldtypes can be simple or complex. The table lists the simple field types. The complex field types follow.

Data Type	Forms	Meaning
integer	<code>int8, int16, int32, int64</code>	1, 2, 4, and 8-byte signed integers
	<code>uint8, uint16, uint32, uint64</code>	1, 2, 4, and 8-byte unsigned integers
floating Point	<code>sfloat</code>	Single-precision floating point
	<code>dfloat</code>	Double-precision floating point
string	<code>string</code>	Variable-length string
	<code>string [max=n_codepoint_units]</code>	Variable-length string with upper bound on length
	<code>string[n_codepoint_units]</code>	Fixed-length string
wstring	<code>wstring</code>	
	<code>wstring [max=n_codepoint_units]</code>	
	<code>wstring[n_codepoint_units]</code>	
decimal	<code>decimal[p]</code>	Decimal value with p (precision) digits. p must be between 1 and 255 inclusive.
	<code>decimal[p, s]</code>	Decimal value with p digits and s (scale) digits to the right of the decimal point. p must be between 1 and 255 inclusive, and s must be between 0 and p inclusive.

Data Type	Forms	Meaning
date and time	date	Date with year, month, and day
	time	Time with one second resolution
	time[microseconds]	Time with one microsecond resolution
	timestamp	Date/time with one second resolution
	timestamp[microseconds]	Date/time with one microsecond resolution
raw	raw	Variable length binary data.
	raw[max=n]	Variable length binary data with at most n bytes.
	raw[n]	Fixed length (n -byte) binary data.
	raw[align=k]	Variable length binary data, aligned on k -byte boundary ($k = 1, 2, 4$, or 8)
	raw[max=n, align=k]	Variable length binary data with at most n bytes, aligned on k -byte boundary ($k = 1, 2, 4$, or 8)
	raw[n, align=k]	Fixed length (n -byte) binary data, aligned on k -byte boundary ($k = 1, 2, 4$, or 8)

WebSphere DataStage supports the following complex field types:

- vector fields
- subrecord fields
- tagged fields.

Note: Tagged fields cannot be used in expressions; they can only be transferred from input to output.

Local variables

Local variables are used for storage apart from input and output records. You must declare and initialize them before use within your transformation expressions.

The scope of local variables differs depending on which code segment defines them:

- Local variables defined within the global and initialize code segments can be accessed before, during, and after record processing.
- Local variables defined in the mainloop code segment are only accessible for the current record being processed.
- Local variables defined in the finish code segment are only accessible after all records have been processed.
- Local variables can represent any of the simple value types:
 - int8, uint8, int16, uint16, int32, uint32, int64, uint64
 - sfloat, dfloat
 - decimal
 - string
 - date, time, timestamp
 - raw

Declarations are similar to C, as in the following examples:

- `int32 a[100];` declares `a` to be an array of 100 32-bit integers
- `dfloat b;` declares `b` to be an double-precision float
- `string c;` declares `c` to be a variable-length string
- `string[n] e;` declares `e` to be a string of length n
- `string[n] ff[m];` declares `f` to be an array of m strings, each of length n
- `decimal[p] g;` declares `g` to be a decimal value with p (*precision*) digits
- `decimal[p, s] h;` declares `h` to be a decimal value with p (*precision*) digits and s (*scale*) digits to the right of the decimal

You cannot initialize variables as part of the declaration. They can only be initialized on a separate line. For example:

```
int32 a;  
a = 0;
```

The result is uncertain if a local variable is used without being initialized.

There are no local variable pointers or structures, but you can use arrays.

Expressions

The Transformation Language supports standard C expressions, with the usual operator precedence and use of parentheses for grouping. It also supports field names as described in "Data types and record fields", where the field name is specified in the schema for the data set.

Language elements

The Transformation Language supports the following elements:

- Integer, character, floating point, and string constants
- Local variables
- Field names
- Arithmetic operators
- Function calls
- Flow control
- Record processing control
- Code segmentation
- Data set name specification

Note that there are no date, time, or timestamp constants.

operators

The Transformation Language supports several unary operators, which all apply only to simple value types.

Symbol	Name	Applies to	Comments
<code>~</code>	One's complement	Integer	<code>~a</code> returns an integer with the value of each bit reversed
<code>!</code>	Complement	Integer	<code>!a</code> returns 1 if <code>a</code> is 0; otherwise returns 0
<code>+</code>	Unary plus	Numeric	<code>+a</code> returns <code>a</code>

Symbol	Name	Applies to	Comments
-	Unary minus	Numeric	-a returns the negative of a
++	Incrementation operator	Integer	a++ or ++a returns a + 1
--	Decrementation operator	Integer	a-- or --a returns a - 1.

The Transformation Language supports a number of binary operators, and one ternary operator.

Symbol	Name	Applies to	Comments
+	Addition	Numeric	
-	Subtraction	Numeric	
*	Multiplication	Numeric	
/	Division	Numeric	
%	Modulo	Integers	a % b returns the remainder when a is divided by b
<<	Left shift	Integer	a << b returns a left-shifted b-bit positions
>>	Right shift	Integer	a >> b returns a right-shifted b-bit positions
==	Equals	Any; a and b must be numeric or of the same data type	a == b returns 1 (true) if a equals b and 0 (false) otherwise.
<	Less than	Same as ==.	a < b returns 1 if a is less than b and 0 otherwise. (See the note below the table.)
>	Greater than	Same as ==	a > b returns 1 if a is greater than b and 0 otherwise. (See the note below the table.)
<=	Less than or equal to	Same as ==	a <= b returns 1 if a < b or a == b, and 0 otherwise. (See the note below the table.)
>=	Greater than or equal to	Same as ==	a >= b returns 1 if a > b or a == b, and 0 otherwise. (See the note below the table.)
!=	Not equals	Same as ==	a != b returns 1 if a is not equal to b, and 0 otherwise.
^	Bitwise exclusive OR	Integer	a ^ b returns an integer with bit value 1 in each bit position where the bit values of a and b differ, and a bit value of 0 otherwise.

Symbol	Name	Applies to	Comments
&	Bitwise AND	Integer	a & b returns an integer with bit value 0 in each bit position where the bit values of a and b are both 1, and a bit value of 0 otherwise.
	Bitwise (inclusive) OR	Integer	a b returns an integer with a bit value 1 in each bit position where the bit value a or b (or both) is 1, and 0 otherwise.
&&	Logical AND	Any; a and b must be numeric or of the same data type	a && b returns 0 if either a == 0 or b == 0 (or both), and 1 otherwise.
	Logical OR	Any; a and b must be numeric or of the same data type	a b returns 1 if either a != 0 or b != 0 (or both), and 0 otherwise.
+	Concatenation	String	a + b returns the string consisting of substring a followed by substring b.
?:			The ternary operator lets you write a conditional expression without using the if...else keyword. a ? b : c returns the value of b if a is true (non-zero) and the value of c if a is false.
=	Assignment	Any scalar; a and b must be numeric, numeric strings, or of the same data type	a = b places the value of b into a. Also, you can use "=" to do default conversions among integers, floats, decimals, and numeric strings.

Note: For the <, >, <=, and >= operators, if a and b are strings, lexicographic order is used. If a and b are date, time, or timestamp, temporal order is used.

The expression a * b * c evaluates as (a * b) * c. We describe this by saying that multiplication has left to right associativity. The expression a + b * c evaluates as a + (b * c). We describe this by saying multiplication has higher precedence than addition. The following table describes the precedence and associativity of the Transformation Language operators. Operators listed in the same row of the table have the same precedence, and you use parentheses to force a particular order of evaluation. Operators in a higher row have a higher order of precedence than operators in a lower row.

Table 59. Precedence and Associativity of Operators

Operators	Associativity
() []	left to right
! ~ ++ -- + - (unary)	right to left
* / %	left to right
+ - (binary)	left to right

Table 59. Precedence and Associativity of Operators (continued)

Operators	Associativity
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
	left to right
&&	left to right
:	left to right for right to left for :
?	right to left
=	right to left

Conditional Branching

The Transformation Language provides facilities for conditional branching. The following sections describe constructs available for conditional branching.

```
if ... else
  if (expression) statement1 else statement2;
```

If *expression* evaluates to a non-zero value (true) then *statement1* is executed. If *expression* evaluates to 0 (false) then *statement2* is executed. Both *statement1* and *statement2* can be compound statements.

You can omit else *statement2*. In that case, if *expression* evaluates to 0 the if statement has no effect.

Sample usage:

```
if (a < b)
  abs_difference = b - a;
else
  abs_difference = a - b;
```

This code sets *abs_difference* to the absolute value of *b - a*.

For Loop

```
for ( expression1 ; expression2; expression3) statement;
```

The order of execution is:

1. *expression1*. It is evaluated only once to initialize the loop variable.
2. *expression2*. If it evaluates to false, the loop terminates; otherwise, these expressions are executed in order:

```
  statement
  expression3
```

Control then returns to 2.

A sample usage is:

```
sum = 0;
sum_squares = 0;
for (i = 1; i < n; i++)
```

```
{
    sum = sum + 1;
    sum_squares = sum_squares + i*i;
}
```

This code sets sum to the sum of the first n integers and sum_squares to the sum of the squares of the first n integers.

While Loop

```
while ( expression ) statement ;
```

In a while loop, *statement*, which might be a compound statement, is executed repeatedly as long as *expression* evaluates to true. A sample usage is:

```
sum = 0;
i = 0;
while ((a[i] >= 0) && (i < n))
{
    sum = sum + a[i];
    i++;
}
```

This evaluates the sum of the array elements a[0] through a[n-1], or until a negative array element is encountered.

Break

The break command causes a for or while loop to exit immediately. For example, the following code does the same thing as the while loop shown immediately above:

```
sum = 0;
for (i = 0; i < n; i++)
{
    if (a[i] >= 0)
        sum = sum + a[i];
    else
        break;
}
```

Continue

The continue command is related to the break command, but used less often. It causes control to jump to the top of the loop. In the while loop, the test part is executed immediately. In a for loop, control passes to the increment step. If you want to sum all positive array entries in the array a[n], you can use the continue statement as follows:

```
sum = 0;
for (i = 0; i < n; i++)
{
    if (a[i] <= 0)
        continue;
    sum = sum + a[i];
}
```

This example could easily be written using an else statement rather than a continue statement. The continue statement is most useful when the part of the loop that follows is complicated, to avoid nesting the program too deeply.

Built-in functions

This section defines functions that are provided by the Transformation Language. It is presented in a series of tables that deal with data transformation functions of the following types:

- Lookup table functions
- Data conversion functions
- Mathematical functions
- String field functions
- Ustring field functions
- Bit manipulation functions
- Job monitoring functions
- Miscellaneous functions

When a function generates an output value, it returns the result. For functions with optional arguments, simply omit the optional argument to accept the default value. Default conversions among integer, float, decimal, and numeric string types are supported in the input arguments and the return value of the function. All integers can be signed or unsigned.

The transform operator has default NULL handling at the record-level with individual field "overrides". Options can be entered at the record level or the field level.

Lookup table functions

Function	Description
<code>lookup(<i>lookup_table</i>)</code>	Performs a lookup on the table using the current input record. It fills the current record of the lookup table with the first record found. If a match is not found, the current record is empty. If this is called multiple times on the same record, the record is filled with the current match if there is one and a new lookup will not be done.
<code>next_match(<i>lookup_table</i>)</code>	Gets the next record matched in the lookup and puts it into the current record of the table.
<code>clear_lookup(<i>lookup_table</i>)</code>	Checks to see if the current lookup record has a match. If this method returns false directly after the <code>lookup()</code> call, no matches were found in the table. Returns a boolean value specifying whether the record is empty or not.
<code>int8 is_match(<i>lookup_table</i>)</code>	Checks to see if the current lookup record has a match. If this method returns false directly after the <code>lookup()</code> call, no matches were found in the table. Returns a boolean value specifying whether the record is empty or not.

Data conversion functions

date field functions

WebSphere DataStage performs no automatic type conversion of date fields. Either an input data set must match the operator interface or you must effect a type conversion by means of the transform or modify operator.

A date conversion to or from a numeric field can be specified with any WebSphere DataStage numeric data type. WebSphere DataStage performs the necessary modifications and either translates a numeric field to the source data type or translates a conversion result to the numeric data type of the destination. For example, you can use the transformation function `month_day_from_date()` to convert a date to an `int8`, or to an `int16`, `int32`, `dfloat`, and so on.

date format

The default format of the date contained in the string is yyyy-mm-dd. However, you can specify an optional format string that defines another format. The format string requires that you provide enough information for WebSphere DataStage to determine a complete date (either day, month, and year, or year and day of year).

The *format_string* can contain one or a combination of the following elements:

Table 60. Date format tags

Tag	Variable width availability	Description	Value range	Options
%d	import	Day of month, variable width	1...31	s
%dd		Day of month, fixed width	01...31	s
%ddd	with v option	Day of year	1...366	s, v
%m	import	Month of year, variable width	1...12	s
%mm		Month of year, fixed width	01...12	s
%mmm		Month of year, short name, locale specific	Jan, Feb ...	t, u, w
%mmmm	import/export	Month of year, full name, locale specific	January, February ...	t, u, w, -N, +N
%yy		Year of century	00...99	s
%yyyy		Four digit year	0001 ...9999	
%NNNNyy		Cutoff year plus year of century	yy = 00...99	s
%e		Day of week, Sunday = day 1	1...7	
%E		Day of week, Monday = day 1	1...7	
%eee		Weekday short name, locale specific	Sun, Mon ...	t, u, w
%eeee	import/export	Weekday long name, locale specific	Sunday, Monday ...	t, u, w, -N, +N
%W	import	Week of year (ISO 8601, Mon)	1...53	s
%WW		Week of year (ISO 8601, Mon)	01...53	s

When you specify a date format string, prefix each component with the percent symbol (%) and separate the string's components with a suitable literal character.

The default date_format is %yyyy-%mm-%dd.

Where indicated the tags can represent variable-width data elements. Variable-width date elements can omit leading zeroes without causing errors.

The following options can be used in the format string where indicated in the table:

s Specify this option to allow leading spaces in date formats. The s option is specified in the form:

`%(tag,s)`

Where *tag* is the format string. For example:

`%(m,s)`

indicates a numeric month of year field in which values can contain leading spaces or zeroes and be one or two characters wide. If you specified the following date format property:

`%(d,s)/%(m,s)/%yyyy`

Then the following dates would all be valid:

`8/ 8/1958`

`08/08/1958`

`8/8/1958`

- ▀ Use this option in conjunction with the %ddd tag to represent day of year in variable-width format. So the following date property:

`%(ddd,v)`

represents values in the range 1 to 366. (If you omit the v option then the range of values would be 001 to 366.)

- ▀ Use this option to render uppercase text on output.
- ▀ Use this option to render lowercase text on output.
- ▀ Use this option to render titlecase text (initial capitals) on output.

The u, w, and t options are mutually exclusive. They affect how text is formatted for output. Input dates will still be correctly interpreted regardless of case.

- N Specify this option to left justify long day or month names so that the other elements in the date will be aligned.
- +N Specify this option to right justify long day or month names so that the other elements in the date will be aligned.

Names are left justified or right justified within a fixed width field of *N* characters (where *N* is between 1 and 99). Names will be truncated if necessary. The following are examples of justification in use:

`%dd-%(mmmm,-5)-%yyyyy`

`21-Augus-2006`

`%dd-%(mmmm,-10)-%yyyyy`

`21-August -2005`

`%dd-%(mmmm,+10)-%yyyyy`

`21- August-2005`

The locale for determining the setting of the day and month names can be controlled through the locale tag. This has the format:

`%(L,'locale')`

Where *locale* specifies the locale to be set using the *language_COUNTRY.variant* naming convention supported by ICU. See *NLS Guide* for a list of locales. The default locale for month names and weekday names markers is English unless overridden by a %L tag or the APT_IMPEXP_LOCALE environment variable (the tag takes precedence over the environment variable if both are set).

Use the locale tag in conjunction with your time format, for example the format string:

```
%({L,'es'})%eeee, %dd %mmmm %yyyy
```

Specifies the Spanish locale and would result in a date with the following format:

```
miércoles, 21 septiembre 2005
```

The format string is subject to the restrictions laid out in the following table. A format string can contain at most one tag from each row. In addition some rows are mutually incompatible, as indicated in the 'incompatible with' column. When some tags are used the format string requires that other tags are present too, as indicated in the 'requires' column.

Table 61. Format tag restrictions

Element	Numeric format tags	Text format tags	Requires	Incompatible with
year	%yyyy, %yy, %[nnnn]yy	-	-	-
month	%mm, %m	%mmm, %mmmm	year	week of year
day of month	%dd, %d	-	month	day of week, week of year
day of year	%ddd		year	day of month, day of week, week of year
day of week	%e, %E	%eee, %eeee	month, week of year	day of year
week of year	%WW		year	month, day of month, day of year

When a numeric variable-width input tag such as %d or %m is used, the field to the immediate right of the tag (if any) in the format string cannot be either a numeric tag, or a literal substring that starts with a digit. For example, all of the following format strings are invalid because of this restriction:

```
%d%m-%yyyy
```

```
%d%mm-%yyyy
```

```
%({d})%({mm})-%yyyy
```

```
%h00 hours
```

The *year_cutoff* is the year defining the beginning of the century in which all two-digit years fall. By default, the year cutoff is 1900; therefore, a two-digit year of 97 represents 1997.

You can specify any four-digit year as the year cutoff. All two-digit years then specify the next possible year ending in the specified two digits that is the same or greater than the cutoff. For example, if you set the year cutoff to 1930, the two-digit year 30 corresponds to 1930, and the two-digit year 29 corresponds to 2029.

On import and export, the *year_cutoff* is the base year.

This property is mutually exclusive with days_since, text, and julian.

You can include literal text in your date format. Any Unicode character other than null, backslash, or the percent sign can be used (although it is better to avoid control codes and other non-graphic characters). The following table lists special tags and escape sequences:

Tag	Escape sequence
%%	literal percent sign
\%	literal percent sign
\n	newline
\t	horizontal tab
\\"	single backslash

date Uformat

The date uformat provides support for international components in date fields. It's syntax is:

String%macroString%macroString%macroString

where *%macro* is a date formatting macro such as *%mmm* for a 3-character English month. Only the *String* components of date uformat can include multi-byte Unicode characters.

Note: Any argument that has to be double quoted cannot be a field name or a local variable. An argument must have the data format of its type.

Function	Description
<code>date date_from_days_since(int32 , " date " format_variable)</code>	Returns date by adding the given integer to the baseline date. Converts an integer field into a date by adding the integer to the specified base <i>date</i> . The <i>date</i> must be in the format <i>yyyy-mm-dd</i> and must be either double quoted or a variable.
<code>date date_from Julian_day(uint32)</code>	Returns the date given a Julian day.
<code>date date_from_string(string , " date_format " date_uformat format_variable)</code>	Returns a date from the given <i>string</i> formatted in the optional format specification. By default the <i>string</i> format is <i>yyyy-mm-dd</i> . For format descriptions, see and "date Uformat".
<code>date date_from_ustring(string , " date_format " date_uformat format_variable)</code>	Returns a date from the given <i>ustring</i> formatted in the optional format specification. By default the <i>ustring</i> format is <i>yyyy-mm-dd</i> . For format descriptions, see and "date Uformat".
<code>string string_from_date(date , " date_format " date_uformat)</code>	Converts the <i>date</i> to a string representation using the given format specification. By default the <i>ustring</i> format is <i>yyyy-mm-dd</i> . For format descriptions, see and "date Uformat".
<code>ustring ustring_from_date(date , " date_format " date_uformat)</code>	Converts the date to a <i>ustring</i> representation using the given format specification. By default the <i>ustring</i> format is <i>yyyy - mm -dd</i> . For format descriptions, see and "date Uformat".
<code>date date_from_timestamp(timestamp)</code>	Returns the date from the given <i>timestamp</i> .
<code>int32 days_since_from_date(date , " source_date " format_variable)</code>	Returns a value corresponding to the number of days from <i>source_date</i> to <i>date</i> . <i>source_date</i> must be in the form <i>yyyy - mm -dd</i> and must be double quoted or be a variable.

Function	Description
uint32 julian_day_from_date(<i>date</i>)	Returns a Julian date given the <i>date</i> .
int8 month_day_from_date(<i>date</i>)	Returns the day of the month given the <i>date</i> . For example, the date 07-23-2001 returns 23.
int8 month_from_date(<i>date</i>)	Returns the month from the given <i>date</i> . For example, the date 07-23-2001 returns 7.
date next_weekday_from_date (<i>date</i> , " <i>day</i> " <i>format_variable</i>)	The value returned is the date of the specified day of the week soonest after <i>date</i> (including the <i>date</i>). The <i>day</i> argument is optional. It is a string or variable specifying a day of the week. You can specify <i>day</i> by either the first three characters of the day name or the full day name. By default, the value is Sunday.
date previous_weekday_from_date (<i>date</i> , " <i>day</i> " <i>format_variable</i>)	Returns the previous weekday date from <i>date</i> . The destination contains the closest date for the specified day of the week earlier than the source date (including the source date) The <i>day</i> argument is optional. It is a string or variable specifying a day of the week. You can specify <i>day</i> using either the first three characters of the day name or the full day name. By default, the value is Sunday.
int8 weekday_from_date (<i>date</i> , " <i>origin_day</i> " <i>format_variable</i>)	Returns the day of the week from <i>date</i> . The optional argument <i>origin_day</i> is a string or variable specifying the day considered to be day zero of the week. You can specify the day using either the first three characters of the day name or the full day name. If omitted, Sunday is day zero.
int16 year_day_from_date(<i>date</i>)	Returns the day of the year (1-366) from <i>date</i> .
int16 year_from_date(<i>date</i>)	Returns the year from <i>date</i> . For example, the date 07-23-2001 returns 2001.
int8 year_week_from_date(<i>date</i>)	Returns the week of the year from <i>date</i> . For example, the date 07-23-2001 returns 30.

decimal and float Field Functions

You can do the following transformations using the decimal and float field functions.

- Assign a decimal to an integer or float or numeric string, or compare a decimal to an integer or float or numeric string.
- Specify an optional fix_zero argument (int8) to cause a decimal field containing all zeros to be treated as a valid zero.
- Optionally specify a value for the rounding type (*r_type*) for many conversions. The values of *r_type* are:
 - ceil: Round the source field toward positive infinity. This mode corresponds to the IEEE 754 Round Up mode.
Examples: 1.4 -> 2, -1.6 -> -1
 - floor: Round the source field toward negative infinity. This mode corresponds to the IEEE 754 Round Down mode.
Examples: 1.6 -> 1, -1.4 -> -2

- round_inf: Round or truncate the source field toward the nearest representable value, breaking ties by rounding positive values toward positive infinity and negative values toward negative infinity. This mode corresponds to the COBOL ROUNDED mode.
Examples: 1.4 -> 1, 1.5 -> 2, -1.4 -> -1, -1.5 -> -2
- trunc_zero (default): Discard any fractional digits to the right of the right-most fractional digit supported in the destination, regardless of sign. For example, if the destination is an integer, all fractional digits are truncated. If the destination is another decimal with a smaller scale, round or truncate to the scale size of the destination decimal. This mode corresponds to the COBOL INTEGER-PART function.
Examples: 1.6 -> 1, -1.6 -> -1

Function	Description
decimal decimal_from_decimal (<i>decimal</i> , " <i>r_type</i> " <i>format_variable</i>)	Returns <i>decimal</i> in decimal representation, changing the precision and scale according to the returned type. The rounding type, <i>r_type</i> , might be ceil, floor, round_inf, or trunc_zero as described above this table. The default rtype is trunc_zero.
decimal decimal_from_dfloat (<i>dfloat</i> , " <i>r_type</i> " <i>format_variable</i>)	Returns <i>dfloat</i> in decimal representation. The rounding type, <i>r_type</i> , might be ceil, floor, round_inf, or trunc_zero as described above this table. The default is trunc_zero.
decimal decimal_from_string (<i>string</i> , " <i>r_type</i> " <i>format_variable</i>)	Returns <i>string</i> in decimal representation. The rounding type, <i>r_type</i> , might be ceil, floor, round_inf, or trunc_zero as described above this table. The default is trunc_zero.
decimal decimal_from_ushort (<i>ushort</i> , " <i>r_type</i> " <i>format_variable</i>)	Returns <i>ushort</i> in decimal representation. The rounding type, <i>r_type</i> , might be ceil, floor, round_inf, or trunc_zero as described above this table. The default is trunc_zero.
dfloat dfloat_from_decimal (<i>decimal</i> , "fix-zero" <i>format_variable</i>)	Returns decimal in dfloat representation.
int32 int32_from_decimal (<i>decimal</i> , " <i>r_type</i> fix_zero")	Returns int32 in decimal representation. The rounding type, <i>r_type</i> , might be ceil, floor, round_inf, or trunc_zero as described above this table. The default is trunc_zero.
int64 int64_from_decimal (<i>decimal</i> , " <i>r_type</i> fix_zero")	Returns int64 in decimal representation. The rounding type, <i>r_type</i> , might be ceil, floor, round_inf, or trunc_zero as described above this table. The default is trunc_zero.
uint64 uint64_from_decimal (<i>decimal</i> , " <i>r_type</i> fix_zero")	Returns uint64 in decimal representation. The rounding type, <i>r_type</i> , might be ceil, floor, round_inf, or trunc_zero as described above this table. The default is trunc_zero.
string string_from_decimal (<i>decimal</i> , "fix_zero" <i>suppress_zero</i>)	Returns string in decimal representation. fix_zero causes a decimal field containing all zeros to be treated as a valid zero. suppress_zero argument specifies that the returned ustring value will have no leading or trailing zeros. Examples: 000.100 -> 0.1; 001.000 -> 1; -001.100 -> -1.1

Function	Description
<code>wstring wstring_from_decimal (<i>decimal</i> , "fix_zero suppress_zero" <i>format_variable</i>)</code>	Returns <i>wstring</i> in decimal representation. fix_zero causes a decimal field containing all zeros to be treated as a valid zero. suppress_zero argument specifies that the returned <i>wstring</i> value will have no leading or trailing zeros. Examples: 000.100 -> 0.1; 001.000 -> 1; -001.100 -> -1.1
<code>string string_from_decimal (<i>decimal</i> , "fix_zero suppress_zero" <i>format_variable</i>)</code>	Returns <i>string</i> in decimal representation. fix_zero causes a decimal field containing all zeros to be treated as a valid zero. suppress_zero argument specifies that the returned <i>string</i> value will have no leading or trailing zeros. Examples: 000.100 -> 0.1; 001.000 -> 1; -001.100 -> -1.1
<code>dfloat mantissa_from_dfloat(<i>dfloat</i>)</code>	Returns the mantissa (the digits right of the decimal point) from <i>dfloat</i> .
<code>dfloat mantissa_from_decimal (<i>decimal</i>)</code>	Returns the mantissa (the digits right of the decimal point) from <i>decimal</i> .

raw Field Functions

Use the raw field functions to transform a string into a raw data type and to determine the length of a raw value.

Function	Description
<code>raw raw_from_string(<i>string</i>)</code>	Returns <i>string</i> in raw representation.
<code>raw u_raw_from_string(<i>wstring</i>)</code>	Returns <i>wstring</i> in raw representation.
<code>int32 raw_length(<i>raw</i>)</code>	Returns the length of the raw field.

time and timestamp field functions

WebSphere DataStage performs no automatic conversions to or from the time and timestamp data types. You must use the modify or transform operator if you want to convert a source or destination field. Most field conversions extract a portion of the time, such as hours or minutes, and write it into a destination field.

Time conversion to a numeric field can be used with any WebSphere DataStage numeric data type. WebSphere DataStage performs the necessary modifications to translate a conversion result to the numeric data type of the destination. For example, you can use the transformation function `hours_from_time()` to convert a time to an int8, or to an int16, int32, dfloat, and so on.

The `string_from_time()` and `time_from_string()` conversion functions take a format as a parameter of the conversion. The default format of the time in the string is hh:nn:ss. However, you can specify an optional format string defining another time format. The format string must contain a specification for hours, minutes, and seconds.

time Uformat

The time uformat provides support for international components in time fields. Its syntax is:

`String % macroString % macroString % macroString`

where `%macro` is a time formatting macro such as `%hh` for a two-digit hour. See below for a description of the date format macros. Only the `String` components of time uformat can include multi-byte Unicode characters.

timestamp Uformat

This format is a concatenation of the date uformat and time uformat which are described in "date Uformat" and "time Uformat". The order of the formats does not matter, but the two formats cannot be mixed.

time Format

The possible components of the `time_format` string are given in the following table:

Table 62. Time format tags

Tag	Variable width availability	Description	Value range	Options
<code>%h</code>	import	Hour (24), variable width	0...23	s
<code>%hh</code>		Hour (24), fixed width	0...23	s
<code>%H</code>	import	Hour (12), variable width	1...12	s
<code>%HH</code>		Hour (12), fixed width	01...12	s
<code>%n</code>	import	Minutes, variable width	0...59	s
<code>%nn</code>		Minutes, fixed width	0..59	s
<code>%s</code>	import	Seconds, variable width	0..59	s
<code>%ss</code>		Seconds, fixed width	0..59	s
<code>%s.N</code>	import	Seconds + fraction ($N = 0..6$)	-	s, c, C
<code>%ss.N</code>		Seconds + fraction ($N = 0..6$)	-	s, c, C
<code>%SSS</code>	with v option	Milliseconds	0...999	s, v
<code>%SSSSSS</code>	with v option	Microseconds	0...999999	s, v
<code>%aa</code>	German	am/pm marker, locale specific	am, pm	u, w

By default, the format of the time contained in the string is `%hh:%nn:%ss`. However, you can specify a format string defining the format of the string field.

You must prefix each component of the format string with the percent symbol. Separate the string's components with any character except the percent sign (%).

Where indicated the tags can represent variable-fields on import, export, or both. Variable-width date elements can omit leading zeroes without causing errors.

The following options can be used in the format string where indicated:

s Specify this option to allow leading spaces in time formats. The s option is specified in the form:

`%(tag,s)`

Where *tag* is the format string. For example:

`%(n,s)`

indicates a minute field in which values can contain leading spaces or zeroes and be one or two characters wide. If you specified the following date format property:

`%(h,s):$(n,s):$(s,s)`

Then the following times would all be valid:

`20: 6:58`

`20:06:58`

`20:6:58`

v Use this option in conjunction with the %SSS or %SSSSSS tags to represent milliseconds or microseconds in variable-width format. So the time property:

`%(SSS,v)`

represents values in the range 0 to 999. (If you omit the v option then the range of values would be 000 to 999.)

u Use this option to render the am/pm text in uppercase on output.

w Use this option to render the am/pm text in lowercase on output.

c Specify this option to use a comma as the decimal separator in the %ss.N tag.

C Specify this option to use a period as the decimal separator in the %ss.N tag.

The c and C options override the default setting of the locale.

The locale for determining the setting of the am/pm string and the default decimal separator can be controlled through the locale tag. This has the format:

`%(L,'locale')`

Where *locale* specifies the locale to be set using the *language_COUNTRY.variant* naming convention supported by ICU. See *NLS Guide* for a list of locales. The default locale for am/pm string and separators markers is English unless overridden by a %L tag or the APT_IMPEXP_LOCALE environment variable (the tag takes precedence over the environment variable if both are set).

Use the locale tag in conjunction with your time format, for example:

`%L('es')%HH:%nn %aa`

Specifies the Spanish locale.

The format string is subject to the restrictions laid out in the following table. A format string can contain at most one tag from each row. In addition some rows are mutually incompatible, as indicated in the 'incompatible with' column. When some tags are used the format string requires that other tags are present too, as indicated in the 'requires' column.

Table 63. Format tag restrictions

Element	Numeric format tags	Text format tags	Requires	Incompatible with
hour	%hh, %h, %HH, %H	-	-	-

Table 63. Format tag restrictions (continued)

Element	Numeric format tags	Text format tags	Requires	Incompatible with
am/pm marker	-	%aa	hour (%HH)	hour (%hh)
minute	%nn, %n	-	-	-
second	%ss, %s	-	-	-
fraction of a second	%ss.N, %s.N, %SSS, %SSSSSS	-	-	-

You can include literal text in your date format. Any Unicode character other than null, backslash, or the percent sign can be used (although it is better to avoid control codes and other non-graphic characters). The following table lists special tags and escape sequences:

Tag	Escape sequence
%%	literal percent sign
\%	literal percent sign
\n	newline
\t	horizontal tab
\\"	single backslash

Function	Description
int8 hours_from_time(<i>time</i>)	Returns the hour portion of the given time.
int32 microseconds_from_time(<i>time</i>)	Returns the number of microseconds from the given time.
dfloat midnight_seconds_from_time (<i>time</i>)	Returns the number of seconds from midnight to <i>time</i> .
int8 minutes_from_time(<i>time</i>)	Returns the number of minutes from <i>time</i> .
dfloat seconds_from_time(<i>time</i>)	Returns the number of seconds from time.
dfloat seconds_since_timestamp (<i>timestamp</i> , "source_timestamp_string" <i>format_variable</i>)	Returns the number of seconds from <i>timestamp</i> to the base timestamp, or optionally the second timestamp argument for the number of seconds between timestamps. The <i>source_timestamp_string</i> argument must be double quoted or be a variable.
time time_from_midnight_seconds(<i>dfloat</i>)	Returns the time given the number of seconds (<i>dfloat</i>) since midnight.
time time_from_string (<i>string</i> , <i>time_format</i> <i>time_uformat</i> <i>format_variable</i>)	Returns a time representation of <i>string</i> using the optional <i>time_format</i> , <i>time_uformat</i> , or <i>format_variable</i> . By default, the time format is <i>hh:mm:ss</i> . For format descriptions, see and "time Uformat".
time time_from_ustring (<i>ustring</i> , <i>time_format</i> <i>time_uformat</i> <i>format_variable</i>)	Returns a time representation of <i>ustring</i> using the optional <i>time_format</i> , <i>time_uformat</i> , or <i>format_variable</i> specification. By default, the time format is <i>hh:mm:ss</i> . For format descriptions, see and "time Uformat".
string string_from_time (<i>time</i> , " <i>time_format</i> " <i>format_variable</i> <i>time_uformat</i>)	Returns a string from <i>time</i> . The format argument is optional. The default time format is <i>hh:mm:ss</i> . For format descriptions, see and "time Uformat".
string string_from_time (<i>time</i> , " <i>time_format</i> " <i>format_variable</i> <i>time_format</i>)	Returns a ustring from <i>time</i> . The format argument is optional. The default time format is <i>hh:mm:ss</i> . For format descriptions, see and "time Uformat".

Function	Description
time time_from_timestamp(<i>timestamp</i>)	Returns the time from <i>timestamp</i> .
date date_from_timestamp(<i>timestamp</i>)	Returns the date from the given <i>timestamp</i> .
timestamp timestamp_from_date_time (<i>date</i> , <i>time</i>)	Returns a timestamp from <i>date</i> and <i>time</i> . The <i>date</i> specifies the date portion (<i>yyyy - nn - dd</i>) of the timestamp. The <i>time</i> argument specifies the time to be used when building the timestamp. The <i>time</i> argument must be in the <i>hh : nn :ss</i> format.
timestamp timestamp_from_seconds_since (<i>dfloat</i> , " <i>original_timestamp_string</i> " <i>format_variable</i>)	Returns the timestamp from the number of seconds (<i>dfloat</i>) from the base timestamp or the <i>original_timestamp_string</i> argument. The <i>original_timestamp_string</i> must be double quoted or be a variable.
timestamp timestamp_from_string (<i>string</i> , " <i>timestamp_format</i> " <i>timestamp_uformat</i> <i>format_variable</i>)	Returns a timestamp from <i>string</i> , in the optional <i>timestamp_format</i> , <i>timestamp_uformat</i> , or <i>format_variable</i> . The <i>timestamp_format</i> must be double quoted or be a variable. The default format is <i>yyyy - nn - dd hh : nn : ss</i> . <i>timestamp_format</i> is described in .
timestamp timestamp_from_ushort (<i>ushort</i> , " <i>timestamp_format</i> " <i>timestamp_uformat</i> <i>format_variable</i>)	Returns a timestamp from <i>ushort</i> , in the optional format specification. The <i>timestamp_format</i> must be a double quoted string, a <i>uformat</i> , or a variable. The default format is <i>yyyy - nn - dd hh : nn : ss</i> . <i>timestamp_uformat</i> is described in .
string string_from_timestamp (<i>timestamp</i> , " <i>timestamp_format</i> " <i>format_variable</i>)	Returns a string from <i>timestamp</i> . The formatting specification is optional. The default format is <i>yyyy - mm - dd hh : mm : ss</i> .
ushort ushort_from_timestamp (<i>timestamp</i> , " <i>timestamp_format</i> " <i>format_variable</i>)	Returns a ushort from <i>timestamp</i> . The formatting specification is optional. The default format is <i>yyyy - mm - dd hh : mm : ss</i> .
timestamp timestamp_from_time (<i>time</i> , <i>time_format</i> <i>time_uformat</i>)	Returns a timestamp from <i>time</i> . For format descriptions, see and "time Uformat"
date date_from_timestamp(<i>timestamp</i>)	Returns the date from the given <i>timestamp</i> .
timestamp timestamp_from_timet(<i>int32</i>)	Returns a timestamp from the given UNIX time_t representation (<i>int32</i>).
int32 timet_from_timestamp(<i>timestamp</i>)	Returns the UNIX time_t representation of <i>timestamp</i> .

null handling functions

lists the transformation functions for NULL handling.

All WebSphere DataStage data types support nulls. As part of processing a record, an operator can detect a null and take the appropriate action, for example, it can omit the null field from a calculation or signal an error condition.

WebSphere DataStage represents nulls in two ways.

- It allocates a single bit to mark a field as null. This type of representation is called an out-of-band null.
- It designates a specific field value to indicate a null, for example a numeric field's most negative possible value. This type of representation is called an in-band null. In-band null representation can be disadvantageous because you must reserve a field value for nulls and this value cannot be treated as valid data elsewhere.

The null-handling functions can change a null representation from an out-of-band null to an in-band null and from an in-band null to an out-of-band null.

Function	Description
destination_field handle_null (<i>source_field</i> , <i>value</i>)	Change the <i>source_field</i> NULL representations from out-of-band representation to an in-band representation. The <i>value</i> field assigns the value that corresponds to NULL.
destination_field make_null (<i>source_field</i> , <i>value</i>)	Changes <i>source_field</i> NULL representation from in-band NULL representation to out-of-band. The <i>value</i> field allows multiple valid NULL values to be inputted as arguments.
int8 notnull(<i>source_field</i>)	Returns 1 if <i>source_field</i> is not NULL, otherwise returns 0.
int8 null(<i>source_field</i>)	Returns 1 if <i>source_field</i> is NULL, otherwise returns 0.
set_null()	This function is used with "=" to set the left side output field, when it is nullable, to null. For example: a-field = set_null();
int8 is_dfloat_inband_null (<i>dfloat</i>)	Returns 1 if <i>dfloat</i> is an inband null; otherwise it returns 0.
int8 is_int16_inband_null (<i>int16</i>)	Returns 1 if <i>int16</i> is an inband null; otherwise it returns 0.
int8 is_int32_inband_null (<i>int32</i>)	Returns 1 if <i>int32</i> is an inband null; otherwise it returns 0.
int8 is_int64_inband_null (<i>int64</i>)	Returns 1 if <i>int64</i> is an inband null; otherwise it returns 0.
int8 is_sfloat_inband_null (<i>sfloat</i>)	Returns 1 if <i>sfloat</i> is an inband null; otherwise it returns 0.
int8 is_string_inband_null (<i>string</i>)	Returns 1 if <i>string</i> is an inband null; otherwise it returns 0.
int8 u_is_string_inband_null (<i>cstring</i>)	Returns 1 if <i>cstring</i> is an inband null; otherwise it returns 0.

Note: Null-handling functions cannot be used for subrecord fields.

Mathematical functions

Function	Description
int32 abs(<i>int32</i>)	Returns the absolute value of <i>int32</i> .
dfloat acos(<i>dfloat</i>)	Returns the principal value of the arc cosine of <i>dfloat</i> .
dfloat asin(<i>dfloat</i>)	Returns the principal value of the arc sine of <i>dfloat</i> .
dfloat atan(<i>dfloat</i>)	Returns the principal value of the arc tangent of <i>dfloat</i> .
dfloat atan2(<i>dfloat</i> , <i>dfloat</i>)	Returns the principal value of the arc tangent of y/x (where y is the first argument).
dfloat ceil(<i>decimal</i>)	Returns the smallest dfloat value greater than or equal to <i>decimal</i> .
dfloat cos(<i>dfloat</i>)	Returns the cosine of the given angle (<i>dfloat</i>) expressed in radians.
dfloat cosh(<i>dfloat</i>)	Returns the hyperbolic cosine of <i>dfloat</i> .

Function	Description
<code>dfloat exp(dfloat)</code>	Returns the exponential of <i>dfloat</i> .
<code>dfloat fabs(dfloat)</code>	Returns the absolute value of <i>dfloat</i> .
<code>dfloat floor(decimal)</code>	Returns the largest <i>dfloat</i> value less than or equal to <i>decimal</i> .
<code>dfloat ldexp(dfloat , int32)</code>	Reconstructs <i>dfloat</i> out of the mantissa and exponent of <i>int32</i> .
<code>uint64 llabs(int64)</code>	Returns the absolute value of <i>int64</i> .
<code>dfloat log(dfloat)</code>	Returns the natural (base e) logarithm of <i>dfloat</i> .
<code>dfloat log10(dfloat)</code>	Returns the logarithm to the base 10 of <i>dfloat</i> .
<code>int32 max(int32 , int32)</code>	Returns the larger of the two integers.
<code>int32 min(int32 , int32)</code>	Returns the smaller of the two integers.
<code>dfloat pow(dfloat , dfloat)</code>	Returns the result of raising x (the first argument) to the power y (the second argument).
<code>uint32 rand()</code>	Returns a pseudo-random integer between 0 and $2^{32} - 1$. The function uses a multiplicative congruential random-number generator with period 2^{32} . See the UNIX man page for <i>rand</i> for more details.
<code>uint32 random()</code>	Returns a random integer between 0 and $2^{31} - 1$. The function uses a nonlinear additive feedback random-number generator employing a default state array size of 31 long integers to return successive pseudo-random numbers. The period of this random-number generator is approximately $16 \times (2^{31} - 1)$. Compared with <i>rand</i> , <i>random</i> is slower but more random. See the UNIX man page for <i>random</i> for more details.
<code>dfloat sin(dfloat)</code>	Returns the sine of <i>dfloat</i> expressed in radians.
<code>dfloat sinh(dfloat)</code>	Returns the hyperbolic sine of <i>dfloat</i> .
<code>dfloat sqrt(dfloat)</code>	Returns the square root of <i>dfloat</i> .
<code>int32 quotient_from_dfloat (dfloat1 , dfloat2)</code>	Returns the value of the quotient after <i>dfloat1</i> is divided by <i>dfloat2</i> .
<code>srand(uint32)</code>	Sets a new seed (<i>uint32</i>) for the <i>frand()</i> or <i>srand()</i> random number generator.
<code>srandom(uint32)</code>	Sets a random seed for the <i>random()</i> number generator. See the UNIX man page for <i>srandom</i> for more details.
<code>dfloat tan(dfloat)</code>	Returns the tangent of the given angle (<i>dfloat</i>) expressed in radians.
<code>dfloat tanh(dfloat)</code>	Returns the hyperbolic tangent of <i>dfloat</i> .

String field functions

Strings can be assigned (=), compared (==, <, >=, and so on), and concatenated (+) in the Transformation Language. In addition, the functions described in below are available for string manipulations, and the functions described in are available for *ustring* manipulations. When a long string is assigned to a short string, the long string is truncated to the length of the short string. The term white space refers to spaces, tabs, and any other blank space.

You can construct a string lookup table to use when default conversions do not yield satisfactory results. A string lookup table is a table of two columns and as many rows as are required to perform a conversion to or from a string as shown in the following table.

Numeric Value	String or Ustring
numVal1	string1 ustring1
numVal2	string2 ustring1
...	...
numVal3	stringn ustringn

Each row of the lookup table specifies an association between a 16-bit integer or unsigned 32-bit integer value and a string or ustring. WebSphere DataStage scans the Numeric Value or the String or Ustring column until it encounters the value or string to be translated. The output is the corresponding entry in the row.

The numeric value to be converted might be of the int16 or the uint32 data type. WebSphere DataStage converts strings to values of the int16 or uint32 data type using the same table.

If the input contains a numeric value or string that is not listed in the table, WebSphere DataStage operates as follows:

- If a numeric value is unknown, an empty string is returned by default. However, you can set a default string value to be returned by the string lookup table.
- If a string has no corresponding value, 0 is returned by default. However, you can set a default numeric value to be returned by the string lookup table.

A table definition defines the rows of a string or ustring lookup table and has the following form:

```
{propertyList} ('string' | 'cstring' = value; 'string' | 'cstring'= value; ... )
```

where:

propertyList is one or more of the following options; the entire list is enclosed in braces and properties are separated by commas if there are more than one:

- *case_sensitive*: perform a case-sensitive search for matching strings; the default is case-insensitive.
- *default_value = defVal*: the default numeric value returned for a string that does not match any of the strings in the table.
- *default_string = defString*: the default string returned for numeric values that do not match any numeric value in the table.
- *string* or *cstring* specifies a comma-separated list of strings or cstrings associated with *value*; enclose each string or cstring in quotes.
- *value* specifies a comma-separated list of 16-bit integer values associated with *string* or *cstring*.

Function	Description
int8 is_alnum(<i>string</i>)	Returns 1 true if <i>string</i> consists entirely of alphanumeric characters.
int8 is_alpha(<i>string</i>)	Returns 1 true if <i>string</i> consists entirely of alphabetic characters.
int8 is_numeric(<i>string</i>)	Returns 1 true if <i>string</i> consists entirely of numeric characters, including decimal and sign.

Function	Description
int8 is_valid (" <i>type_string</i> ", " <i>value_string</i> ")	<p>Returns 1 (true) if <i>value_string</i> is valid according to <i>type_string</i>, including NULL. The <i>type_string</i> argument is required. It must specify a WebSphere DataStage schema data type.</p> <p>Integer types are checked to ensure the <i>value_string</i> is numeric (signed or unsigned), a whole number, and a valid value (for example, 1024 can not be assigned to an int8 type).</p> <p>Decimal types are checked to ensure the <i>value_string</i> is numeric (signed or unsigned) and a valid value.</p> <p>Float types are checked to ensure the <i>value_string</i> is numeric (signed or unsigned) and a valid value (exponent is valid).</p> <p>String is always valid with the NULL exception below.</p> <p>For all types, if the field cannot be set to NULL and the string is NULL, 0 (false) is returned.</p> <p>Date, time, and timestamp types are checked to ensure they are correct, using the optional format argument, and valid values.</p> <p>Raw cannot be checked since the input is a string.</p>
int16 lookup_int16_from_string (<i>string</i> , " <i>table_definition</i> " <i>table_variable</i>)	Returns an integer corresponding to <i>string</i> using <i>table_definition</i> string or variable. See "String Conversions and Lookup Tables" for more information.
string lookup_string_from_int16 (<i>int16</i> , " <i>table_definition</i> " <i>table_variable</i>)	Returns a string corresponding to <i>int16</i> using <i>table_definition</i> string or variable. See "String Conversions and Lookup Tables" for more information.
string lookup_string_from_uint32 (<i>uint32</i> , " <i>table_definition</i> " <i>table_variable</i>)	Returns a string corresponding to <i>uint32</i> using <i>table_definition</i> string or variable. See "String Conversions and Lookup Tables" for more information.
uint32 lookup_uint32_from_string (<i>string</i> , " <i>table_definition</i> " <i>table_variable</i>)	Returns an unsigned integer from <i>string</i> using <i>table_definition</i> string or variable.
string lower_case(<i>string</i>)	Converts <i>string</i> to lowercase. Non-alphabetic characters are ignored in the transformation.
string string_from_date (<i>date</i> , " <i>date_format</i> " <i>format_variable</i> <i>date_ufORMAT</i>)	<p>Converts <i>date</i> to a string representation using the specified optional formatting specification.</p> <p>By default, the date format is <i>yyyy - mm - dd</i>. For format descriptions, see "date Uformat".</p>
string string_from_decimal (<i>decimal</i> , "fix_zero suppress_zero" <i>format_variable</i>)	<p>Returns a string from <i>decimal</i>.</p> <p><i>fix_zero</i> causes a decimal field containing all zeros to be treated as a valid zero.</p> <p><i>suppress_zero</i> argument specifies that the returned <i>ustring</i> value will have no leading or trailing zeros.</p> <p>Examples:</p> <p>000.100 -> 0.1; 001.000 -> 1; -001.100 -> -1.1</p> <p>The formatting specification is optional.</p>

Function	Description
<code>string string_from_time (time , " time_format " format_variable time_uformat)</code>	Returns a string from <i>time</i> . The format argument is optional. The default time format is <i>hh : nn :ss</i> . For format descriptions, see and "time Uformat".
<code>string string_from_timestamp (timestamp , " timestamp_format " format_variable)</code>	Returns a string from <i>timestamp</i> . The formatting specification is optional. The default format is <i>yyyy-mm-dd hh:mm:ss</i> .
<code>string soundex (input_string , length , censusOption)</code>	Returns a string which represents the phonetic code for the string input word. Input words that produce the same code are considered phonetically equivalent. The empty string is returned if the input string is empty. <i>length</i> is an int8 and can be any value between 4 and 10. The default is 4. <i>censusOption</i> can be 0, 1, or 2 where 0 (the default) is enhanced soundex and not a census code; 1 are the normal census codes used in all censuses from 1920 on; and 2 are special census codes used intermittently in 1880, 1900, and 1910.
<code>string upper_case(string)</code>	Converts <i>string</i> to uppercase. Non-alphabetic characters are ignored in the transformation.
<code>string compact_whitespace (string)</code>	Returns a <i>string</i> after reducing all consecutive white space in <i>string</i> to a single space.
<code>string pad_string (string , pad_string , pad_length)</code>	Returns the string with the <i>pad_string</i> appended to the bounded length string for <i>pad_length</i> number of characters. <i>pad_length</i> is an int16. When the given <i>string</i> is a variable-length string, it defaults to a bounded-length of 1024 characters. If the given <i>string</i> is a fixed-length string, this function has no effect.
<code>string strip_whitespace(string)</code>	Returns <i>string</i> after stripping all white space in the string.
<code>string trim_leading_trailing (string)</code>	Returns <i>string</i> after removing all leading and trailing white space.
<code>string trim_leading(string)</code>	Returns a <i>string</i> after removing all leading white space.
<code>string trim_trailing(string)</code>	Returns a <i>string</i> after removing all trailing white space.
<code>int32 string_order_compare (string1 , string2, justification)</code>	Returns a numeric value specifying the result of the comparison. The numeric values are: -1: <i>string1</i> is less than <i>string2</i> 0: <i>string1</i> is equal to <i>string2</i> 1: <i>string1</i> is greater than <i>string2</i> The string <i>justification</i> argument is either 'L' or 'R'. It defaults to 'L' if not specified. 'L' means a standard character comparison, left to right. 'R' means that any numeric substrings within the strings starting at the same position are compared as numbers. For example an 'R' comparison of "AB100" and "AB99" indicates that AB100 is great than AB99, since 100 is greater than 99. The comparisons are case sensitive.

Function	Description
string replace_substring (<i>expression1</i> , <i>expression2</i> , <i>string</i>)	<p>Returns a string value that contains the given <i>string</i> , with any characters in <i>expression1</i> replaced by their corresponding characters in <i>expression2</i>. For example:</p> <pre>replace_substring ("ABC:", "abZ", "AGDCBDA")</pre> <p>returns "aGDZbDa", where any "A" gets replaced by "a", any "B" gets replaced by "b" and any "C" gets replaced by "Z".</p> <p>If <i>expression2</i> is longer than <i>expression1</i>, the extra characters are ignored.</p> <p>If <i>expression1</i> is longer than <i>expression2</i>, the extra characters in <i>expression1</i> are deleted from the given string (the corresponding characters are removed.) For example:</p> <pre>replace_substring("ABC", "ab", "AGDCBDA")</pre> <p>returns "aGDbDa".</p>
int32 count_substring (<i>string</i> , <i>substring</i>)	<p>Returns the number of times that <i>substring</i> occurs in <i>string</i> . If <i>substring</i> is an empty string, the number of characters in <i>string</i> is returned.</p>
int32 dcount_substring (<i>string</i> , <i>delimiter</i>)	<p>Returns the number of fields in <i>string</i> delimited by <i>delimiter</i> , where <i>delimiter</i> is a string. For example,</p> <pre>dcount_substring("abcFdefFghi", "F")</pre> <p>returns 3.</p> <p>If <i>delimiter</i> is an empty string, the number of characters in the string + 1 is returned. If delimiter is not empty, but does not exist in the given string, 1 is returned.</p>
string double_quote_string (<i>expression</i>)	<p>Returns the given string <i>expression</i> enclosed in double quotes.</p>
string substring_by_delimiter (string, <i>delimiter</i> , <i>occurrence</i> , <i>numsubstr</i>)	<p>The <i>string</i> and <i>delimiter</i> arguments are string values, and the <i>occurrence</i> and <i>numsubstr</i> arguments are int32 values.</p> <p>This function returns <i>numsubstr</i> substrings from <i>string</i> , delimited by <i>delimiter</i> and starting at substring number <i>occurrence</i> . An example is:</p> <pre>substring_by_delimiter ("abcFdefFghiFjkl", "F", 2, 2)</pre> <p>The string "defFghi" is returned.</p> <p>If <i>occurrence</i> is < 1, then 1 is assumed. If <i>occurrence</i> does not point to an existing field, the empty string is returned. If <i>numsubstr</i> is not specified or is less than 1, it defaults to 1.</p>
int32 index_of_substring (<i>string</i> , <i>substring</i> , <i>occurrence</i>)	<p>Returns the starting position of the nth occurrence of <i>substring</i> in <i>string</i> . The <i>occurrence</i> argument is an integer indicating the nth occurrence.</p> <p>If there is no nth occurrence or string doesn't contain any <i>substring</i> , -1 is returned. If <i>substring</i> is an empty string, -2 is returned.</p>

Function	Description
string left_substring (<i>string</i> , <i>length</i>)	Returns the first <i>length</i> characters of <i>string</i> . If <i>length</i> is 0, it returns the empty string. If <i>length</i> is greater than the length of the string, the entire <i>string</i> is returned.
string right_substring (<i>string</i> , <i>length</i>)	Returns the last <i>length</i> characters of <i>string</i> . If <i>length</i> is 0, it returns the empty string. If <i>length</i> is greater than the length of <i>string</i> , the entire <i>string</i> is returned.
string string_of_space(<i>count</i>)	Returns a string containing <i>count</i> spaces. The empty string is returned for a <i>count</i> of 0 or less.
string single_quote_string (<i>expression</i>)	Returns the <i>expression</i> string enclosed in single quotes.
string string_of_substring (<i>string</i> , <i>count</i>)	Returns a <i>string</i> containing <i>count</i> occurrences of <i>string</i> . The empty string is returned for a count of 0 or less.
string trimc_string (<i>string</i> [, <i>character</i> [, <i>option</i>]])	If only <i>string</i> is specified, all leading and trailing spaces and tabs are removed, and all multiple occurrences of spaces and tabs are reduced to a single space or tab. If <i>string</i> and <i>character</i> are specified, option defaults to 'R' The available option values are: 'A' remove all occurrences of character 'B' remove both leading and trailing occurrences of character. 'D' remove leading, trailing, and redundant white-space characters. 'E' remove trailing white-space characters 'F' remove leading white-space characters 'L' remove all leading occurrences of character 'R' remove all leading, trailing, and redundant occurrences of character 'T' remove all trailing occurrences of character
string system_time_date()	Returns the current system time in this 24-hour format: <i>hh : mm :ss dd : mmm : yyyy</i>
int32 offset_of_substring (<i>string</i> , <i>substring</i> , <i>position</i>)	Searches for the <i>substring</i> in the <i>string</i> beginning at character number <i>position</i> , where <i>position</i> is an uint32. Returns the starting position of the substring.
int8 string_case_compare (<i>string</i> , <i>string</i>)	This is a case-insensitive version of string_compare() below.
int8 string_compare (<i>string</i> , <i>string</i>)	Compares two strings and returns the index (0 or 1) of the greater string.
int8 string_num_case_compare (<i>string</i> , <i>string</i> , <i>uint16</i>)	This is a case-insensitive version of string_num_compare() below.
string string_num_concatenate (<i>string</i> , <i>string</i> , <i>uint16</i>)	Returns a string after appending <i>uint16</i> characters from the second string onto the first string.
int8 string_num_compare (<i>string</i> , <i>string</i> , <i>uint16</i>)	Compares first <i>uint16</i> characters of two given strings and returns the index (0 or 1) of the greater string.
string string_num_copy (<i>string</i> , <i>uint16</i>)	Returns the first <i>uint16</i> characters from the given string .
int32 string_length(<i>string</i>)	Returns the length of the string.
string substring (<i>string</i> , <i>starting_position</i> , <i>length</i>)	Copies parts of strings to shorter strings by string extraction. The <i>starting_position</i> specifies the starting location of the substring; <i>length</i> specifies the substring length. The arguments <i>starting_position</i> and <i>length</i> are uint16 types and must be positive (≥ 0).

Function	Description
string char_from_num(int32)	Returns an ASCII character from the given <i>int32</i> . If given a value that is not associated with a character such as -1, the function returns a space. An example use is: char_from_num(38) which returns "&"
int32 num_from_char(string)	Returns the numeric value of the ASCII-character in the string. When this function is given an empty string, it returns 0; and when it is given a multi-character string, it uses the first character in the string. An example use is: num_from_char("&") which returns 38.

Ustring field functions

WebSphere DataStage provides the *ustring* type for multi-byte Unicode-character strings. *ustrings* can be assigned (=), compared (==, <, >=, and so on), and concatenated (+) in the Transformation Language. In addition, the functions described in are available for *ustring* manipulations. When a long string is assigned to a short string, the long string is truncated to the length of the short string. The term white space refers to spaces, tabs, and any other blank space.

Function	Description
ustring ustring_from_date (date , " date_format " date_format format_variable)	Converts <i>date</i> to a <i>ustring</i> representation using the optional format specification. By default, the format is <i>yyyy-mm-dd</i> . For format descriptions, see and "date Uformat".
ustring ustring_from_decimal (decimal , "fix_zero suppress_zero" format_variable)	Returns a <i>ustring</i> from <i>decimal</i> . <i>fix_zero</i> causes a decimal field containing all zeros to be treated as a valid zero. <i>suppress_zero</i> argument specifies that the returned <i>ustring</i> value will have no leading or trailing zeros. Examples: 000.100 -> 0.1; 001.000 -> 1; -001.100 -> -1.1 The format specification is optional.
ustring ustring_from_time (time , " time_format " time_uformat format_variable)	Returns a <i>ustring</i> from <i>time</i> using an optional format specification. The default time format is <i>hh:mm:ss</i> . For format descriptions, see and "time Uformat".
ustring ustring_from_timestamp (timestamp , " timestamp_format " format_variable)	Returns a <i>ustring</i> from <i>timestamp</i> . The format specification is optional. The default format is <i>yyyy-mm-dd hh:mm:ss</i> .
int8 u_isalnum(<i>ustring</i>)	Returns 1 (true) if <i>ustring</i> consists entirely of alphanumeric characters.
int8 u_isalpha(<i>ustring</i>)	Returns 1 (true) if <i>ustring</i> consists entirely of alphabetic characters.
int8 u_isnumeric(<i>ustring</i>)	Returns 1 (true) if <i>ustring</i> consists entirely of numeric characters, including decimal and sign.

Function	Description
<code>int16 lookup_int16_from_ustring (<i>ustring</i> , "table_definition" <i>table_variable</i>)</code>	Returns an integer corresponding to <i>ustring</i> using <i>table_definition</i> string or variable. See "String Conversions and Lookup Tables" for more information.
<code>ustring lookup_ustring_from_int16 (<i>int16</i> , "table_definition" <i>table_variable</i>)</code>	Returns a <i>ustring</i> corresponding to <i>int16</i> using <i>table_definition</i> string or <i>table_variable</i> . See "String Conversions and Lookup Tables" for more information.
<code>ustring lookup_ustring_from_uint32 (<i>uint32</i> , "table_definition" <i>table_variable</i>)</code>	Returns a <i>ustring</i> corresponding to <i>uint32</i> using <i>table_definition</i> string or variable. See "String Conversions and Lookup Tables" for more information.
<code>uint32 lookup_uint32_from_ustring (<i>string</i> , "table_definition" <i>table_variable</i>)</code>	Returns an unsigned integer from <i>ustring</i> using <i>table_definition</i> or <i>table_variable</i> .
<code>int8 u_is_valid (" <i>type_ustring</i> " , " <i>value_ustring</i> ")</code>	<p>Returns 1 (true) if <i>value_ustring</i> is valid according to <i>type_ustring</i>, including NULL. The <i>type_ustring</i> argument is required. It must specify a WebSphere DataStage schema data type.</p> <p>Integer types are checked to ensure the <i>value_ustring</i> is numeric (signed or unsigned), a whole number, and a valid value (for example, 1024 can not be assigned to an int8 type).</p> <p>Decimal types are checked to ensure the <i>value_ustring</i> is numeric (signed or unsigned) and a valid value.</p> <p>Float types are checked to ensure the <i>value_ustring</i> is numeric (signed or unsigned) and a valid value (exponent is valid).</p> <p>String is always valid with the NULL exception below.</p> <p>For all types, if the field cannot be set to NULL and the string is NULL, 0 (false) is returned.</p> <p>Date, time, and timestamp types are checked to ensure they are correct, using the optional format argument, and valid values.</p> <p>Raw cannot be checked since the input is a string.</p>
<code>ustring u_lower_case(<i>ustring</i>)</code>	Converts <i>ustring</i> to lowercase. Non-alphabetic characters are ignored in the transformation.
<code>ustring u_upper_case(<i>ustring</i>)</code>	Converts <i>ustring</i> to uppercase. Non-alphabetic characters are ignored in the transformation.
<code>ustring u_compact_whitespace (<i>ustring</i>)</code>	Returns the <i>ustring</i> after reducing all consecutive white space in <i>ustring</i> to a single space.
<code>ustring u_pad_string (<i>ustring</i> , <i>pad_ustring</i> , <i>pad_length</i>)</code>	<p>Returns the <i>ustring</i> with <i>pad_ustring</i> appended to the bounded length string for <i>pad_length</i> number of characters. <i>pad_length</i> is an int16.</p> <p>When the given <i>ustring</i> is a variable-length string, it defaults to a bounded-length of 1024 characters. If the given <i>ustring</i> is a fixed-length string, this function has no effect.</p>
<code>ustring u_strip_whitespace (<i>ustring</i>)</code>	Returns <i>ustring</i> after stripping all white space in the string.
<code>ustring u_trim_leading_trailing (<i>ustring</i>)</code>	Returns <i>ustring</i> after removing all leading and trailing white space.

Function	Description
<code>cstring u_trim_leading(<i>cstring</i>)</code>	Returns <i>cstring</i> after removing all leading white space.
<code>cstring u_trim_trailing (<i>cstring</i>)</code>	Returns a <i>cstring</i> after removing all trailing white space.
<code>int32 u_string_order_compare (<i>cstring1</i> , <i>cstring2</i>, <i>justification</i>)</code>	<p>Returns a numeric value specifying the result of the comparison. The numeric values are:</p> <ul style="list-style-type: none"> -1: <i>cstring1</i> is less than <i>cstring2</i> 0: <i>cstring1</i> is equal to <i>cstring2</i> 1: <i>cstring1</i> is greater than <i>cstring2</i> <p>The string <i>justification</i> argument is either 'L' or 'R'. It defaults to 'L' if not specified. 'L' means a standard character comparison, left to right. 'R' means that any numeric substrings within the strings starting at the same position are compared as numbers. For example an 'R' comparison of "AB100" and "AB99" indicates that AB100 is great than AB99, since 100 is greater than 99. The comparisons are case sensitive.</p>
<code>cstring u_replace_substring (<i>expression1</i> , <i>expression2</i>, <i>cstring</i>)</code>	<p>Returns a <i>cstring</i> value that contains the given <i>cstring</i> , with any characters in <i>expression1</i> replaced by their corresponding characters in <i>expression2</i> . For example:</p> <pre>u_replace_substring ("ABC", "abZ", "AGDCBDA")</pre> <p>returns "aGDZbDa", where any "A" gets replaced by "a", any "B" gets replaced by "b" and any "C" gets replaced by "Z".</p> <p>If <i>expression2</i> is longer than <i>expression1</i> , the extra characters are ignored.</p> <p>If <i>expression1</i> is longer than <i>expression2</i>, the extra characters in <i>expression1</i> are deleted from the given string (the corresponding characters are removed.) For example:</p> <pre>u_replace_substring("ABC", "ab", "AGDCBDA")</pre> <p>returns "aGDbDa".</p>
<code>int32 u_count_substring (<i>cstring</i> , <i>sub_cstring</i>)</code>	Returns the number of times that <i>sub_cstring</i> occurs in <i>cstring</i> . If <i>sub_cstring</i> is an empty string, the number of characters in <i>cstring</i> is returned.
<code>int32 u_dcount_substring (<i>cstring</i> , <i>delimiter</i>)</code>	<p>Returns the number of fields in <i>cstring</i> delimited by <i>delimiter</i> , where <i>delimiter</i> is a string. For example,</p> <pre>dcount_substring("abcFdefFghi", "F")</pre> <p>returns 3.</p> <p>If <i>delimiter</i> is an empty string, the number of characters in the string + 1 is returned. If <i>delimiter</i> is not empty, but does not exist in the given string, 1 is returned.</p>
<code>cstring u_double_quote_string (<i>expression</i>)</code>	Returns the given <i>cstring</i> <i>expression</i> enclosed in double quotes.

Function	Description
<code>cstring u_substring_by_delimiter (<i>cstring</i>, <i>delimiter</i>, <i>occurrence</i>, <i>numsubstr</i>)</code>	<p>The <i>delimiter</i> argument is a <i>cstring</i> value, and the <i>occurrence</i> and <i>numsubstr</i> arguments are <i>int32</i> values.</p> <p>This function returns <i>numsubstr</i> substrings from <i>cstring</i> , delimited by <i>delimiter</i> and starting at substring number <i>occurrence</i> . An example is:</p> <pre data-bbox="802 418 1390 449">u_substring_by_delimiter ("abcFdefFghiFjkl", "F", 2, 2)</pre> <p>The string "defFghi" is returned.</p> <p>If occurrence is < 1, then 1 is assumed. If <i>occurrence</i> does not point to an existing field, the empty string is returned. If <i>numsubstr</i> is not specified or is less than 1, it defaults to 1.</p>
<code>int32 u_index_of_substring (<i>cstring</i>, <i>sub_cstring</i>, <i>occurrence</i>)</code>	<p>Returns the starting position of the nth occurrence of <i>sub_cstring</i> in <i>cstring</i>. The <i>occurrence</i> argument is an integer indicating the nth <i>occurrence</i> .</p> <p>If there is no nth occurrence, 0 is returned; if <i>sub_cstring</i> is an empty string, -2 is returned; and if <i>cstring</i> doesn't contain any <i>sub_cstring</i> , -1 is returned.</p>
<code>cstring u_left_substring (<i>cstring</i> , <i>length</i>)</code>	<p>Returns the first <i>length</i> characters of <i>cstring</i>. If <i>length</i> is 0, it returns the empty string. If <i>length</i> is greater than the length of the <i>cstring</i> , the entire <i>cstring</i> is returned.</p>
<code>cstring u_right_substring (<i>cstring</i> , <i>length</i>)</code>	<p>Returns the last <i>length</i> characters of <i>cstring</i> . If <i>length</i> is 0, it returns the empty string. If <i>length</i> is greater than the length of <i>cstring</i> , the entire <i>cstring</i> is returned.</p>
<code>cstring u_string_of_space(<i>count</i>)</code>	<p>Returns a <i>cstring</i> containing <i>count</i> spaces. The empty string is returned for a <i>count</i> of 0 or less.</p>
<code>cstring u_single_quote_string (<i>expression</i>)</code>	<p>Returns <i>expression</i> enclosed in single quotes.</p>
<code>cstring u_string_of_substring (<i>cstring</i> , <i>count</i>)</code>	<p>Returns a <i>cstring</i> containing <i>count</i> occurrences of <i>cstring</i>. The empty string is returned for a count of 0 or less.</p>
<code>cstring u_trimc_string (<i>cstring</i> [, <i>character</i> [, <i>option</i>]])</code>	<p>If only <i>cstring</i> is specified, all leading and trailing spaces and tabs are removed, and all multiple occurrences of spaces and tabs are reduced to a single space or tab.</p> <p>If <i>cstring</i> and <i>character</i> are specified, option defaults to 'R' The available option values are:</p> <ul style="list-style-type: none"> 'A' remove all occurrences of character 'B' remove both leading and trailing occurrences of character 'D' remove leading, trailing, and redundant white-space characters 'E' remove trailing white-space characters 'F' remove leading white-space characters 'L' remove all leading occurrences of character 'R' remove all leading, trailing, and redundant occurrences of character 'T' remove all trailing occurrences of character
<code>cstring u_system_time_date()</code>	<p>Returns the current system time in this 24-hour format:</p> <p><i>hh:mm:ss dd:mmm:yyyy</i></p>
<code>int32 u_offset_of_substring (<i>cstring</i> , <i>sub_cstring</i> , <i>position</i>)</code>	<p>Searches for the <i>sub_cstring</i> in the <i>cstring</i> beginning at character number <i>position</i>, where <i>position</i> is an <i>uint32</i>. Returns the starting position of the substring.</p>
<code>int8 u_string_case_compare (<i>cstring</i> , <i>cstring</i>)</code>	<p>This is a case-insensitive version of <i>u_string_compare()</i> below.</p>

Function	Description
int8 u_string_compare (<i>cstring</i> , <i>cstring</i>)	Compares two <i>cstring</i> s and returns the index (0 or 1) of the greater string.
int8 u_string_num_case_compare (<i>cstring</i> , <i>cstring</i> , <i>uint16</i>)	This is a case-insensitive version of <i>u_string_num_compare()</i> below.
cstring u_string_num_concatenate (<i>cstring</i> , <i>cstring</i> , <i>uint16</i>)	Returns a <i>cstring</i> after appending <i>uint16</i> characters from the second <i>cstring</i> onto the first <i>cstring</i> .
int8 u_string_num_compare (<i>utring</i> , <i>cstring</i> , <i>uint16</i>)	Compares first <i>uint16</i> characters of two given <i>cstring</i> s and returns the index (0 or 1) of the greater <i>cstring</i> .
cstring u_string_num_copy (<i>cstring</i> , <i>uint16</i>)	Returns the first <i>uint16</i> characters from the given <i>cstring</i> .
int32 u_string_length(<i>cstring</i>)	Returns the length of the <i>cstring</i> .
cstring u_substring (<i>cstring</i> , <i>starting_position</i> , <i>length</i>)	Copies parts of <i>cstring</i> s to shorter strings by string extraction. The <i>starting_position</i> specifies the starting location of the substring; <i>length</i> specifies the substring length. The arguments <i>starting_position</i> and <i>length</i> are <i>uint16</i> types and must be positive (≥ 0).
cstring u_char_from_num(<i>int32</i>)	Returns a <i>cstring</i> character value from the given <i>int32</i> . If given a value that is not associated with a character such as -1, the function returns a space. An example use is: <i>u_char_from_num(38)</i> which returns "&"
int32 u_num_from_char(<i>cstring</i>)	Returns the numeric value of the character in the <i>cstring</i> . When this function is given an empty string, it returns 0; and when it is given a multi-character string, it uses the first character in the string. An example use is: <i>u_num_from_char("&")</i> which returns 38

Bit manipulation functions

Function	Description
string bit_expand (<i>uint64</i>)	Expands the given <i>uint64</i> to a string containing the binary representation.
cstring u_bit_expand (<i>uint64</i>)	Expands the given <i>uint64</i> to a <i>cstring</i> containing the binary representation.
uint64 bit_compress(<i>string</i>)	Converts the string binary representation to an <i>uint64</i> field.
uint64 u_bit_compress(<i>cstring</i>)	Converts the <i>cstring</i> binary representation to an <i>uint64</i> field.
uint64 set_bit (<i>uint64</i> , <i>list_of_bits</i> , <i>bit_state</i>)	Turns the <i>uint64</i> bits that are listed by number in the string <i>list_of_bits</i> on or off, depending on whether the value of the <i>bit_state</i> integer is 1 or 0. <i>bit_state</i> is an optional argument, and has a default value of 1 which turns the list of bits on. An example use is: <i>set_bit(0, "1,3,5,7")</i> which returns 85.
uint64 u_set_bit (<i>uint64</i> , <i>list_of_bits</i> , <i>bit_state</i>)	This function is a internationalized version of <i>set_bit()</i> above.

Job monitoring functions

The Job Monitor reports on the current state of a job and supplies information about the operators in your data flow. By default, it continually gathers information about your jobs, but it does not send the information unless you request it from your user-written client.

The information it supplies falls into four categories: job status, metadata, monitor, and summary, and it is given in XML notation. If you do not have job monitoring disabled with the top-level -nomonitor osh option, you can also obtain custom report and summary information about the transform operator using the functions in the table below.

There are six functions: three for the string type and three for the ustring type. The name_string or name_ustring argument can be used to specify a name for the custom information, the description_string or description_ustring argument can be used to describe the type of information, and the value_string or value_ustring argument can be used to give the details of the information.

Function	Description
<code>string set_custom_summary_info (name_string , description_string, value_string)</code>	Call this function in the finish code segment.
<code>wstring u_set_custom_summary_info (name_ustring ,description_ustring, value_ustring)</code>	
<code>string send_custom_report (name_string , description_string, value_string)</code>	Call this function in the initialize code segment.
<code>wstring u_send_custom_report (name_ustring , description_ustring , value_ustring)</code>	
<code>string set_custom_instance_report (name_string , description_string, value_string)</code>	Call this function in the mainloop code segment.
<code>wstring u_set_custom_instance_report (name_ustring , description_ustring, value_ustring)</code>	

Miscellaneous functions

The following table describes functions in the Transformation Language that do not fit into any of the above categories.

Function	Description
<code>void force_error (error_message_string)</code>	Terminates the data flow when an error is detected, and prints <code>error_message_string</code> to stderr.
<code>w_force_error (error_message_wstring)</code>	Terminates the data flow when an error is detected, and prints <code>error_message_wstring</code> to stderr.
<code>string get_environment(string)</code>	Returns the current value of <code>string</code> , a UNIX environment variable. The functionality is the same as the C <code>getenv</code> function.
<code>wstring u_get_environment (wstring)</code>	Returns the current value of <code>wstring</code> , a UNIX environment variable. The functionality is the same as the C <code>getenv</code> function.
<code>int16 get_partition_num()</code>	Returns the current partition number.
<code>int16 get_num_of_partitions()</code>	Returns the number of partitions.
<code>void print_message (message_string)</code>	Prints <code>message_string</code> to stdout.

Function	Description
<code>u_print_message (message_ustring)</code>	Prints <code>message_ustring</code> to stdout.
<code>uint32 size_of(value)</code>	Returns the actual size <code>value</code> when it is stored, not the length.

binary operators with decimal and numeric fields

The Transformation Language supports the placement of binary operators between two decimals and between a decimal and one of the numeric field types.

The binary operators are +, -, *, and /.

The numeric field types are int8, uint8, int16, uint16, int32, uint32, int64, uint64, sfloat, and dfloat.

Generally there are no restrictions on the form of an operand. It can be a constant, a variable, a function call that returns a numeric value, a simple expression such as an addition of two variables, or a complicated expression that consists of function calls as well as arithmetic operations.

By default, the transform operator sets the precision and scale of any temporary internal decimal variable created during arithmetic operations to:

```
[TRX_DEFAULT_MAX_PRECISION=38,  
TRX_DEFAULT_MAX_SCALE=10]  
with RoundMode equal to eRoundInf.
```

You can override the default values using these environment variables:

- APT_DECIMAL_INTERM_PRECISION *value*
- APT_DECIMAL_INTERM_SCALE *value*
- APT_DECIMAL_INTERM_ROUNDMODE ceil | floor | round_inf | trunc_zero

Fatal errors might occur at runtime if the precision of the destination decimal is smaller than that of the source decimal

The transformation language versus C

The Transformation Language contains additions to C to handle record processing. Also, parts of C are not supported. The following sections indicated C language items that are not supported by the Transformation Language or that work differently in the Transformation Language.

Keywords

Keywords in the Transformation Language are not case sensitive.

Local variables

You cannot initialize variables as part of the declaration. There are no local variable pointers or structures. Enums are not supported.

Operators

Several C operators are not part of the language:

- The comma operator
- Composite assignment operators such as +=.

Flow control

The switch and case C keywords do not appear in the Transformation Language.

The if ... else if construct for multiple branching is not supported. You can accomplish the same effect by using multiple if statements in sequence together with complex boolean expressions. For example, where in C you could write:

```
if (size < 7)
    tag = "S";
else if (size < 9)
    tag = "M";
else
    tag = "L";
```

in the Transformation Language you would write:

```
if (size < 7)
    tag = "S";
if (size >= 7 && size < 9)
    tag = "M";
if (size >= 9)
    tag = "L";
```

Other unsupported C language elements

The Transformation Language does not support:

- Casting
- Labeled statements
- Pre-Processor commands

Using the transform operator

This section gives examples on how to construct data transformations using the transform operator.

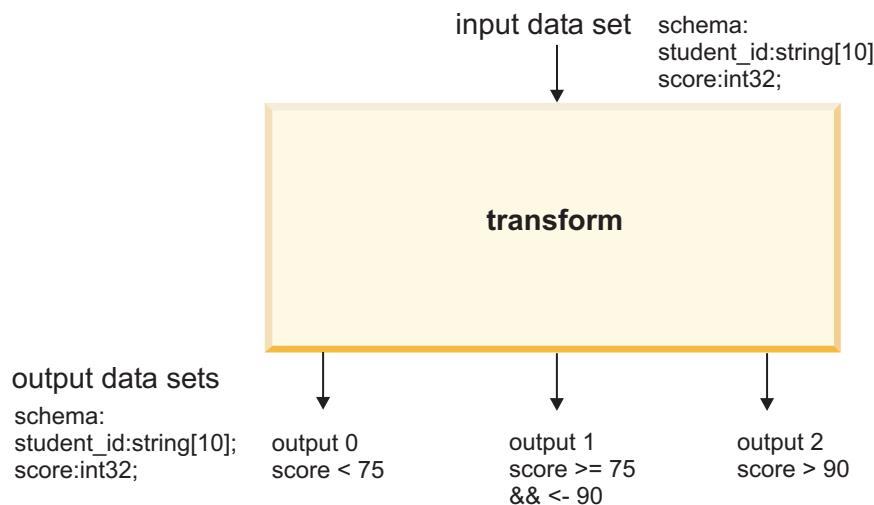
Example 1: student-score distribution

Job logic

In this example, the transform operator is used to determine the student-score distribution for multiple classes. The input data set contains a record for each student in the classes. Each record contains a student_id field and a score field. The score field has the numerical score the student received for the class.

The input student records are distributed to one of three output data sets based on the value of the score field. Records with a score of under 75 (poor) are written to output 0; records with a score between 75 and 90 (fair) are written to output 1; and records with a score of over 90 (good) are written to output 2.

Data flow diagram



After the records are processed, a report is printed that shows the number of students in each score category.

Highlighted transformation language components

This section points out the Transformation Language components this example illustrates. The Transformation Language code is given in the next section.

- Code segmentation. The use of the `global{}}, initialize{}}, mainloop{}}, and finish{}} segments.`
- Global variables. How to declare global variables and where to declare them. There is a single global variable: `jobName`.
- Stage variables. How to declare stage variables and where to declare them. Examples are `recNum0`, `recNum1`, and `recNum2`.
- Record flow controls. How to use the conditional statement with the `writerecord` command.
- Local variables. How to declare and initialize local variables. Examples from the code are `numOfPoorScores`, `numOfFairScores`, and `numOfGoodScores`.
- Default conversions. How to do default conversions using the assignment operator. An example that converts an `int32` to a `string` is:
- `numOfGoodScore=recNum2`
- Message logging. How to make function calls to `print_message()`.

Transformation language

The record-processing logic can be expressed in the Transformation Language as follows. The expression file name for this example is `score_distr_expr`.

```
global
{
    // the global variable that contains the name for each job run
    string jobName;
}
initialize
{
    // the number of records in output 0
    int32 recNum0;

    // the number of records in output 1
    int32 recNum1;

    // the number of records in output 2
    int32 recNum2;
```

```

// initialization
recNum0 = 0;
recNum1 = 0;
recNum2 = 0;
}
mainloop
{
    // records in output 0
    if (score < 75)
    {
        recNum0++;
        writerecord 0;
    }
    // records of output 1
    if (score >= 75 && score <= 90)
    {
        RecNum1++;
        writerecord 1;
    }
    // records of output2
    if (score > 90)
    {
        recNum2++;
        writerecord 2;
    }
}
finish
{
    // define a string local variable to store the number of
    // students with poor scores
    string NumOfPoorScores;
    numOfPoorScores = recNum0;
    // define a string local variable to store the number of
    // students with fair scores
    string NumOfFairScores;
    numOffairScores = recNum1;
    // define a string local variable to store the number of
    // students with good scores
    string NumOfGoodScores;
    numOfGoodScores = recNum2;
    // Print out the number of records in each output data set
    print_message(jobName+ "has finished running.");
    print_message("The number of students having poor scores are"
+numOfPoorScores);
    print_message("The number of students having fair scores are"
+numOfFairScores);
    print_message("The number of students having good scores are"
+numOfGoodScores);
}

```

osh command

An example osh command to run this job is:

```
osh -params "jobName=classA" -f score_distr
```

osh script

The contents of score_distr are:

```
#compile the expression code
transform -inputschema record(student_id:string[10];score:int32;)
-outputschema record(student_id:string[10];score:int32;)
-outputschema record(student_id:string[10];score:int32;)
-outputschema record(student_id:string[10];score:int32;)
-expressionfile score_distr_expr -flag compile -name score_map;
#run the job
```

```

import -schema record(student_id:string[10];score:int32)
  -file [&jobName].txt | transform -flag run -name score_map
0> export -schema record(student_id:string[10];score:int32)
    -filename [&jobName]poor_score.out -overwrite
1> -export -schema record(student_id:string[10];score:int32)
    -file [&jobName]fair_score.out -overwrite
2> -export -schema record(student_id:string[10];score:int32)
    -file [&jobName]good_score.out -overwrite

```

Example input and output

The input from classA.txt is:

```

A112387567 80
A218925316 95
A619846201 70
A731347820 75
A897382711 85
A327637289 82
A238950561 92
A238967521 87
A826381931 66
A763567100 89

```

The outputs are:

classApoor_score.out:

```

A619846201 70
A826381931 66

```

classAfair_score.out:

```

A112387567 80
A731347820 75
A897382711 85
A327637289 82
A238967521 87
A763567100 89

```

classAgood_score.out:

```

A218925316 95
A238950561 92

```

The global variable jobName is initialized using the -params option. To determine the score distribution for class B, for example, assign jobName another value:

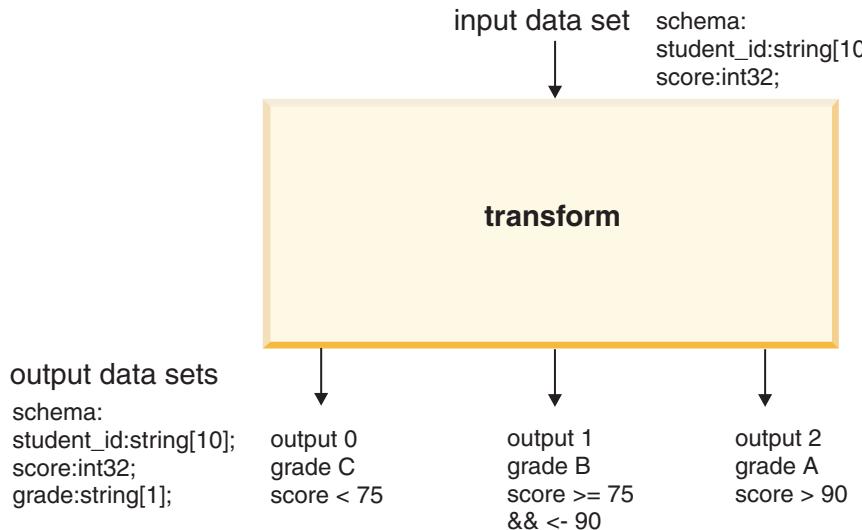
```
osh -params "jobName=classB" -f score_distr
```

Example 2: student-score distribution with a letter grade added to example

Job logic

The job logic in this example is similar to that in Example 1. In addition to the student_id and score fields, an additional field indicating the letter grade is added to each output record. The letter grade is based on the value of the score field: grade C for a score of under 75; grade B for a score between 75 and 90; and grade A for a score over 90.

Data flow diagram



Highlighted transformation language components

This example demonstrates the use of the following components. The Transformation Language code is given in the next section.

- data set alias. How to define and use data set aliases.
- Stage variables. Declare stage variables in a row. An example is:
int32 recNum0, recNum1, recNum2;
- String constants. How to assign a constant to a string variable. For example:
out0.grade = "C"
- Local variables. How to declare local variables in a row. For example:
int32 numOfCs, numOfBs, numOfAs

Transformation language

The record-processing logic can be expressed in the Transformation Language as follows. The expression file name for this example is score_grade_expr.

```

inputname 0 in0;
outputname 0 out0;
outputname 1 out1;
outputname 2 out2;
global
{
    // the global variable that contains the name for each job run
    string jobName;
}
initialize
{
    // the number of records in the outputs
    int32 recNum0, recNum1, recNum2;

    // initialization
    recNum0 = 0;
    recNum1 = 0;
    recNum2 = 0;
}
mainloop
{
    if (in0.score < 75)

```

```

    {
        recNum0++;
        out0.grade = "C";
        writerecord 0;
    }
    if (in0.score >= 75 && in0.score <= 90)
    {
        recNum1++;
        out1.grade = "B";
        writerecord 1;
    }
    if (in0.score > 90)
    {
        recNum2++;
        out2.grade = "A";
        writerecord 2;
    }
}
finish
{
    // define string local variables to store the number of
    // students having different letter grades
    string num0fCs, num0fBs, num0fAs;
    // default conversions using assignments
    num0fCs = recNum0;
    num0fBs = recNum1;
    num0fAs = recNum2;
    // Print out the number of records in each output data set
    print_message(jobName+ " has finished running.");
    print_message("The number of students getting C is "
        +num0fCs);
    print_message("The number of students getting B is "
        +num0fBs);
    print_message("The number of students getting A is "
        +num0fAs);
}

```

osh command

An example osh command to run this job is:

```
osh -params "jobName=classA" -f score_grade
```

osh script

The contents of score_grade are:

```

#compile the expression code
transform -inputschema record(student_id:string[10];score:int32;)
-outputschema record
    (student_id:string[10];score:int32;grade:string[1])
-outputschema record
    (student_id:string[10];score:int32;grade:string[1])
-outputschema record
    (student_id:string[10];score:int32;grade:string[1])
-expressionfile score_grade_expr -flag compile -name score_map;
#run the job
import -schema record(student_id:string[10];score:int32;)
-file [&jobName].txt | transform -flag run -name score_map
0> -export record(student_id:string[10];score:int32;grade:string[1])
    -file [&jobName]poor_scores.out -overwrite
1> -export record(student_id:string[10];score:int32;grade:string[1])
    -file [&jobName]fair_scores.out -overwrite
2> -export record(student_id:string[10];score:int32;grade:string[1])
    -file [&jobName]good_scores.out -overwrite

```

Example input and output

The input from classA.txt is the same as in "Example 1: Student-Score Distribution" .

The outputs are:

classApoor_scores.out

```
A619846201 70 C  
A826381931 66 C
```

classAfair_scores.out

```
A112387567 80 B  
A731347820 75 B  
A897382711 85 B  
A327637289 82 B  
A238967521 87 B  
A763567100 89 B
```

classAgood_scores.out

```
A218925316 95 A  
A238950561 92 A
```

Example 3: student-score distribution with a class field added to example

Job logic

The job logic in this example is similar to that in "Example 2: Student-Score Distribution With a Letter Grade Added to Example 1" . The difference is that another output field is added, named class.

The student class is determined by the first character of the student_id. If the first character is B, the student's class is Beginner; if the first character is I, the student's class is Intermediate; and if the first character is A, the student's class is Advanced. Records with the same class field are written to the same output.

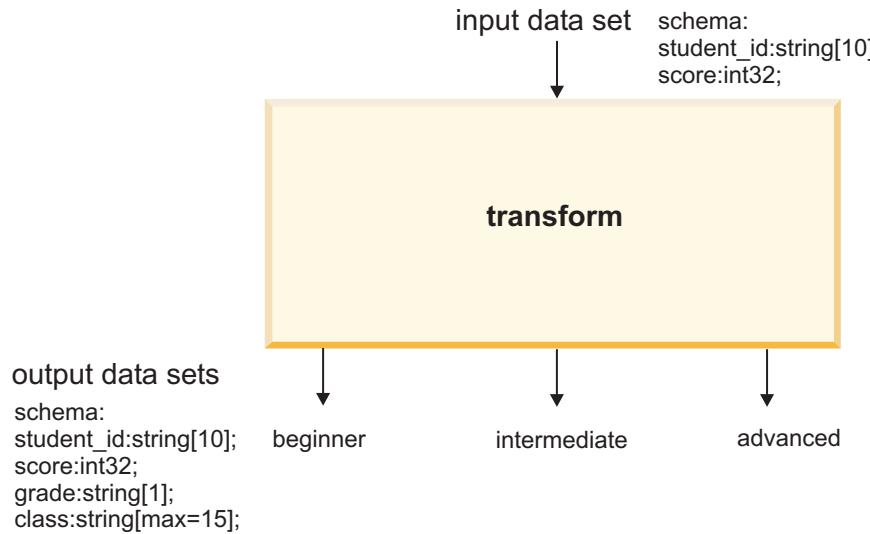
The score field is not only transferred from input to output, but is also changed. If the score field is less than 75, the output score is:

```
(in0.score+(200-in0.score*2)/4)
```

Otherwise the output score is:

```
(in0.score+(200-in0.score*2)/3)
```

Data flow diagram



Highlighted transformation language components

This example demonstrates the use of the following components. The Transformation Language code follows in the next section.

- Binary and ternary operators. How to use them in expressions. An example is: score_local.
- String manipulations. How to do string comparisons, string assignments, and function calls. Examples are: class_local and class_init.
- Local variables. How to use local variables in expressions. Examples are score_local, grade_local, and class_local.

Transformation language

The record-processing logic can be expressed in the Transformation Language as follows. The expression file for this example is score_class_expr.

```

inputname 0 int0;
outputname 0 out0;
outputname 1 out1;
outputname 2 out2;
mainloop
{
    //define an int32 local variable to store the score
    int32 score_local;
    score_local=(in0.score < 75) ? (in0.score+(200-in0.score*2)/4):
        (in0.score+(200-in0.score*2)/3)
    // define a string local variable to store the grade
    string[1] grade_local;
    if (score_local < 75) grade_local = "C";
    if (score_local >= 75 && score_local <= 90) grade_local = "B";
    if (score_local > 90) grade_local = "A";
    // define string local variables to check the class level
    string[max=15] class_local;
    string[1] class_init;
    class_init = substring(in0.student_id,0,1);
    if (class_init == "B") class_local = "Beginner";
    if (class_init == "I") class_local = "Intermediate";
    if (class_init == "A") class_local = "Advanced";
    // outputs
    if (class_local == "Beginner")
    {
        out0.score = score_local;
    }
}

```

```

        out0.grade = grade_local;
        out0.class = class_local;
        writerecord 0;
    }
    if (class_local == "Intermediate")
    {
        out1.score = score_local;
        out1.grade = grade_local;
        out1.class = class_local;
        writerecord 1;
    }
    if (class_local == "Advanced")
    {
        out2.score = score_local;
        out2.grade = grade_local;
        out2.class = class_local;
        writerecord 2;
    }
}

```

osh command

The osh command to run this job is:

```
osh -f score_class
```

osh script

The contents of score_class are:

```

#compile the expression code
transform -inputschema record(student_id:string[10];score:int32;)
-outputschema record
  (student_id:string[10];score:int32;grade:string[1];
   class:string[max=15])
-outputschema record
  (student_id:string[10];score:int32;grade:string[1];
   class:string[max=15])
-outputschema record
  (student_id:string[10];score:int32;grade:string[1];
   class:string[max=15])
-expressionfile score_class_expr -flag compile -name score_map;
#run the job
import -schema record(student_id:string[10];score:int32;
  -file score.txt | transform -flag run -name score_map
0> -export record(student_id:string[10];score:int32;grade:string[1];
  class:string[max=15]) -file beginner.out -overwrite
1> -export record(student_id:string[10];score:int32;grade:string[1];
  class:string[max=15]) -filename intermediate.out -overwrite
2> -export record(student_id:string[10];score:int32;grade:string[1];
  class:string[max=15]) -file advanced.out -overwrite

```

Example input and output

The input from score.txt is

```
B112387567 80
A218925316 95
A619846201 70
I731347820 75
B897382711 85
I327637289 82
A238950561 92
I238967521 87
B826381931 66
A763567100 89
```

The outputs are:

beginner.out

```
B112387567 93 A Beginner
B897382711 95 A Beginner
B826381931 83 B Beginner
```

intermediate.out

```
I731347820 91 A Intermediate
I327637289 94 A Intermediate
I238967521 95 A Intermediate
```

advanced.out

```
A218925316 98 A Advanced
A619846201 85 B Advanced
A238950561 97 A Advanced
A763567100 96 A Advanced
```

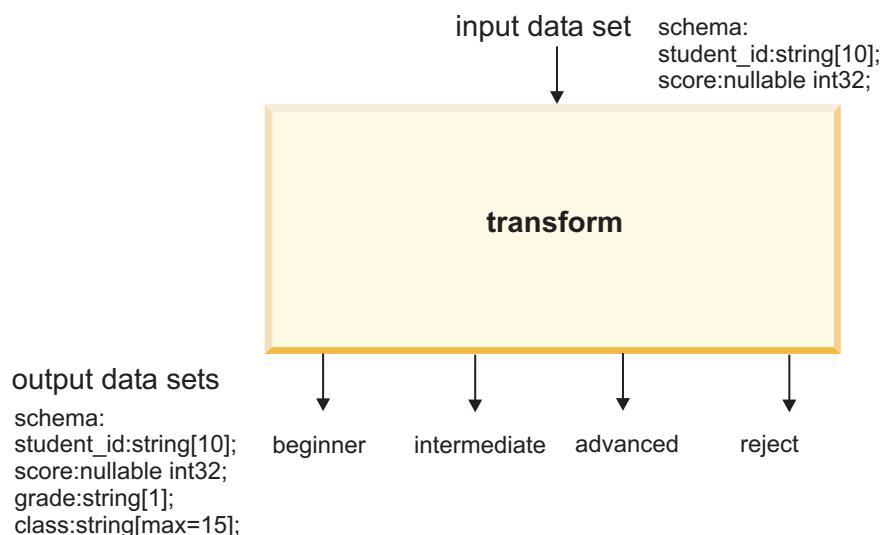
Example 4. student record distribution with null score values and a reject

Job logic

The job logic in this example is the same as in "Example 3: Student-Score Distribution with a Class Field Added to Example 2". The difference is that the input records contain null score fields, and records with null score fields are transferred to a reject data set. This example shows you how to specify and use a reject data set.

The Transformation Language expression file is the same as that for Example 3.

Data flow diagram



osh command

The osh command to run this job is:

```
osh -f score_reject
```

osh script

The contents of score_reject are:

```
#compile the expression code
transform -inputschema record(student_id:string[10];
    score:nullable int32;
-outputschema record
    (student_id:string[10];score:nullable int32;grade:string[1];
    class:string[max=15])
-outputschema record
    (student_id:string[10];score:nullable int32;grade:string[1];
    class:string[max=15])
-outputschema record
    (student_id:string[10];score:nullable int32;grade:string[1];
    class:string[max=15])
-expressionfile score_class_expr -flag compile -name score_map
    -reject;
#run the job
import -schema record(student_id:string[10];score:nullable int32
    {null_field='NULL'}) -file score_null.txt | transform -flag run
    -name score_map
0> -export -schema record(student_id:string[10];
    score:nullable int32{null_field='NULL'}; grade:string[1];
    class:string[max=15]) -file beginner.out -overwrite
1> -export record(student_id:string[10];
    score:nullable int32{null_field='NULL'}; grade:string[1];
    class:string[max=15]) -file intermediate.out -overwrite
2> -export record(student_id:string[10];
    score:nullable int32{null_field='NULL'}; grade:string[1];
    class:string[max=15]) -file advanced.out -overwrite
3> -export record(student_id:string[10];
    score:nullable int32{null_field='NULL'}); -file reject.out -overwrite
```

Example input and output

The input from score_null.txt is

```
B112387567 NULL
I218925316 95
A619846201 70
I731347820 75
B897382711 85
I327637289 NULL
A238950561 92
I238967521 87
B826381931 66
A763567100 NULL
```

The outputs are

beginner.out

```
B897382711 95 A Beginner
B826381931 83 B Beginner
```

intermediate.out

```
I218925316 98 A Intermediate
I731347820 91 A Intermediate
I238967521 95 A Intermediate
```

advanced.out

```
A619846201 85 B Advanced
A238950561 97 A Advanced
```

```

reject.out
B112387567 NULL
I327637289 NULL
A763567100 NULL

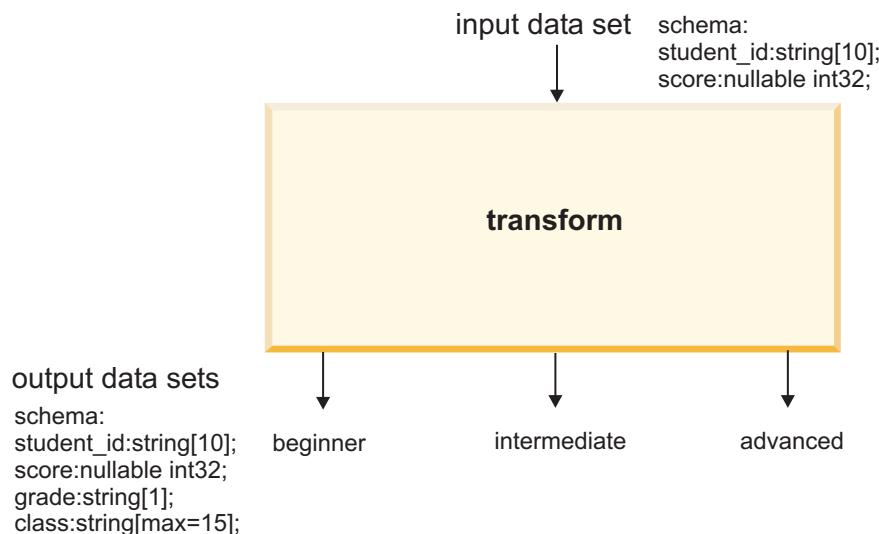
```

Example 5. student record distribution with null score values handled

Job logic

The job logic in this example is the same as "Example 4. Student Record Distribution With Null Score Values And a Reject Data Set". The difference is that null values are appropriately handled so that records with null score fields are not sent to a reject data set.

Data flow diagram



Highlighted transformation language components

This example shows how to handle nulls in input fields, and how to set nulls on output fields. The Transformation Language is in the next section.

The input score field is assigned to the local variable score_local. Since, by default, the score field is nullable and score_local is non-nullable, any record containing a null score is either dropped or sent to the reject data set. To avoid this and to insure that all records go to their appropriate outputs, null handling is used.

There are three ways to handle nulls:

1. Use a function call to handle_null(). For example:
`score_local = handle_null(in0.score, "-100");`
2. Use a function call to null() and the ternary operator. For example:
`score_local = null(in0.score)?-100:in0.score;`
3. Use a function call to notnull() and the ternary operator. For example:
`score_local = notnull(in0.score)?in0.score:-100;`

Setting a nullable field to null occurs when the record is written to output. In this example, after the statement `out0.score = score_local` executes, `out0.score` no longer contains null values, since the old null

value has been replaced by -100. To reinstate the null field, the function make_null() is called to mark that any field containing -100 is a null field. When the score is -100, grade should be null; therefore, set_null() is called.

Transformation language

The record processing logic can be expressed in the Transformation Language as follows. The expression file name for this example is score_null_handling_expr.

```
inputname 0 in0;
outputname 0 out0;
outputname 1 out1;
outputname 2 out2;
mainloop
{
// define an int32 local variable to store the score
int32 score_local;
// handle the null score
score_local = handle_null(in0.score,"-100");
// alternatives:
// score_local = null(in0.score)?-100:in0.score;
// score_local = notnull(in0.score)?in0.score:-100;
// define a string local variable to store the grade
string[1] grade_local;
grade_local = "F";
if ( score_local < 60 && score_local >= 0 )
    grade_local = "D";
if ( score_local < 75 && score_local >= 60 )
    grade_local = "C";
if ( score_local >= 75 && score_local <= 90 )
    grade_local = "B";
if ( score_local > 90 )
    grade_local = "A";
// define string local variables to check the class level
string[max=15] class_local;
string[1] class_init;
    class_init = substring(in0.student_id,0,1);
    if ( class_init == "B" )
        class_local = "Beginner";
    if ( class_init == "I" )
        class_local = "Intermediate";
    if ( class_init == "A" )
        class_local = "Advanced";
// outputs
if ( class_local == "Beginner" )
{
    out0.score = score_local;
    out0.score = make_null(out0.score,"-100");
    if ( grade_local == "F" )
        out0.grade = set_null();
    else
        out0.grade = grade_local;
    out0.class = class_local;
    writerecord 0;
}
if ( class_local == "Intermediate" )
{
    out1.score = score_local;
    out1.score = make_null(out1.score,"-100");
    if ( grade_local == "F" )
        out1.grade = set_null();
    else
        out1.grade = grade_local;
    out1.class = class_local;
    writerecord 1;
}
if ( class_local == "Advanced" )
```

```

    {
        out2.score = score_local;
        out2.score = make_null(out2.score,"-100");
        if ( grade_local == "F" )
            Sout2.grade = set_null();
        else
            out2.grade = grade_local;
        out2.class = class_local;
        writerecord 2;
    }
}

```

osh command

The osh script to run this job is:

```
osh -f score_null_handling
```

osh script

The contents of score_null_handling are:

```

# compile the expression code
transform -inputschema record(student_id:string[10];score:nullable int32;)
-outputschema record(student_id:string[10];score:nullable      int32;grade:nullable
string[1];class:string[max=15])
-outputschema record(student_id:string[10];score:nullable      int32;grade:nullable
string[1];class:string[max=15])
-outputschema record(student_id:string[10];score:nullable      int32;grade:nullable
string[1];class:string[max=15])
-expressionfile score_null_handling_expr -flag compile
    -name score_map;
# run the job
import -schema record(student_id:string[10];score:nullable      int32{null_field='NULL'}) -file
score_null.txt | transform -flag run -name score_map
0> -export record(student_id:string[10];score:nullable      int32{null_field='NULL'};grade:nullable
string[1]{null_field='FAILED'};class:string[max=5];)
    -file beginner.out -overwrite
1> -export record(student_id:string[10];score:nullable      int32{null_field='NULL'};grade:nullable
string[1]{null_field='FAILED'};class:string[max=5];)
    -file intermediate.out -overwrite
2> -export record(student_id:string[10];score:nullable      int32{null_field='NULL'};grade:nullable
string[1]{null_field='FAILED'};class:string[max=5];)
    -file advanced.out -overwrite

```

Example input and output

The input from score_null.txt is:

```
B112387567 NULL
I218925316 95
A619846201 70
I731347820 75
B897382711 85
I327637289 NULL
A238950561 92
I238967521 87
B826381931 66
A763567100 NULL
```

The outputs are:

beginner.out

```
B112387567 NULL FAILED Beginner
B897382711 85 B Beginner
B826381931 66 C Beginner
```

intermediate.out

```
I218925316 95 A Intermediate
I731347820 75 B Intermediate
I327637289 NULL FAILED Intermediate
I238967521 87 B Intermediate
```

advanced.out

```
A619846201 70 C Advanced
A238950561 92 A Advanced
A763567100 NULL FAILED Advanced
```

Example 6. student record distribution with vector manipulation

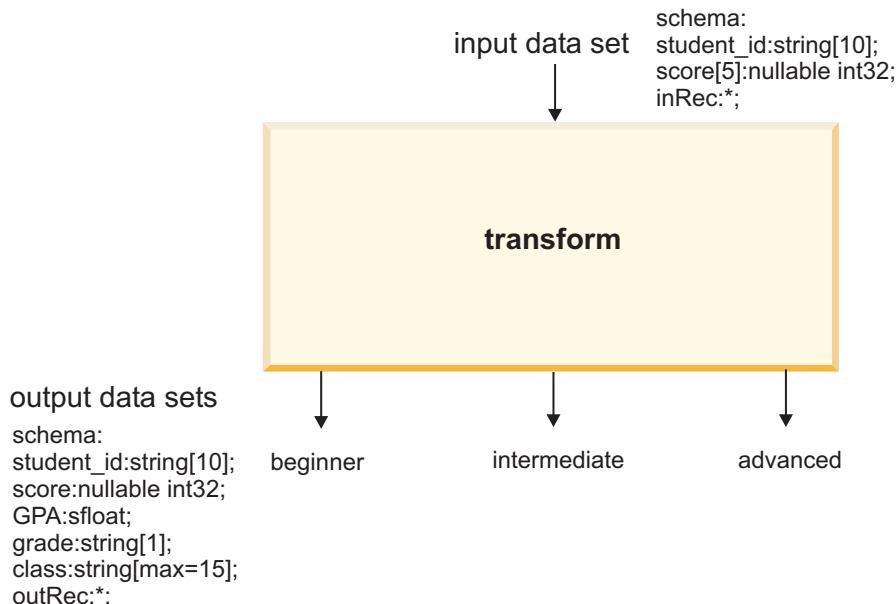
Job logic

The job logic in this example is similar to that in "Example 5. Student Record Distribution With Null Score Values Handled". As in Example 5, this example has two fields in its input schema: `student_id` and `score`. The `score` field, however, is not a single integer field as in Example 5 but is a vector of five elements which represent the class scores a student has received in a single semester. Only three out of five classes are required, therefore, the vector elements are specified to be nullable so that a `NONE` score can represent an elective course not taken by the student, rather than a failing grade as in Example 5.

In addition to the explicit input schema containing the `score` and `student_id` fields, the `transform` operator in this example receives a `term` string field from its up-stream operator via a schema variable. The `term` field indicates whether the semester is a final or a midterm semester.

The output field `grade_local` is a vector of five elements, and a `GPA` float field is added to the output data set.

Data flow diagram



Highlighted transformation language components

This example highlights the following components:

- Vector variable definition. Examples are:

```
int32 score_local[5], and string[1] grade_local[5]
```

- Vector variables with flow loops. For example, while and for loops.
- Using flow loops with controls. For example, continue statements.
- Using vector variables in expressions. For example, if statements and assignments.
- Using a schema variable for an implicit transfer. For example in -inputschema and -outputschema declarations, the field term is transferred from the up-stream operator to the transform operator and from the transform input to its output.

Transformation language

The record-processing logic can be expressed in the Transformation Language as follows. The expression file name for this example is score_vector_expr.

```

inputname 0 in0;
outputname 0 out0;
outputname 1 out1;
outputname 2 out2;
mainloop
{
    // define an int32 local vector variable to store the scores
    int32 score_local[5];
    int32 vecLen;
    vecLen = 5;
    // handle null score
    int32 i;
    i = 0;
    while ( i < vecLen )
    {
        score_local[i] = handle_null(in0.score[i],"-100");
        // alternatives
        // score_local[i] = null(in0.score[i])?-100:in0.score[i];
        // score_local[i] = notnull(in0.score[i])?in0.score[i]:-100;

        i++;
    }
    // define a string local vector variable to store the grades
    string[1] grade_local[5];
    // define sfloat local variables to calculate GPA.
    sfloat tGPA_local, GPA_local;
    tGPA_local = 0.0;
    GPA_local = 0.0;
    // define an int8 to count the number of courses taken.
    int8 num0fScore;
    num0fScore = 0;
    for ( i = 0; i < vecLen; i++)
    {
        // Null score means the course is not taken,
        // and will not be counted.
        if ( score_local[i] == -100)
        {
            grade_local[i] = "S";
            continue;
        }
        num0fScore++;
        if ( score_local[i] < 60 && score_local[i] >= 0 )
        {
            grade_local[i] = "D";
            tGPA_local = tGPA_local + 1.0;
        }
        if ( score_local[i] < 75 && score_local[i] >= 60 )
        {
            grade_local[i] = "C";
            tGPA_local = tGPA_local + 2.0;
        }
        if ( score_local[i] >= 75 && score_local[i] <= 90 )
        {

```

```

        grade_local[i] = "B";
        tGPA_local = tGPA_local + 3.0;
    }
    if ( score_local[i] > 90 )
    {
        grade_local[i] = "A";
        tGPA_local = tGPA_local + 4.0;
    }
}
if ( numOfScore > 0 )
    GPA_local = tGPA_local / numOfScore;
// define string local variables to check the class level
string[max=15] class_local;
string[1] class_init;
class_init = substring(in0.student_id,0,1);
if ( class_init == "B" )
    class_local = "Beginner";
if ( class_init == "I" )
    class_local = "Intermediate";
if ( class_init == "A" )
    class_local = "Advanced";
// outputs
if (class_local == "Beginner")
{
    for ( i = 0; i < vecLen; i++)
    {
        out0.score[i] = score_local[i];
        out0.score[i] = make_null(out0.score[i],"-100");
        out0.grade[i] = grade_local[i];
    }
    out0.GPA = GPA_local;
    out0.class = class_local;
    writerecord 0;
}
if ( class_local == "Intermediate" )
{
    for ( i = 0; i < vecLen; i++)
    {
        out1.score[i] = score_local[i];
        out1.score[i] = make_null(out1.score[i],"-100");
        out1.grade[i] = grade_local[i];
    }
    out1.GPA = GPA_local;
    out1.class = class_local;
    writerecord 1;
}
if ( class_local == "Advanced" )
{
    for ( i = 0; i < vecLen; i++)
    {
        out2.score[i] = score_local[i];
        out2.score[i] = make_null(out2.score[i],"-100");
        out2.grade[i] = grade_local[i];
    }
    out2.GPA = GPA_local;
    out2.class = class_local;
    writerecord 2;
}
}

```

osh command

The osh script to run this job is:

```
osh -f score_vector
```

osh script

The contents of score_vector are:

```
# compile the expression code
transform -inputschema record(student_id:string[10];score[5]:nullable int32;inRec:*) -outputschema
record(student_id:string[10];score[5]:nullable int32;
grade[5]:string[1];GPA:sfloat;class:string[max=15];outRec:*) -outputschema
record(student_id:string[10];score[5]:nullable int32;
grade[5]:string[1];GPA:sfloat;class:string[max=15];outRec:*)
-outputschema record(student_id:string[10];score[5]:nullable int32;
grade[5]:string[1];GPA:sfloat;class:string[max=15];outRec:*) -expressionfile score_vector_expr -flag
compile -name score_map;
# run the job
import -schema record(student_id:string[10];score[5]:nullable int32{null_field='NULL'};term:string) -
file score_vector.txt | transform -flag run -name score_map
0> -export record(student_id:string[10];score[5]:nullable
int32{null_field='NULL'};grade[5]:string[1];GPA:sfloat{out_format=
'%4.2g'};class:string[max=15];term:string;) -file beginner.out -overwrite
1> -export record(student_id:string[10];score[5]:nullable
int32{null_field='NULL'};grade[5]:string[1];GPA:sfloat{out_format='%4.2g'};class:string[max=15];ter
m:string;) -file intermediate.out
-overwrite
2> -export record(student_id:string[10];score[5]:nullable
int32{null_field='NULL'};grade[5]:string[1];GPA:sfloat{out_format='%4.2g'};class:string[max=15];ter
m:string;) -file advanced.out -overwrite
```

Example input and output

The input from score_vector.txt is:

```
B112387567 NULL 90 87 62 NULL Final
I218925316 95 NULL 91 88 NULL Midterm
A619846201 70 82 85 68 NULL Final
I731347820 75 NULL 89 93 95 Final
B897382711 85 90 96 NULL NULL Midterm
I327637289 NULL NULL 88 92 76 Final
A238950561 92 97 89 85 83 Final
I238967521 87 NULL 86 NULL 82 Midterm
B826381931 66 73 82 NULL NULL Midterm
A763567100 NULL NULL 53 68 92 Final
```

The outputs are:

beginner.out

```
B112387567 NULL 90 87 62 NULL S B B C S 2.7 Beginner Final
B897382711 85 90 96 NULL NULL B B A S S 3.3 Beginner Midterm
B826381931 66 73 82 NULL NULL C C B S S 2.3 Beginner Midterm
```

intermediate.out

```
I218925316 95 NULL 91 88 NULL A S A B S 3.7 Intermediate Midterm
I731347820 75 NULL 89 93 95 B S B A A 3.5 Intermediate Final
I327637289 NULL NULL 88 92 76 S S B A B 3.3 Intermediate Final
I238967521 87 NULL 86 NULL 82 B S B S B 3 Intermediate Midterm
```

advanced.out

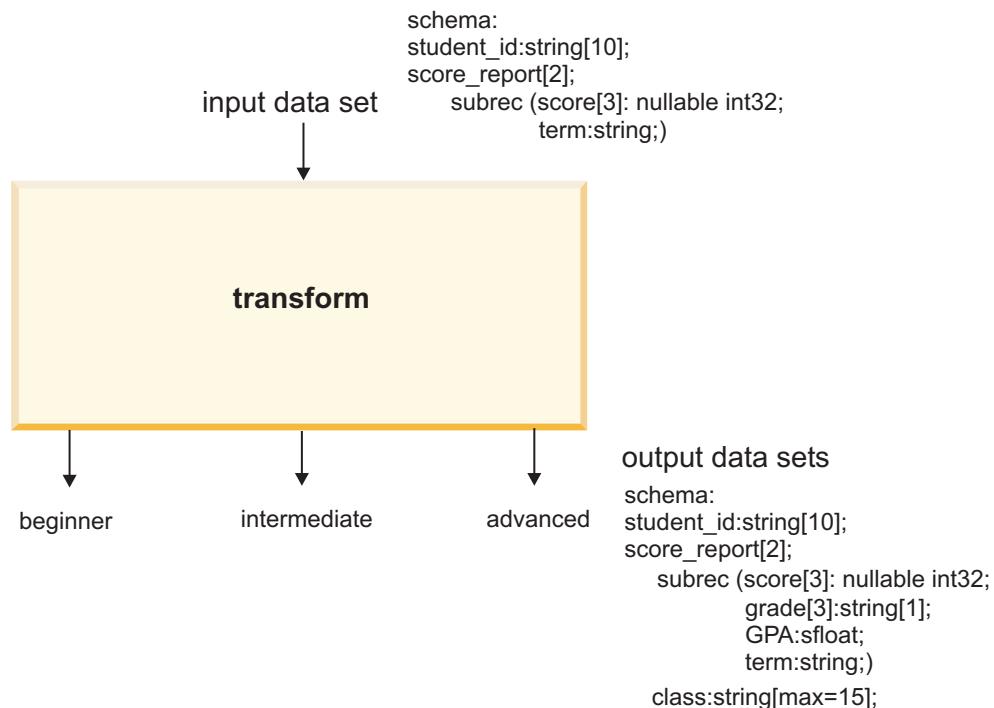
```
A619846201 70 82 85 68 NULL C B B C S 2.5 Advanced Final
A238950561 92 97 89 85 83 A A B B B 3.4 Advanced Final
A763567100 NULL NULL 53 68 92 S S D C A 2.3 Advanced Final
```

Example 7: student record distribution using sub-record

Job logic

The job logic in this example is similar to Example 6. The difference is that the input score_report field is a vector of 2-element sub-records. The schema of each sub-record is the same as the top-level record in Example 6 except that the subfield score is a vector of three elements. The sub-fields are student_id, score, and term. A new output sub-field, GPA, is added to each sub-record. A new output field, class, is added to the top-level record. No null-handling functions are used in this example because null-handling is not applicable to sub-records.

Data flow diagram



Highlighted transformation language component

This example shows you how to use sub-records in expressions.

Transformation language

The record-processing logic can be expressed in the Transformation Language as follows. The expression file name for this example is score_subrec_expr.

```
inputname 0 in0;
outputname 0 out0;
outputname 1 out1;
outputname 2 out2;
mainloop
{
    // define an int32 local vector variable to store the scores
    int32 score_local[3];
    // define an index to store vector or subrec length
    int32 vecLen, subrecLen;
    vecLen = 3;
    subrecLen = 2;
    // index
    int32 i, j;
```

```

i = 0;
j = 0;
// define a string local vector variable to store the grades
string[1] grade_local[3];
// define sfloat local variables to calculate GPA.
sfloat tGPA_local, GPA_local;
tGPA_local = 0.0;
GPA_local = 0.0;
// define string local variables to check the class level
string[1] class_init;
string[max=15] class_local;
class_init = substring(in0.student_id,0,1);
if ( class_init == "B" )
    class_local = "Beginner";
if ( class_init == "I" )
    class_local = "Intermediate";
if ( class_init == "A" )
    class_local = "Advanced";
// calculate grade and GPA
// The outer loop controls subrec
// The inner loop controls sub-fields
for ( j = 0; j < subrecLen; j++)
{
    for ( i = 0; i < vecLen; i++)
    {
        score_local[i] = in0.score_report[j].score[i];
        if ( score_local[i] < 60 && score_local[i] >= 0 )
        {
            grade_local[i] = "D";
            tGPA_local = tGPA_local + 1.0;
        }
        if ( score_local[i] < 75 && score_local[i] >= 60 )
        {
            grade_local[i] = "C";
            tGPA_local = tGPA_local + 2.0;
        }
        if ( score_local[i] >= 75 && score_local[i] <= 90 )
        {
            grade_local[i] = "B";
            tGPA_local = tGPA_local + 3.0;
        }
        if ( score_local[i] > 90 )
        {
            grade_local[i] = "A";
            tGPA_local = tGPA_local + 4.0;
        }
    }
    GPA_local = tGPA_local / vecLen;
    // outputs
    if ( class_local == "Beginner" )
    {
        for ( i = 0; i < vecLen; i++)
        {
            out0.score_report[j].score[i] = score_local[i];
            out0.score_report[j].grade[i] = grade_local[i];
        }
        out0.score_report[j].GPA = GPA_local;
    }
    if ( class_local == "Intermediate" )
    {
        for ( i = 0; i < vecLen; i++)
        {
            out1.score_report[j].score[i] = score_local[i];
            out1.score_report[j].grade[i] = grade_local[i];
        }
        out1.score_report[j].GPA = GPA_local;
    }
}

```

```

if ( class_local == "Advanced" )
{
    for ( i = 0; i < vecLen; i++)
    {
        out2.score_report[j].score[i] = score_local[i];
        out2.score_report[j].grade[i] = grade_local[i];
    }
    out2.score_report[j].GPA = GPA_local;
}
// initialize these variables for next subrec
GPA_local = 0;
tGPA_local = 0;
}
// outputs
if ( class_local == "Beginner" )
{
    out0.class = class_local;
    writerecord 0;
}
if ( class_local == "Intermediate" )
{
    out1.class = class_local;
    writerecord 1;
}
if ( class_local == "Advanced" )
{
    out2.class = class_local;
    writerecord 2;
}
}

```

osh command

osh -f score_subrec

osh script

The contents of score_subrec are:

```

# compile the expression code
transform -inputschemafile score_subrec_input.schema
-outputschemafile score_subrec_output.schema
-outputschemafile score_subrec_output.schema
-outputschemafile score_subrec_output.schema
-expressionfile score_subrec_expr -flag compile -name score_map;
# run the job
import -schemafile score_subrec_input.schema -file score_subrec.txt | transform -flag run -name
score_map > a.v >b.v >c.v;
0> -export -schemafile score_subrec_output.schema -file beginner.out -overwrite < a.v;
1> -export -schemafile score_subrec_output.schema -file
intermediate.out -overwrite < b.v;
2> -export -schemafile score_subrec_output.schema -file advanced.out -overwrite < c.v;

```

Input schema

The contents of score_subrec_input.schema are:

```

record(
    student_id:string[10];
    score_report[2]:subrec(score[3]:nullable int32;term:string;
)

```

Output schema

The contents of score_subrec_output.schema are:

```

record(
    student_id:string[10];
    score_report[2]:subrec(score[3]:nullable int32;
    grade[3]:string[1];
    GPA:sfloat{out_format='%.4.2g'};
    term:string;
    class:string[max=15];
)

```

Example input and output

The input from score_subrec.txt is:

```

B112387567 90 87 62 Final 80 89 52 Midterm
I218925316 95 91 88 Midterm 92 81 78 Final
A619846201 70 82 85 Final 60 89 85 Midterm
I731347820 75 89 93 Final 85 79 92 Midterm
B897382711 85 90 96 Midterm 88 92 96 Final
I327637289 88 92 76 Final 82 96 86 Midterm
A238950561 92 97 89 Final 90 87 91 Midterm
I238967521 87 86 82 Midterm 97 96 92 Final
B826381931 66 73 82 Midterm 86 93 82 Final
A763567100 53 68 92 Final 48 78 92 Midterm

```

The outputs are:

beginner.out

```

B112387567 90 87 62 B B C 2.7 Final 80 89 52 B B D 2.3 Midterm Beginner
B897382711 85 90 96 B B A 3.3 Midterm 88 92 96 B A A 3.7 Final Beginner
B826381931 66 73 82 C C B 2.3 Midterm 86 93 82 B A B 3.3 Final Beginner

```

intermediate.out

```

I218925316 95 91 88 A A B 3.7 Midterm 92 81 78 A B B 3.3 Final Intermediate
I731347820 75 89 93 B B A 3.3 Final 85 79 92 B B A 3.3 Midterm Intermediate
I327637289 88 92 76 B A B 3.3 Final 82 96 86 B A B 3.3 Midterm Intermediate
I238967521 87 86 82 B B B 3 Midterm 97 96 92 A A A 4 Final Intermediate

```

advanced.out

```

A619846201 70 82 85 C B B 2.7 Final 60 89 85 C B B 2.7 Midterm Advanced
A238950561 92 97 89 A A B 3.7 Final 90 87 91 B B A 3.3 Midterm Advanced
A763567100 53 68 92 D C A 2.3 Final 48 78 92 D B A 2.7 Midterm Advanced

```

Example 8: external C function calls

This example demonstrates how external C functions can be included in the Transformation Language. See "External Global C-Function Support" for the details of C-function support.

This example contains these components:

- C header file
- C source file
- Transformation Language expression file
- osh script to run the transform operator

C header file: functions.h

```

int my_times( int x , int y );
unsigned int my_sum( unsigned int x , unsigned int y );
void my_print_message( char* msg );
#if defined (_alpha)

```

```

long my_square( long x , long y );
#else
long long my_square( long long x , long long y );
#endif

C source file: functions.c

#include <stdio.h>
#include "functions.h"
int my_times( int x , int y )
{
    int time;
    time = x * y;
    return time;
}
unsigned int my_sum( unsigned int x, unsigned int y )
{
    unsigned int sum;
    sum = x + y;
    return sum;
}
void my_print_message(char* msg)
{
    printf("%s\n",msg);
    return;
}
#if defined(__alpha)
long my_square( long x, long y )
{
    long square;
    square = x*x + y*y;
    return square ;
}
#else
long long my_square( long long x , long long y )
{
    long long square;
    square = x*x + y*y;
    return square ;
}
#endif

```

Transformation language

The expression file name for this example is t_extern_func.

```

extern int32 my_times(int32 x, int32 y);
extern uint32 my_sum(uint32 x, uint32 y);
extern void my_print_message(string msg);
extern int64 my_square(int64 x, int64 y);
inputname 0 in0;
outputname 0 out0;
mainloop
{
    out0.times= my_times(in0.a1,in0.a2);
    out0.sum= my_sum(in0.a1,in0.a2);
    out0.square= my_square(in0.a1,in0.a2);
    my_print_message("HELLO WORLD!");
    writerecord 0;
}

```

osh script

```

transform
-inputschema record(a1:int32;a2:int32)
-outputschema record(times:int32;sum:uint32;square:int64;)
-expressionfile t_extern_func
-flag compile

```

```

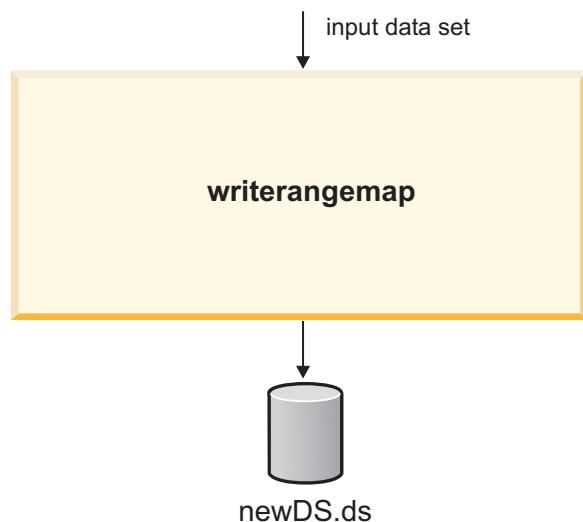
-name my_extern_func
-staticobj /DIR/functions.o;
generator -schema record(a1:int32;a2:int32) | transform -flag run -name my_extern_func |
peek;

```

Writerangemap operator

The writerangemap operator takes an input data set produced by sampling and partition sorting a data set and writes it to a file in a form usable by the range partitioner. The range partitioner uses the sampled and sorted data set to determine partition boundaries.

Data flow diagram



The operator takes a single data set as input. You specify the input interface schema of the operator using the `-interface` option. Only the fields of the input data set specified by `-interface` are copied to the output file.

writerangemap: properties

Table 64. writerangemap properties

Property	Value
Number of input data sets	1
Number of output data sets	0 (produces a data file as output)
Input interface schema:	specified by the interface arguments
Output interface schema	none
Transfer behavior	inRec to outRec without modification
Execution mode	sequential only
Partitioning method	range
Preserve-partitioning flag in output set	set
Composite operator	no

Writerangemap: syntax and options

The syntax for the writerangemap operator is shown below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose the value in single quotes.

```
writerangemap
  [-key fieldname
   [-key fieldname ...]]
  | [-interface schema]
  -collation_sequence locale
  | collation_file_pathname | OFF
    [-overwrite]
  -rangemap filename
```

Table 65. Writerangemap options

Option	Use
-key	<p>-key <i>fieldname</i></p> <p>Specifies an input field copied to the output file. Only information about the specified field is written to the output file. You only need to specify those fields that you use to range partition a data set.</p> <p>You can specify multiple -key options to define multiple fields.</p> <p>This option is mutually exclusive with -interface. You must specify either -key or -interface, but not both.</p>
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none">Specify a predefined IBM ICU localeWrite your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i>Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence.By default, WebSphere DataStage sorts strings using byte-wise comparisons. <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.htm</p>
-interface	<p>-interface <i>schema</i></p> <p>Specifies the input fields copied to the output file. Only information about the specified fields is written to the output file. You only need to specify those fields that you use to range partition a data set.</p> <p>This option is mutually exclusive with -key; that is, you can specify -key or -interface, but not both.</p>

Table 65. Writerangemap options (continued)

Option	Use
-overwrite	-overwrite Tells the operator to overwrite the output file, if it exists. By default, the operator does not overwrite the output file. Instead it generates an error and aborts the job if the file already exists.
-rangemap	-rangemap <i>filename</i> Specifies the pathname of the output file which will contain the sampled and sorted data.

Using the writerange operator

For an example showing the use of the writerange operator, see "Example: Configuring and Using range Partitioner".

Chapter 8. The import/export library

The import and export operators are used by stages that read and write files or data sets.

Generally, operators process only data formatted as and contained in data sets. The import operator imports data into WebSphere DataStage and converts it to an data set. The export operator converts data sets to external representations.

The import/export utility consists of these operators:

- The import operator: imports one or more data files into a single data set (see "Import Operator").
- The export operator: exports a data set to one or more data files (see "Export Operator").

Data must be formatted as a WebSphere DataStage data set in order for WebSphere DataStage to partition and process it. However, most WebSphere DataStage jobs process data that is represented in a format other than WebSphere DataStage data sets including flat files, text or binary files that consist of data without metadata (a description of the data). The import and export operators are used to convert flat data files to data sets, and to convert data sets to flat data files.

A flat file must be a single file, not a parallel set of files or fileset.

Flat files processed by WebSphere DataStage contain record-based data stored in a binary or text format. Although the layout of the data contained in these files is record-based, the files do not have a built-in schema describing the record layout.

To import and export flat files, you must describe their layout by means of WebSphere DataStage record schemas. Record schemas define the name and data type of each field in a record. They can be partial or complete. (See "Record Schemas") You can also optionally specify data properties. Properties supply information about the format of the imported or exported data. Properties can apply to both records and fields. (See "Import/Export Properties").

In order for WebSphere DataStage to process such data, it must first be imported into a WebSphere DataStage data set. After data processing operations have been completed, data can be stored as a WebSphere DataStage data set or exported to its original format or a different format.

The following data formats cannot be imported by the import operator:

- filesets, or parallel sets of files
- RDBMS tables
- SAS data sets
- COBOL data files

For RDBMS tables and parallel SAS data sets, import and export are performed implicitly. In other words, you can work with these data sets as if they were WebSphere DataStage data sets. For COBOL data files, you use the COBOL conversion utility to import and export the data. For other flat files, you must use the import and export operators.

Note: Do not use the import or export operators for RDBMS tables, SAS data sets, or COBOL data files. Only use import or export for flat files that are not COBOL data files.

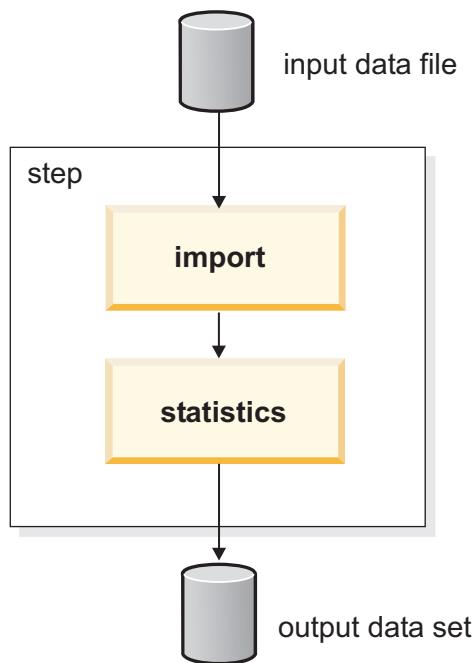
Record schemas

A record schema is an implicit or explicit description of the layout and properties of the record-oriented data contained in a WebSphere DataStage data set. Implicit (default) record schemas are discussed in "The Default Import Schema" and "The Default Export Schema".

When you invoke either the import or the export operator, you must explicitly specify a schema, as in the following two examples.

Import example 1: import schema

In this example, the import operator imports data and the statistics operator calculates statistics on the Income field of the imported data set before writing it to disk. The diagram shows the data flow for this example.

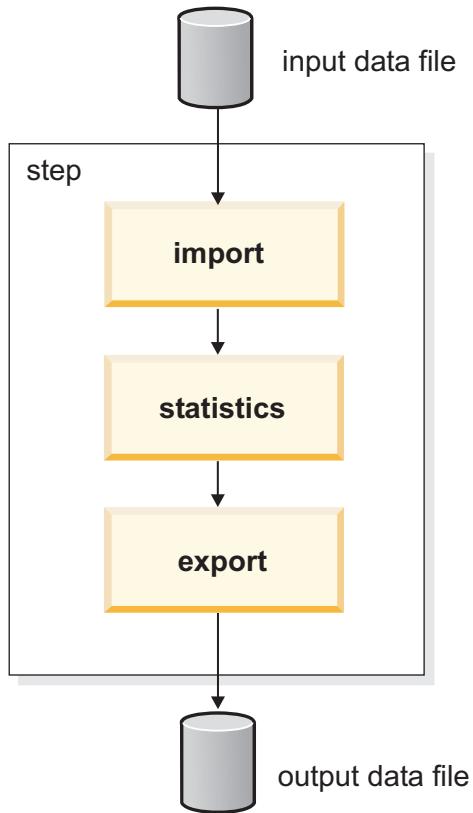


Here is the schema for importing the data into a WebSphere DataStage data set as defined in osh.

```
record (
    Name:string;
    Age:int8;
    Gender:uint8;
    Income:sfloat;
)
```

Example 2: export schema

In this example, the same imported data set is again stored as a flat file after the statistics operator has processed it. As is sometimes the case, the same schema as in "Example 1: Import Schema" is used, and the data is returned to its original format.



Field and record properties

To define a record schema, you specify the following for each field:

- A field name
- A data type
- An optional length specification for string and raw fields
- An optional length specification for vector fields
- An optional nullability specification (by default, imported fields are not nullable)
- An optional set of one or more import/export properties, which can apply to the record as a whole or to a specific field. Properties are enclosed in braces ({}). Record properties appear after the keyword record and field properties appear before the final semicolon (;) in the definition of an individual field.

Below is a completely described data set record, where each byte of the record is contained in a field and all fields are accessible by WebSphere DataStage because they have the required field definition information.

a:sfloat	b:int8	c:int8	d:dfloat	e:sfloat	f:decimal	g:decimal	h:int8
----------	--------	--------	----------	----------	-----------	-----------	--------

The following osh code defines this record:

```
record (
    a:sfloat;
    b:int8;
    c:int8;
    d:dfloat;
```

```

e:sfloat;
f:decimal[1,0];
g:decimal[1,0];
h:int8;

)

```

If you wish to modify this example to specify that each record is to be delimited by a comma (,) you would add the record-level property `delim = ','`:

```

record {delim = ','} (
    a:sfloat;
    b:int8;
    c:int8;
    d:dfloat;
    e:sfloat;
    f:decimal[1,0];
    g:decimal[1,0];
    h:int8;
)

```

The same property `delim = ','` could be used as a field-level property, for example:

```

record (
    a:sfloat;
    b:int8 {delim = ','};
    c:int8;
    d:dfloat;
    e:sfloat;
    f:decimal[1,0];
    g:decimal[1,0];
    h:int8;
)

```

In this case, only field `b` would be delimited by a comma. For export, the other fields would be followed by the default delimiter, the ascii space (0x20).

These brief examples only scratch the surface of schema definition. See "Import/Export Properties" for more information on record and field properties.

Complete and partial schemas

The schemas discussed so far are all examples of *complete* schemas, that is, a schema where you define each field. WebSphere DataStage allows you to define another type of schema: a partial schema. To define a partial schema, define only the fields that will be acted on. For example, to process records with tens or hundreds of fields, you might have to access, and therefore define, only a few of them for sorting keys.

Here is a partial record schema where field information is defined for only two fields:

	c:int8		f:decimal	
--	--------	--	-----------	--

The use of partial record schemas has advantages and disadvantages, which are summarized in the following table.

Schema Type	Advantages	Disadvantages
partial	Partial record schemas are simpler than complete record schemas, because you define only the fields of interest in the record. The import operator reads these records faster than records with a complete schema because it does not have to interpret as many field definitions.	WebSphere DataStage treats the entire imported record as an atomic object; that is, WebSphere DataStage cannot add, remove, or modify the data storage within the record. (WebSphere DataStage can, however, add fields to the beginning or end of the record.)
complete	WebSphere DataStage can add fields to the body of the record and remove or modify any existing field. Removing fields from a record allows you to minimize its storage requirement.	The import operator takes longer to read records with a complete schema than records with a partial schema.

Defining partial record schemas

To define a partial record schema, you specify the intact property of the record schema. For information about this property, see "intact". A record with no fields is the simplest kind to define and import, as in the following example:

```
record {intact, record_length=82, record_delim='\r\n'} ()
```

The record schema defines an 82-byte record as specified by the record_length=82 property. The 82 bytes includes 80 bytes of data plus two bytes for the newline and carriage-return characters delimiting the record as specified by record_delim='\r\n'. No information is provided about any field. On import, the two bytes for the carriage-return and line-feed characters are stripped from each record of the data set and each record contains only the 80 bytes of data.

An imported record with no defined fields can be used only with keyless partitioners, collectors, and operators. The term keyless means that no individual fields of the record must be accessed in order for it to be processed. WebSphere DataStage supplies keyless operators to:

- Copy a data set
- Sample a data set to create one or more output data sets containing a random sample of records from the input data set
- Combine multiple input data sets into a single output data set
- Compress or decompress a data set.

In addition, you can use keyless partitioners, such as random or same, and keyless collectors, such as round-robin, with this type of record.

An imported record with no defined fields cannot be used for field-based processing such as:

- Sorting the data based on one or more key fields, such as a name or zip code
- Calculating statistics on one or more fields, such as income or credit card balance
- Creating analytic models to evaluate your data

A record with no field definitions might be passed to an operator that runs UNIX code. Such code typically has a built-in record and field processing ability.

You can define fields in partial schemas so that field-based processing can be performed on them. To do so, define only the fields to be processed. For example, the following partial record schema defines two fields of a record, Name and Income, thereby making them available for use as inputs to partitioners, collectors, and other operators.

```

record { intact, record_length=82, record_delim_string='\r\n' }
  (Name: string[20] { position=12, delim=none });
  Income: int32 { position=40, delim=',', text };
)

```

In this schema, the defined fields occur at fixed offsets from the beginning of the record. The Name field is a 20-byte string field starting at byte offset 12 and the Income field is a 32-bit integer, represented by an ASCII text string, that starts at byte position 40 and ends at the first comma encountered by the import operator.

When variable-length fields occur in the record before any fields of interest, the variable-length fields must be described so that the import or export operator can determine the location of a field of interest.

Note: When variable-length fields occur in the record before any fields of interest, the variable-length fields must be described so that the import and export operators can determine the location of a field of interest.

For example, suppose that the record schema shown above is modified so that the Name and Income fields follow variable-length fields (V1, V2, and V3). Instead of fixed offsets, the Name and Income fields have relative offsets, based on the length of preceding variable-length fields. In this case, you specify the position of the Name and Income fields by defining the delimiters of the variable-length fields, as in the following example:

```

record { intact, record_delim_string='\r\n' }
  ( v1: string { delim=',' };
    v2: string { delim=',' };
    Name: string[20] { delim=none };
    v3: string { delim=',' };
    Income: int32 { delim=',', text };
)

```

Exporting with partial record schemas

The intact portion of a record is an atomic object; that is, WebSphere DataStage cannot add, remove, or modify the data storage contained in the intact portion. When a data set containing an intact is exported, the entire intact portion is exported.

There are two types of data sets containing an intact definition.

- “unmodified intact record”
- “Intact record with additional fields” on page 321

unmodified intact record:

If a schema being exported consists of a single intact record and nothing else, the intact record is exported using its original import record formatting.

For example, consider a data set created using the following import schema:

```

record { intact, record_length=82, record_delim_string='\r\n' }
  (name: string[20] { position=12, delim=none };
   income: uint32 { position=40, delim=',', text });
)

```

In order to export this data set using the original record schema and properties, you specify an empty record schema, as follows:

```
record ()
```

Because there are no properties, no braces ({}) are needed.

This empty record schema causes the export operator to use the original record schema and properties.

However, defining any record property in the export schema overrides all record-level properties specified at import. That is, all record-level properties are ignored by the export operator. For example, if you specify the following record schema to the export operator:

```
record { record_delim='\n' } ()
```

the intact record is exported with a newline character as the delimiter. The carriage-return character ('\r') is dropped. Each exported record is 81 bytes long: 80 bytes of data and one byte for the newline character.

Intact record with additional fields:

When a schema being exported consists of multiple intact records or one intact record together with other fields, consider each intact record to be a separate field of the exported record.

For example, consider a data set with the following record schema:

```
record      (
    a: int32;
    b: string;
    x: record {intact, record_delim='\r\n', text, delim=',', }
        (name: string[20] { position=0 });
        address: string[30] { position=44 };
    );
    y: record {intact, record_length=244, binary, delim=none }
        (v1: int32 { position=84 });
        v2: decimal(10,0) { position=13 };
    );
)
```

The data set contains two intact fields, x and y, and two other fields, a and b. The intact fields retain all properties specified at the time of the import.

When this type of data set is exported, the record-level properties for each intact field (specified in braces after the key word record) are ignored and you must define new record-level properties for the entire record.

For example, if you want to export records formatted as follows:

- a two-byte prefix is written at the beginning of each record
- the fields are ordered x, y, and a; field b is omitted
- a comma is added after x
- y has no delimiter
- a is formatted as binary.

then the export schema is:

```
record { record_prefix=2 }
  ( x: record { intact } () { delim=',' };
    y: record { intact } () { delim=none };
    a: int32 { binary };
  )
```

Note that you do not include the fields of the x and y intact records in the export schema because the entire intact record is always exported. However, this export schema does contain field-level properties for the intact records to control delimiter insertion.

Implicit import and export

An implicit import or export is any import or export operation that WebSphere DataStage performs without your invoking the corresponding operator. WebSphere DataStage performs implicit import and export operations when it recognizes that an input or output file is not stored as an WebSphere DataStage data set (its extension is not .ds).

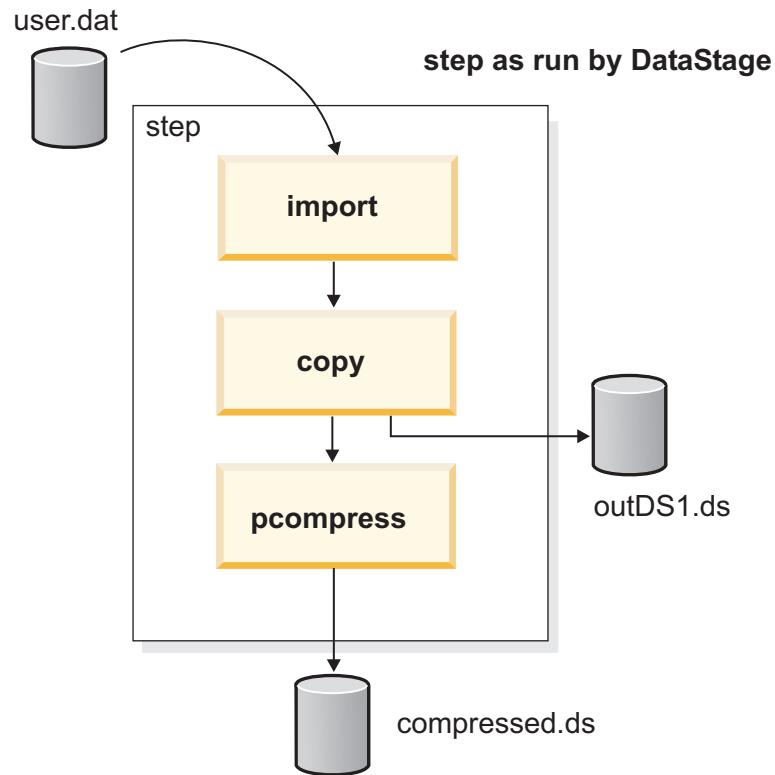
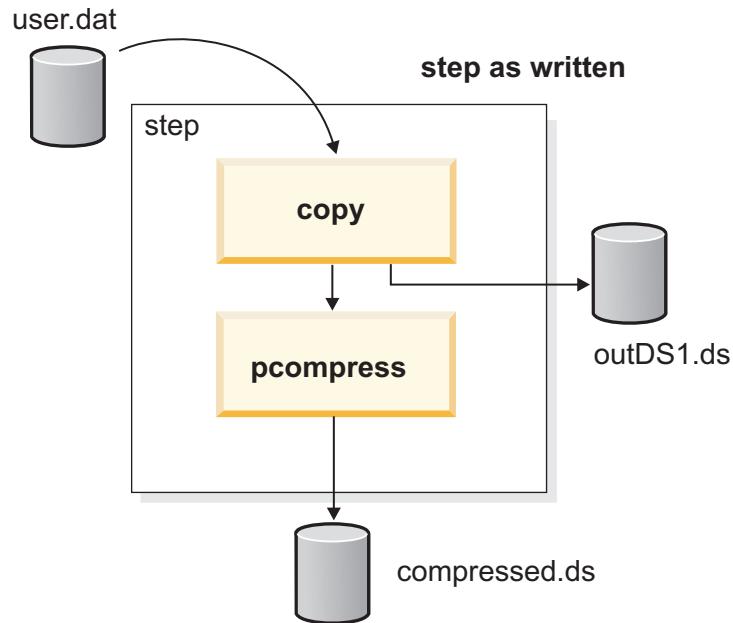
Implicit import/export operations with no schemas specified

Consider a program that takes a flat file as input and makes two copies of the file. The first copy is written to a WebSphere DataStage data set. The second copy is compressed. No import operation is specified. Here is the osh command for the example:

```
$ osh "copy < user.dat > outDS1.ds | pcompress > compressed.ds"
```

No schema is specified, so WebSphere DataStage uses the default schema (See "The Default Import Schema" and "The Default Export Schema").

The diagram shows the data flow model of this step. WebSphere DataStage automatically performs the import operation required to read the flat file.



Implicit operator insertion also works for export operations. The following example shows the tsort operator writing its results to a data file, again with no schema specified:

```
$ osh "tsort -key a -hash -key b < inDS.ds > result.dat"
```

The output file does not end in the extension .ds, so WebSphere DataStage inserts an export operator to write the output data set to the file.

The examples of implicit import/export insertion shown so far all use the default record schemas. However, you can also insert your own record schema to override the defaults.

The next sections discuss:

- "The Default Import Schema"
- "The Default Export Schema"
- "Overriding the Defaults"

The default import schema

The default import schema specifies that imported records contain a single variable-length string field where the end of each record is delimited by a newline character. Here is the default import schema used by WebSphere DataStage:

```
record {record_delim = '\n'}  
(rec:string;)
```

The following figure shows the record layout of the source data as defined by the default import schema:

default record layout of source data

record contents	"\n"
-----------------	------

After the import operation, the destination data set contains a single variable-length string field named rec corresponding to the entire imported record, as follows:

variable-length string named rec

The default export schema

The default export schema is:

```
record()
```

This record schema causes WebSphere DataStage to export all fields as follows:

- Each field except the last field in the record is delimited by an ASCII space (0x20)
- The end of the record (and thus the end of the last field) is marked by the newline character (\n)
- All fields are represented as text (numeric data is converted to a text representation).

For example, the following figure shows the layout of a data set record before and after export when the default export schema is applied:

record in the source data set:

int16	string[8]	string[2]	int32
-------	-----------	-----------	-------

record after export as stored in data file:

int16 as text	0x20	string[8]	0x20	string[2]	0x20	int32 as text	"\n"
---------------	------	-----------	------	-----------	------	---------------	------

Overriding the defaults

To override the defaults of an implicit import or export operation, specify a schema to replace the defaults.

Specify the schema as one of these:

- Part of an osh command, as in the following example:

```
$ osh "[ record_schema ] ..."
• A text file and a reference to the file in the command line, as in the following example:
$ osh "copy < [record @'schema_file'] user.dat > outDS1.ds | pcompress > compressed.ds"
```

where schema_file is the name of the text file containing the record schema definition. You can specify a schema anywhere on an osh command line by using this notation:

[record @'schema_file'].

A schema variable defined upstream of a command, and subsequently input to the command, as in the following example:

```
$ import_schema="record (
    Name:string;
    Age:int8 {default = 127};
    Income:dfloat {skip = 2};
    Phone:string;
)"
$ osh "copy < [$import_schema] user.dat > outDS1.ds |
    pcompress > compressed.ds"
```

All implicit import/export operations in the command use that schema. However, you can override it for a particular data file by preceding the specification of a flat file with a different record schema.

To define formatting properties of records as a whole rather than as individual fields in an export operation, use this form:

record {record_properties} ()

In this case, the export operator exports all the fields of the data set formatted according to the record properties specified in the braces.

Error handling during import/export

The import and export operators can return one of two types of error for each record: a failure or a warning.

Failure

A failure means that both:

- The import or export operator failed to read or write the record's value correctly.
- The error makes it unlikely that the rest of the record can be interpreted.

An example of such a failure occurs when the import operator encounters an invalid length value of a variable-length field. On import, by default, an uninterpretable field is not written to the destination data set and processing continues to the next record. On export, by default, an uninterpretable field is not written to the destination file and processing continues to the next record.

However, you can configure the import and export operators to save records causing a failure in a reject data set. You can also configure the operators to terminate the job in the event of a failure. See "Import Operator" and "Export Operator".

WebSphere DataStage issues a message in the case of a failure.

- The message appears for up to five records in a row, for each partition of the imported or exported data set, when the same field causes the failure.
- After the fifth failure and message, messages no longer appear.
- After a record is successfully processed, the message counter is reset to zero and up to five more error messages per partition can be generated for the same field.

Note: No more than 25 failure messages can be output for the same failure condition during an import or export operation.

Warning

A warning means that the import operator or export operator failed to read or write the record's value correctly but that the import or export of that record can continue. An example of a warning condition is a numeric field represented as ASCII text that contains all blanks. When such a condition occurs, the import or export operator does not issue a message and by default drops the record. To override this behavior, define a default value for the field that causes the warning.

If you have defined a default value for the record field that causes the warning:

- The import operator sets the field to its default value, writes the record to the destination data set, and continues with the next record.
- The export operator sets the field to its default value, exports the record, and continues with the next record.

ASCII and EBCDIC conversion tables

This section contains the ASCII to EBCDIC and EBCDIC to ASCII conversion tables used by the WebSphere DataStage import/export operators.

If an input file uses a character set that is not the native character set of the host computer, the import operator must perform a conversion. For example, if ASCII is the native format for strings on your host computer, but the input data file represents strings using EBCDIC, you must convert EBCDIC to ASCII.

This section contains the lookup tables supplied by WebSphere DataStage for converting between ASCII and EBCDIC. These conversion tables are as defined in *DFSMS/MVS V1R2.0 Using Magnetic Tapes*, Document Number SC26-4923-01, by IBM.

EBCDIC to ASCII

The following table is an EBCDIC-to-ASCII conversion table that translates 8-bit EBCDIC characters to 7-bit ASCII characters. All EBCDIC characters that cannot be represented in 7 bits are represented by the ASCII character 0x1A. This translation is not bidirectional. Some EBCDIC characters cannot be translated to ASCII and some conversion irregularities exist in the table. See "Conversion Table Irregularities" for more information.

EBCDIC	ASCII	EBCDIC Meaning	EBCDIC	ASCII	EBCDIC Meaning
00	00	NUL	80	1A	
01	01	SOH	81	61	a
02	02	STX	82	62	b
03	03	ETX	83	63	c
04	1A	SEL	84	64	d
05	09	HT	85	65	e
06	1A	RNL	86	66	f
07	7F	DEL	87	67	g
08	1A	GE	88	68	h
09	1A	SPS	89	69	i
0A	1A	RPT	8A	1A	
0B	0B	VT	8B	1A	

EBCDIC	ASCII	EBCDIC Meaning	EBCDIC	ASCII	EBCDIC Meaning
0C	0C	FF	8C	1A	
0D	0D	CR	8D	1A	
0E	0E	SO	8E	1A	
0F	0F	SI	8F	1A	
10	10	DLE	90	1A	
11	11	DC1	91	6A	j
12	12	DC2	92	6B	k
13	13	DC3	93	6C	l
14	1A	RES/ENP	94	6D	m
15	1A	NL	95	6E	n
16	08	BS	96	6F	o
17	1A	POC	97	70	p
18	18	CAN	98	71	q
19	19	EM	99	72	r
1A	1A	UBS	9A	1A	
1B	1A	CU1	9B	1A	
1C	1C	IFS	9C	1A	
1D	1D	IGS	9D	1A	
1E	1E	IRS	9E	1A	
1F	1F	ITB/IUS	9F	1A	
20	1A	DS	A0	1A	
21	1A	SOS	A1	7E	ø
22	1A	FS	A2	73	s
23	1A	WUS	A3	74	t
24	1A	BYP/INP	A4	75	u
25	0A	LF	A5	76	v
26	17	ETB	A6	77	w
27	1B	ESC	A7	78	x
28	1A	SA	A8	79	y
29	1A	SFE	A9	7A	z
2A	1A	SM/SW	AA	1A	
2B	1A	CSP	AB	1A	
2C	1A	MFA	AC	1A	
2D	05	ENQ	AD	1A	
2E	06	ACK	AE	1A	
2F	07	BEL	AF	1A	
30	1A		B0	1A	
31	1A		B1	1A	
32	16	SYN	B2	1A	
33	1A	IR	B3	1A	

EBCDIC	ASCII	EBCDIC Meaning	EBCDIC	ASCII	EBCDIC Meaning
34	1A	PP	B4	1A	
35	1A	TRN	B5	1A	
36	1A	NBS	B6	1A	
37	04	EOT	B7	1A	
38	1A	SBS	B8	1A	
39	1A	IT	B9	1A	
3A	1A	RFF	BA	1A	
3B	1A	CU3	BB	1A	
3C	14	DC4	BC	1A	
3D	15	NAK	BD	1A	
3E	1A		BE	1A	
3F	1A	SUB	BF	1A	
40	20	(space)	C0	7B	
41	1A	RSP	C1	41	A
42	1A		C2	42	B
43	1A		C3	43	C
44	1A		C4	44	D
45	1A		C5	45	E
46	1A		C6	46	F
47	1A		C7	47	G
48	1A		C8	48	H
49	1A		C9	49	I
4A	5B	¢	CA	1A	
4B	2E	.	CB	1A	
4C	3C	<	CC	1A	
4D	28	(CD	1A	
4E	2B	+	CE	1A	
4F	21		CF	1A	
50	26	&	D0	7D	
51	1A		D1	4A	J
52	1A		D2	4B	K
53	1A		D3	4C	L
54	1A		D4	4D	M
55	1A		D5	4E	N
56	1A		D6	4F	O
57	1A		D7	50	P
58	1A		D8	51	Q
59	1A		D9	52	R
5A	5D	!	DA	1A	
5B	24	\$	DB	1A	

EBCDIC	ASCII	EBCDIC Meaning	EBCDIC	ASCII	EBCDIC Meaning
5C	2A	*	DC	1A	
5D	29)	DD	1A	
5E	3B	;	DE	1A	
5F	5E	..	DF	1A	
60	2D	-	E0	5C	\
61	1A	/	E1	1A	
62	1A		E2	53	S
63	1A		E3	54	T
64	1A		E4	55	U
65	1A		E5	56	V
66	1A		E6	57	W
67	1A		E7	58	X
68	1A		E8	59	Y
69	1A		E9	5A	Z
6A	7C		EA	1A	
6B	2C	'	EB	1A	
6C	25		EC	1A	
6D	5F	-	ED	1A	
6E	3E	>	EE	1A	
6F	3F	?	EF	1A	
70	1A		F0	30	0
71	1A		F1	31	1
72	1A		F2	32	2
73	1A		F3	33	3
74	1A		F4	34	4
75	1A		F5	35	5
76	1A		F6	36	6
77	1A		F7	37	7
78	1A		F8	38	8
79	60		F9	39	9
7A	3A	:	FA	1A	
7B	23	#	FB	1A	
7C	40	@	FC	1A	
7D	27	'	FD	1A	
7E	3D	=	FE	1A	
7F	22	"	FF	1A	

ASCII to EBCDIC

The following table is an ASCII-to-EBCDIC conversion table that translates 7-bit ASCII characters to 8-bit EBCDIC characters. This translation is not bidirectional. Some EBCDIC characters cannot be translated to

ASCII and some conversion irregularities exist in the table. For more information, see "Conversion Table Irregularities".

Table 66. ASCII to EBCDIC Conversion

ASCII	EBCDIC	ASCII Meaning	ASCII	EBCDIC	ASCII Meaning
00	00		40		
01	01		41		
02	02		42		
03	03		43		
04	1A		44		
05	09		45		
06	1A		46		
07			47		
08			48		
09			49		
0A			4A		
0B			4B		
0C			4C		
0D			4D		
0E			4E		
0F			4F		
10			50		
11			51		
12			52		
13			53		
14			54		
15			55		
16			56		
17			57		
18			58		
19			59		
1A			5A		
1B			5B		
1C			5C		
1D			5D		
1E			5E		
1F			5F		
20			60		
21			61		
22			62		
23			63		
24			64		
25			65		

Table 66. ASCII to EBCDIC Conversion (continued)

ASCII	EBCDIC	ASCII Meaning	ASCII	EBCDIC	ASCII Meaning
26			66		
27			67		
28			68		
29			69		
2A			6A		
2B			6B		
2C			6C		
2D			6D		
2E			6E		
2F			6F		
30			70		
31			71		
32			72		
33			73		
34			74		
35			75		
36			76		
37			77		
38			78		
39			79		
3A			7A		
3B			7B		
3C			7C		
3D			7D		
3E			7E		
3F			7F		

Conversion table irregularities

The EBCDIC-to-ASCII and ASCII-to-EBCDIC conversion tables previously shown are standard conversion tables. However, owing to the nature of EBCDIC-to-ASCII and ASCII-to-EBCDIC conversions, certain irregularities exist in the conversion tables. For example, an exclamation point is defined in EBCDIC as 0x5A. In ASCII 7-bit and 8-bit codes, an exclamation point is defined as 0x21.

The table shows the conversion irregularities.

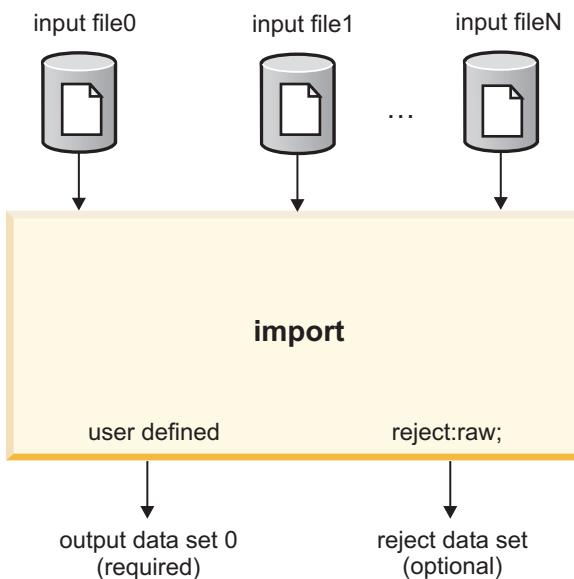
EBCDIC Code		8-Bit ASCII Code		7-Bit ASCII Code	
Graphic	Hex	Graphic	Hex	Graphic	Hex
¢	4A	[5B	[5B
!	5A]	5D]	5D
[AD	(n/a)	D5	SUB	1A

EBCDIC Code		8-Bit ASCII Code		7-Bit ASCII Code	
Graphic	Hex	Graphic	Hex	Graphic	Hex
]	BD	(n/a)	E5	SUB	1A
	4F	!	21	!	21
_	6A	-	7C	-	7C

Import operator

The import operator reads one or more non-DataStage source files and converts the source data to a destination WebSphere DataStage data set.

Data flow diagram



import: properties

Table 67. *import* properties

Property	Value
Number of input data sets	0
Number of output data sets	1, or 2 if you specify optional data sets
Input interface schema	none
Output interface schema	output data set 0: user defined reject data set: reject:raw;
Transfer behavior	none
Execution mode	parallel or sequential
Preserve-partitioning flag in output data set(s)	clear by default
Composite operator	no

The import operator:

- Takes one or more files or named pipes as input. You can import data from one or more sources, provided all the imported data has the same record layout.

- Writes its results to a single output data set.
- Allows you to specify an optional output data set to hold records that are not successfully imported.
- Has an output interface schema corresponding to the import schema of the data files read by the operator.

The import operator can import source files containing several types of data:

- Data in variable-length blocked/spanned records: Imported records have a variable-length format. This format is equivalent to IBM format-V, format-VS, format-VB, and format-VBS files.
- Data in fixed-length records: This format is equivalent to IBM format-F and format-FB files.
- Data in prefixed records: Records fields are prefixed by a length indicator. Records can be of either fixed or variable length.
- Delimited records: Each record in the input file is separated by one or more delimiter characters (such as the ASCII newline character '\n'). Records can be of either fixed or variable length.
- Implicit: Data with no explicit record boundaries.

WebSphere DataStage accepts a variety of record layouts. For example, a data file might represent integers as ASCII strings, contain variable-length records, or represent multi-byte data types (for example, 32-bit integer) in big-endian or little-endian format. See "Import/Export Properties" for more information on data representation.

When the import operator imports a fixed-length string or ustring that is less-than or greater-than the declared schema length specification, an error is generated.

Import: syntax and options

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
import
{ -file filename | -fileset filename | -filepattern pattern }
([-schema schema] | [-schemafile filename] )
[-checkpoint n]
[-dontUseOffsetsWithSources]
[-filter command]
[-first n] [-firstLineColumnNames]
[-keepPartitions]
[-missingFile error | okay]
[-multinode yes | no]
[-readers n]
[recordNumberField recordNumberFieldName]
[-rejects action]
[-reportProgress yes | no]
[-source programe [args]]
[-sourcelist filename]
[-sourceNameField sourceStringFieldName]
```

The following option values can contain multi-byte Unicode values:

- the file names given to the -file, -schemafile, and -sourcelist options
- the base file name for the -fileset option
- -schema option schema
- the program name and arguments for the -source option
- the -filepattern pattern
- the -filter command

There are two types of required options.

- You must include one of the following options to specify the imported files: -file, -filepattern, -fileset, -source, -sourcelist.
- You must include either -schema or -schemafile to define the layout of the imported data unless you specify -fileset and it contains a schema. You can select only one of these three options.

You can optionally include other arguments.

Table 68. Import options

Option	Use
-checkpoint	<p>-checkpoint <i>n</i></p> <p>Import <i>n</i> records per segment to the output data set. (A data segment contains all records written to a data set by a single WebSphere DataStage step.)</p> <p>By default, the value of <i>n</i> is 0, that is, the entire input is imported for each iteration of the step. If the step containing the import operator performs multiple iterations, the output data set of the import operator contains multiple copies of the input data.</p> <p>The source files of the import operation must all be data files or source programs, that is, no pipes or other input devices can be used. In addition, you cannot specify a filter on the input.</p> <p>An import operator that creates a segmented output data set must be contained in a checkpointed step.</p>
-dontUseOffsetsWith Sources	<p>-dontUseOffsetsWithSources</p> <p>Start the source program's output stream at 0. Do not use this option with checkpointing.</p>
-file	<p>-file <i>nodeName :file0 ... -file nodeName :filen</i></p> <p>Specifies an input file or pipe.</p> <p>You can include multiple -file options to specify multiple input files. The file names might contain multi-byte Unicode characters.</p> <p>Note: You can specify a hyphen to signal that import takes its input from the stdin of osh. <i>nodeName</i> optionally specifies the name of the node on which the file is located. If you omit <i>nodeName</i>, the importer assumes that the file is located on the node on which the job is invoked.</p> <p>You cannot use -file with -filepattern, -fileset, -source, or -sourcelist.</p>
-filepattern	<p>-filepattern <i>pattern</i></p> <p>Specifies a group of files to import.</p> <p>For example, you can use the statement:</p> <p>-filepattern data*.txt</p> <p>You cannot use -filepattern with the -file, -fileset, -source, -sourcelist or -readers options.</p>

Table 68. Import options (continued)

Option	Use
-fileset	<p>-fileset <i>file_set .fs</i></p> <p>Specifies <i>file_set .fs</i>, a file containing a list of files, one per line, used as input by the operator. The suffix <i>.fs</i> identifies the file to WebSphere DataStage as a file set. See "File Sets". <i>.file_set</i> might contain multi-byte Unicode characters.</p> <p>To delete a file set and the data files to which it points, invoke the WebSphere DataStage data set administration utility, orchadmin.</p> <p>You cannot use -fileset with -file, -filepattern, -source, or -sourcelist.</p>
-filter	<p>-filter command</p> <p>Specifies a UNIX command to process input files as the data is read from a file and before it is converted into the destination data set. The command might contain multi-byte Unicode characters.</p> <p>For example, you could specify the following filter:</p> <p>-filter 'grep 1997'</p> <p>to filter out all records that do not contain the string "1997". Note that the source data to the import operator for this example must be newline-delimited for grep to function properly.</p> <p>WebSphere DataStage checks the return value of the -filter process, and errors out when an exit status is not OK.</p> <p>You cannot use -filter with -source.</p>
-first	<p>[-first <i>n</i>]</p> <p>This option imports the first <i>n</i> records of a file.</p> <p>This option does not work with multiple nodes, filesets, or file patterns. It does work with multiple files and with the -source and -sourcefile options when file patterns are specified.</p> <p>Here are some osh command examples:</p> <p>osh "import -file file1 -file file2 -first 10 > outfile"</p> <p>osh "import -source 'cat file1' -first 10 > outfile"</p> <p>osh "import -sourcelist sourcefile -first 10 > outfile"</p> <p>osh "import -file file1 -first 5 > outfile"</p>
-firstLineColumnNames	<p>[-firstLineColumnNames]</p> <p>Specifies that the first line of a source file should not be imported.</p>

Table 68. Import options (continued)

Option	Use
-keepPartitions	<p>-keepPartitions</p> <p>Partitions the imported data set according to the organization of the input file(s).</p> <p>By default, record ordering is not preserved, because the number of partitions of the imported data set is determined by the configuration file and any constraints applied to the data set.</p> <p>However, if you specify -keepPartitions, record ordering is preserved, because the number of partitions of the imported data set equals the number of input files and the preserve-partitioning flag is set in the destination data set.</p>
-missingFile	<p>-missingFile error okay</p> <p>Determines how the importer handles a missing input file.</p> <p>By default, the importer fails and the step terminates if an input file is missing (corresponding to -missingFile error). Specify -missingFile okay to override the default.</p> <p>However, if the input file name has a node name prefix of "*", missing input files are ignored and the import operation continues. (This corresponds to -missingFile okay.)</p>
-multinode	<p>[-multinode yes no]</p> <p>-multinode yes specifies the input file is to be read in sections from multiple nodes; -multinode no, the default, specifies the file is to be read by a single node.</p>

Table 68. Import options (continued)

Option	Use
-readers	<p>-readers <i>numReaders</i></p> <p>Specifies the number of instances of the import operator on each processing node. The default is one operator per node per input data file.</p> <p>If <i>numReaders</i> is greater than one, each instance of the import operator reads a contiguous range of records from the input file. The starting record location in the file for each operator (or <i>seek</i> location) is determined by the data file size, the record length, and the number of instances of the operator, as specified by <i>numReaders</i>, which must be greater than zero.</p> <p>All instances of the import operator for a data file run on the processing node specified by the file name.</p> <p>The output data set contains one partition per instance of the import operator, as determined by <i>numReaders</i>.</p> <p>The imported data file(s) must contain fixed-length records (as defined by the <i>record_length=fixed</i> property). The data source must be a file or files. No other devices are allowed.</p> <p>This option is mutually exclusive with the <i>filepattern</i> option.</p>
-recordNumberField	<p>-recordNumberField <i>recordNumberFieldName</i></p> <p>Adds a field with field name <i>recordNumberFieldName</i> with the record number as its value.</p>
-rejects	<p>-rejects { continue fail save }</p> <p>Configures operator behavior if a record is rejected. The default behavior is to continue. Rejected records are counted but discarded. The number of rejected records is printed as a log message at the end of the step.</p> <p>However, you can configure the operator to either fail and terminate the job or save, that is, create output data set 1 to hold reject records.</p> <p>If -rejects fail is specified and a record is not successfully imported, the import operator issues an error and the step terminates. If -rejects fail is not specified and a record is not successfully imported, the import operator issues a warning and the step does not terminate.</p>
-reportProgress	<p>-reportProgress { yes no }</p> <p>By default (yes) the operator displays a progress report at each 10% interval when the importer can ascertain file size. Reporting occurs only if: import reads a file as opposed to a named pipe, the file is greater than 100 KB, records are fixed length, and there is no filter on the file.</p> <p>Disable this reporting by specifying -reportProgress no.</p>

Table 68. Import options (continued)

Option	Use
-schema	<p>-schema <i>record_schema</i></p> <p>Specifies the import record schema. The value to this option might contain multi-byte Unicode characters.</p> <p>You can also specify a file containing the record schema using the syntax:</p> <p>-schema record @'<i>file_name</i>'</p> <p>where <i>file_name</i> is the path name of the file containing the record schema.</p> <p>You cannot use -schema with -schemafile.</p>
-schemafile	<p>-schemafile <i>file_name</i></p> <p>Specifies the name of a file containing the import record schema. It might contain multi-byte Unicode characters.</p> <p>You cannot use -schemafile with -schema.</p>
-sourceNameField	<p>-sourceNameField <i>sourceStringFieldName</i></p> <p>Adds a field named <i>sourceStringFieldName</i> with the import source string as its value</p>

Table 68. Import options (continued)

Option	Use
<p>-source</p>	<p>-source <i>prog_name args</i></p> <p>Specifies the name of a program providing the source data to the import operator. WebSphere DataStage calls <i>prog_name</i> and passes to it any arguments specified. <i>prog_name</i> and <i>args</i> might contain multi-byte Unicode characters.</p> <p>You can specify multiple -source arguments to the operator. WebSphere DataStage creates a pipe to read data from each specified <i>prog_name</i>.</p> <p>You can prefix <i>prog_name</i> with either a node name to explicitly specify the processing node that executes <i>prog_name</i> or a node name prefix of "*", which causes WebSphere DataStage to run <i>prog_name</i> on all processing nodes executing the operator.</p> <p>If this import operator runs as part of a checkpointed step, as defined by -checkpoint, WebSphere DataStage calls <i>prog_name</i> once for each iteration of the step.</p> <p>WebSphere DataStage always appends three arguments to <i>prog_name</i> as shown below:</p> <pre>prog_name args -s <i>H L</i></pre> <p>where <i>H</i> and <i>L</i> are 32-bit integers. <i>H</i> and <i>L</i> are set to 0 for the first step iteration or if the step is not checkpointed.</p> <p>For each subsequent iteration of a checkpointed step, <i>H</i> and <i>L</i> specify the (64-bit) byte offset (<i>H</i> = upper 32 bits, <i>L</i> = lower 32 bits) at which the source program's output stream should be restarted.</p> <p>At the end of each step iteration, <i>prog_name</i> receives a signal indicating a broken pipe. The <i>prog_name</i> can recognize this signal and perform any cleanup before exiting.</p> <p>WebSphere DataStage checks the return value of the -source process, and errors out when an exit status is not OK.</p> <p>You cannot use -source with -filter, -file, -filepattern, or -filesset.</p>

Table 68. Import options (continued)

Option	Use
-sourcelist	<p>-sourcelist <i>file_name</i></p> <p>Specifies a file containing multiple program names to provide source data for the import. This file contains program command lines, where each command line is on a separate line of the file. <i>file_name</i> might contain multi-byte Unicode characters.</p> <p>WebSphere DataStage calls the programs just as if you specified multiple -source options. See the description of -source for more information.</p> <p>You cannot use -sourcelist with -filter, -file, -filepattern, or -filesset.</p>

How to import data

- Follow the procedure described in "Specify the Layout of the Imported Data" below.
- Follow the procedure described in "Specify the Source of the Imported Data" below.

Specify the layout of the imported data

Define a schema describing the layout of the data file to be imported. Refer to the these sections for a discussion of record schemas as they pertain to import/export operations: "Record Schemas" and "Complete and Partial Schemas".

A schema definition can also define record and field properties. Refer to "Import/Export Properties" to learn about setting up record and field properties.

The WebSphere DataStage data set that is output by the import operator is formatted according to the schema you have defined. However, this output schema does not define properties that were set up in the original schema definition.

Note: If you do not provide a schema, WebSphere DataStage assumes that imported data is formatted according to the default schema discussed in "The Default Import Schema".

Specify the source of the imported data

The source of the data to be imported can be one of these:

- A file or named pipe (see "Files and named pipes" on page 341)
- A file set (see "File sets" on page 341)
- A file pattern (see "File patterns" on page 341)
- A source program's output (see "Source program's output" on page 342)
- The output of several source programs defined in a source list (see "List of source programs" on page 342)

You can also explicitly specify the nodes and directories from which the operator imports data (see "Nodes and directories" on page 342).

Files and named pipes

Specify the -file option and the path name of the file or named pipe, which might optionally be preceded by the node name. You might specify more than one file or named pipe. For each one, repeat the operation.

Note: For external sequential data files (as opposed to WebSphere DataStage file sets) that are to be processed by a UNIX custom operator that functions in parallel, first import the data sequentially and then export it in parallel using the file **.file_name* option. The wild card ensures that the operator writes to the file on every node on which it executes. (See "Export Operator".)

File sets

A file set is a text file containing a list of source files to import. The file set must contain one file name per line. The name of the file has the form *file_name.fs*, where .fs identifies the file to WebSphere DataStage as a file set.

Specify the -fileset option and the path name of the file set.

Shown below is a sample file set:

```
--Orchetsrate File Set v1
--LFile
node0:/home/user1/files/file0
node0:/home/user1/files/file1
node0:/home/user1/files/file2
--LFile
node1:/home/user1/files/file0
node1:/home/user1/files/file1
--Schema
record {record_delim="\n"}
( a:int32;
  b:int32;
  c:int16;
  d:sfloat;
  e:string[10];
)
```

The first line of the file set must be specified exactly as shown above.

The list of all files on each processing node must be preceded by the line --LFile and each file name must be prefixed by its node name.

A file set can optionally contain the record schema of the source files as the last section, beginning with --Schema. If you omit this part of the file set, you must specify a schema to the operator by means of either the -schema or -schemafile option.

File patterns

A file pattern specifies a group of similarly named files from which to import data. The wild card character allows for variations in naming. For example, the file pattern inFile*.data imports files that begin with the inFile and end with .data. The file names can contain multi-byte Unicode characters.

For example, to import the data from the files that match the file pattern state*.txt, use the statement:
-filepattern state*.txt

Specify the -filepattern option and the pattern.

Source program's output

You can specify the name of a program providing the source data to the import operator.

Specify the -source option and the program name and program arguments, if any.

List of source programs

You can specify a file containing the names of multiple programs that provide source data to the import operator.

Specify the -sourcelist option and the file name.

Nodes and directories

Indicate the source to be imported by specifying:

[*nodeName* :]*path_name*

The operator assumes that *path_name* is relative to the working directory from which the job was invoked.

You can also indicate one of these:

- A specific processing node on which the operator runs. Do this by specifying the node's name. When you do, WebSphere DataStage creates an instance of the operator on that node to read the *path_name*. The *nodeName* must correspond to a node or fastname parameter of the WebSphere DataStage configuration file.
- All processing nodes on which the operator runs. Do this by specifying the asterisk wild card character (*). When you do, WebSphere DataStage reads *path_name* on every node on which the operator is running.

For example, you can supply the following specification as the file pattern on the osh command line:

:inFile.data

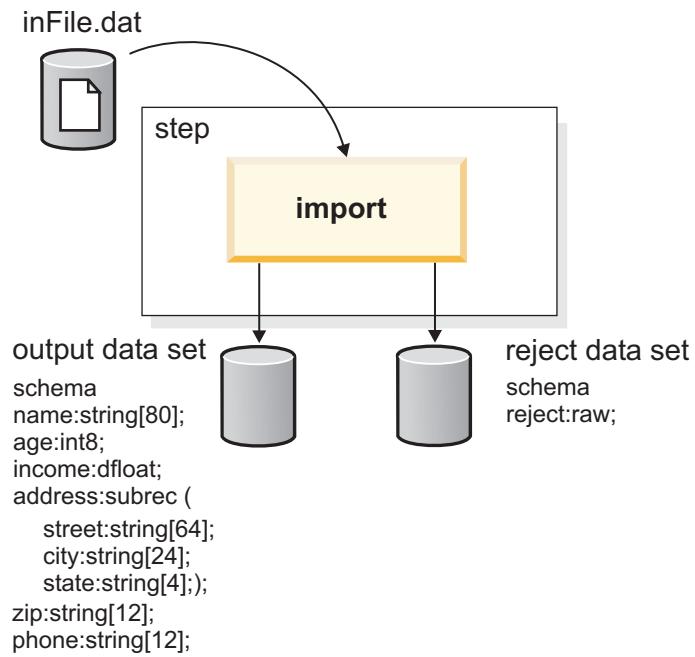
This imports all files of the form inFile*.data residing on all processing nodes of the default node pool. You can include a relative or absolute path as part of inFile. If you do not supply an absolute path, import searches for the files on all nodes using the same current working directory as the one from which you invoked the job.

Example 1: importing from a single data file

In this example, a single file is imported from disk. The file contains fixed-length records and no field or record delimiters. The step contains a single instance of the import operator, which:

- Reads a file from disk
- Converts the file to a single data set
- Saves the output in a persistent data set
- Saves records that cause a failure in a reject data set

The diagram shows the data flow for this example.



Here is the osh schema declaration for this example:

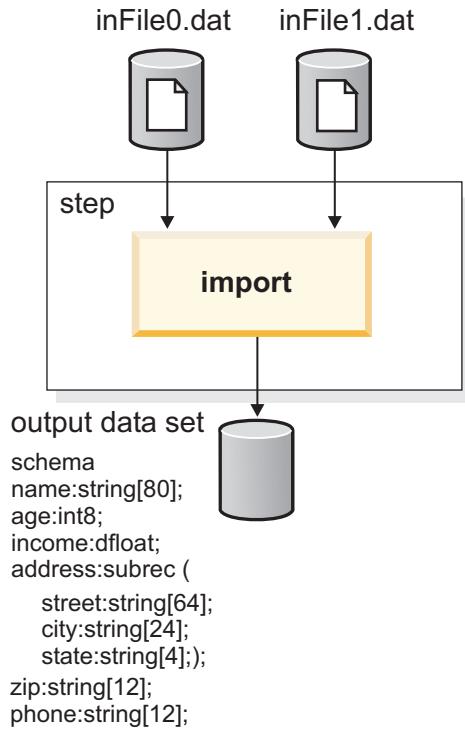
```

$ example1_schema="record {record_length = fixed, delim = none, binary} (
    Name:string[80];
    Age:int8 {default = 127};
    Income:dfloat {text, width = 8};
    Address:subrec (
        Street:string[64];
        City:string[24];
        State:string[4]; );
    Zip:string[12];
    Phone:string[12]; )"

```

Example 2: importing from multiple data files

In this example two files are imported from disk. Each one contains fixed-length records. The diagram shows the data-flow diagram for this example.



The format of the source file and the import schema are the same as those in "Example 1: Importing from a Single Data File" and the step similarly contains a single instance of the import operator. However, this example differs from the first one in that it imports data from two flat files instead of one and does not save records that cause an error into a reject data set.

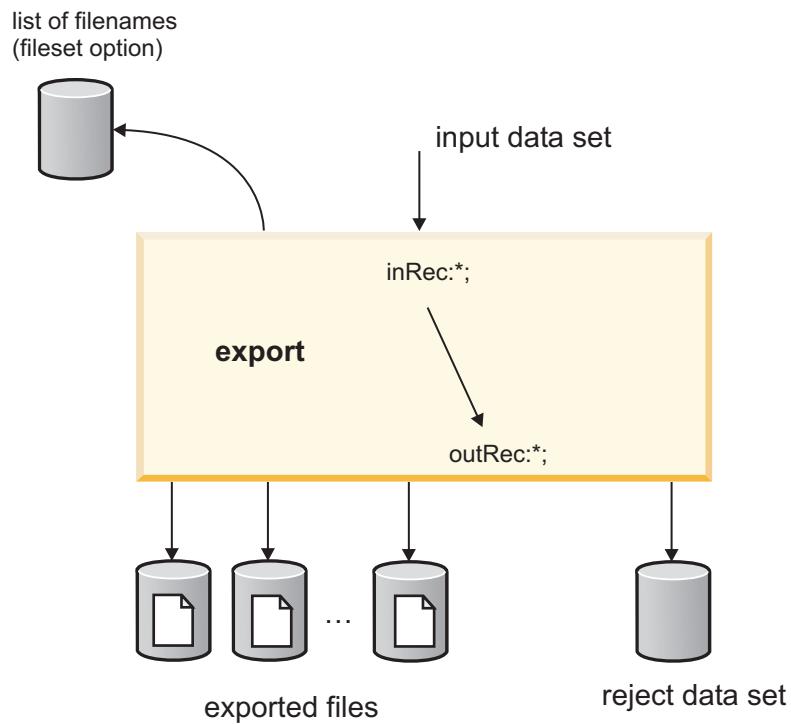
Specify the data flow with the following osh code:

```
$ osh "import -file inFile0.data -file inFile1.data
-schema $example1_schema > outDS.ds "
```

Export operator

The export operator exports a WebSphere DataStage data set to one or more UNIX data files or named pipes located on one or more disk drives.

Data flow diagram



export: properties

Table 69. Export properties

Property	Value
Number of input data sets	1
Number of output data sets	0 or 1
Input interface schema	inRec: *; plus user-supplied export schema
Output interface schema	output files: none reject data set: outRec: *;
Transfer behavior	inRec -> outRec if you specify a reject data set
Execution mode	parallel
Partitioning method	any
Collection method	any
Preserve-partitioning flag in output data set(s)	clear
Composite operator	yes

The export operator:

- Takes a single WebSphere DataStage data set as input.
- Writes its results to one or more output files.
- Allows you to specify an optional output data set to hold records that cause a failure.
- Takes as its input interface schema the schema of the exported data.
- Generates an error and terminates if you try to write data to a file that is not empty. (This behavior can be overridden.)
- Deletes files already written if the step of which it is a part fails. (This behavior can be overridden.)

The source data set might be persistent (stored on disk) or virtual (not stored on disk).

The export operator writes several formats of exported data, including:

- Fixed-length records: a fixed-length format equivalent to IBM format-F and format-FB files.
- Data in prefixed records: Record fields are prefixed by a length indicator. Records can be of either fixed or variable length.
- Delimited records: A delimiter such as the ASCII line feed character is inserted after every record. Records can be of either fixed or variable length.
- Implicit: No explicit boundaries delimit the record.

Note: The export operator does not support variable-length blocked records (IBM format-VB or format-VBS).

WebSphere DataStage accepts a variety of record layouts for the exported file. For example, a data file might represent integers as ASCII strings, contain variable-length records, or represent multi-byte data types in big-endian or little-endian format. See "Import/Export Properties" for more information.

Note: When your character setting is UTF-16, the export operator appends and prepends the byte-order mark OxfeOxff0x00 to every column value.

Export: syntax and options

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
export
{ -file filename | -fileset listName.fs
-destination progname [ args ] | destinationlist filename }
{ -schema schema | -schemafile schemafile }
[-add_bom { utf16be | utf16le | utf8 }]
[-append]
[-create | -replace | -discard_records | -discard_schema_and_records]
[-diskpool diskpool]
[-dontUseOffsetsWithDestinations]
[-filter command ]
[-firstLineColumnNames]
[-maxFileSize numMB] [-nocleanup]
[-overwrite]
[-prefix prefix ]
[-rejects continue | fail | save]
[-single[FilePerPartition]]
[-suffix suffix]
[-writeSchema | -omitSchema]
```

Note: The following option values can contain multi-byte Unicode characters:

- the -file and -destinationlist file names and the base name of the file for the -fileset option
- the program name and arguments to the -destination option
- the -schema option schema and the schema filename given to the -schemafile option
- the prefix value given to the -prefix option, and the suffix value given to the -suffix option
- the -filter option command
- the value given to the -diskpool option

There are two types of required options:

- You must include exactly one of the following to specify the exported files: -file, -fileset, -destination, or -destinationlist.
- You must include exactly one of -schema or -schemafile to define the layout of the exported data.

Table 70. Export operator options

Option	Use
-add_bom	<p><code>-add_bom { utf16be utf16le utf8 }</code></p> <p>With this option you can add a BOM to your exported file.</p> <p>The utf16be value specifies FE FF, utf16le specifies FF FE, and utf8 specifies EF BB BF.</p>
-append	<p><code>-append</code></p> <p>Append exported data to an existing file. By default the step terminates if you attempt to export data to a file that is not empty. This option overrides the default behavior.</p> <p>You cannot use this option with <code>-overwrite</code>.</p>
-destination	<p><code>-destination <i>prog_name</i> [<i>args</i>]</code></p> <p>In single quotation marks specify the name of a program that reads the data generated by the export operator. Specify the program's arguments, if any. WebSphere DataStage calls <i>prog_name</i> and passes to it any specified arguments.</p> <p>You can specify multiple <code>-destination</code> options to the operator: for each <code>-destination</code>, specify the option and supply the <i>prog_name</i> and <i>args</i> (if any). The <i>prog_name</i> and <i>args</i> values might contain multi-byte Unicode values.</p> <p>If this export operator runs as part of a checkpointed step, WebSphere DataStage calls <i>prog_name</i> once for each iteration of the step.</p> <p>WebSphere DataStage always appends three additional arguments to <i>prog_name</i>:</p> $\textit{prog_name} [\textit{args}] -s H L$ <p>where <i>H</i> and <i>L</i> are 32-bit integers. For the first step iteration, or if the step is not checkpointed, <i>H</i> and <i>L</i> are set to 0.</p> <p>For each subsequent iteration of a checkpointed step, <i>H</i> and <i>L</i> specify the (64-bit) byte offset (<i>H</i> = upper 32 bits, <i>L</i> = lower 32 bits) of the exported data in the total export stream from the operator.</p> <p>After all data has been written to the program, <i>prog_name</i> is called once more with an appended switch of <code>-e</code> (corresponding to end of file) and is not passed the <code>-s</code> switch. On last call <i>prog_name</i> can perform any final operation, for example, write a trailing label to a tape.</p> <p>If the export operation fails, WebSphere DataStage calls <i>prog_name</i> once with the appended switch <code>-c</code> (cleanup) and no <code>-s</code> switch. This gives the program an opportunity to clean up.</p> <p>You cannot use this option with <code>-filter</code>.</p>
-destinationlist	<p><code>-destinationlist <i>file_name</i></code></p> <p>Specifies in single quotation marks <i>file_name</i>, the name of a file containing the names of multiple destination programs, where each command line is listed on a separate line of the file. <i>file_name</i> might contain multi-byte Unicode characters.</p> <p>WebSphere DataStage calls the programs as if you specified multiple <code>-destination</code> options. See the description of <code>-destination</code> for more information.</p>
-dontUseOffsetsWith Destinations	<p><code>-dontUseOffsetsWithDestinations</code></p> <p>Do not supply the <code>-s</code>, <i>H</i>, <i>L</i> arguments to destination programs. This means that the byte offset is always 0. See the <code>-destination</code> option for more information.</p>

Table 70. Export operator options (continued)

Option	Use
-file	<p>-file [nodeName:]outFile0</p> <p>Supply the name of an output file or pipe. The file or pipe must be empty unless you specify either the -append or -overwrite option. You can include multiple -file options to specify multiple input files. For each one, specify -file and supply the file name. The file name can contain multi-byte Unicode characters.</p> <p>Note: You can specify a hyphen to signal that export writes its output to the stdout for osh.</p> <p>You cannot use this option with -fileset.</p>

Table 70. Export operator options (continued)

Option	Use
-fileset	<pre>-fileset <i>filesetName.fs</i> {-create -replace -discard_records -discard_schema_and_records} [-diskpool <i>diskpool</i>] [-maxFileSize <i>numMB</i>] [-prefix <i>prefix</i>] [-single[FilePerPartition]] [-suffix <i>suffix</i>] [-writeSchema -omitSchema]</pre> <p>Specifies the name of the file set, <i>filesetName</i>, a file into which the operator writes the names of all data files that it creates. The suffix .fs identifies the file to WebSphere DataStage as a file set. <i>filesetName</i> can contain multi-byte Unicode characters.</p> <p>The name of each export file generated by the operator is written to <i>filesetName.fs</i>, one name per line.</p> <p>The suboptions are:</p> <ul style="list-style-type: none"> -create: Create the file set. If it already exists, this option generates an error. -replace: Remove the existing fileset and replace it with a new one. -discard_records: Keep the existing files and schema listed in <i>filesetName</i>.fs but discard the records; create the file set if it does not exist. -discard_schema_and_records: Keep existing files listed in <i>filesetName</i>.fs but discard the schema and records; create the file set if it does not exist. <p>The previous suboptions are mutually exclusive with each other and also with the -append option.</p> <ul style="list-style-type: none"> -diskpool <i>diskpool</i>: Specify the name of the disk pool into which to write the file set. <i>diskpool</i> can contain multi-byte Unicode characters. -maxFileSize <i>numMB</i>: Specify the maximum file size in MB. Supply integers. The value of <i>numMB</i> must be equal to or greater than 1. -omitSchema: Omit the schema from <i>filesetName</i>.fs. The default is for the schema to be written to the file set. -prefix: Specify the prefix of the name of the file set components. It can contain multi-byte Unicode characters. If you do not specify a prefix, the system writes the following: export <i>username</i>, where <i>username</i> is your login. -replace: Remove the existing file set and create a new one. -singleFilePerPartition: Create one file per partition. The default is to create many files per partition. This can be shortened to -single. -suffix <i>suffix</i>: Specify the suffix of the name of the file set components. It can contain multi-byte Unicode characters. The operator omits the suffix by default. -writeSchema: Use only with -fileset. Write the schema to the file set. This is the default.⁹ <p>You cannot use -fileset with -file or -filter. "File Sets" discusses file sets.</p>
-firstLineColumnNames	<pre>[-firstLineColumnNames]</pre> <p>Specifies that column names be written to the first line of the output file.</p>
-nocleanup	<pre>-nocleanup</pre> <p>Configures the operator to skip the normal data file deletion if the step fails. By default, the operator attempts to delete partial data files and perform other cleanup operations on step failure.</p>

Table 70. Export operator options (continued)

Option	Use
-overwrite	<p>-overwrite</p> <p>The default action of the operator is to issue an error if you attempt to export data to a file that is not empty. Select -overwrite to override the default behavior and overwrite the file.</p> <p>You cannot use this option with -append or -replace.</p>
-rejects	<p>-rejects continue fail save</p> <p>Configures operator behavior if a record is rejected. The default behavior is to continue. Rejected records are counted but discarded. The number of rejected records is printed as a log message at the end of the step.</p> <p>However, you can configure the operator to either fail and terminate the job or save, that is, create output data set 0 to hold reject records.</p> <p>If you use -rejects fail, osh generates an error upon encountering a record that cannot be successfully exported; otherwise osh generates a warning upon encountering a record that cannot be successfully exported.</p>
-schema	<p>-schema <i>record_schema</i></p> <p>Specifies in single quotation marks the export record schema. You can also specify a file containing the record schema using the syntax:</p> <p>-schema record @<i>file_name</i></p> <p>where <i>file_name</i> is the path name of the file containing the record schema. The <i>file_name</i> and <i>record_schema</i> can contain multi-byte Unicode characters.</p> <p>You cannot use this option with -schemafile.</p>
-schemafile	<p>-schemafile <i>schema_file</i></p> <p>Specifies in single quotation marks the name of a file containing the export record schema. The file name can contain multi-byte Unicode characters.</p> <p>This is equivalent to:</p> <p>-schema record @ <i>schema_file</i></p> <p>You cannot use this option with -schema.</p>
-filter	<p>-filter <i>command</i></p> <p>Specifies a UNIX command to process all exported data after the data set is exported but before the data is written to a file. <i>command</i> can contain multi-byte Unicode characters.</p> <p>You cannot use this option with -fileset or -destination.</p>

How to export data

Specify record and field layout of exported data

Provide the export operator with a schema that defines the record format of the exported data. The schema can define:

- The properties of the exported record, that is, the destination record
- The fields of the source data set to be exported

- The properties of exported fields
- The order in which they are exported

The export operator writes only those fields specified by the export schema and ignores other fields, as in the following example, where the schema of the source WebSphere DataStage data set differs from that of the exported data:

Table 71. Example: Data Set Schema Versus Export Schema

Source WebSphere DataStage Data Set Schema	Export Schema
record (Name: string; Address: string; State: string[2]; Age: int8; Gender: string[1]; Income: dfloat; Phone: string;)	record (Gender: string[1]; State: string[2]; Age: int8; Income: dfloat;)

In the example shown above, the export schema drops the fields Name, Address, and Phone and moves the field Gender to the beginning of the record.

Note: If you do not provide a schema, WebSphere DataStage assumes that exported data is formatted according to the default schema discussed in "The Default Export Schema".

Here is how you set up export schemas:

- To export all fields of the source record and format them according to the default export schema, specify the following:
 - record ()
 - Refer to "The Default Export Schema".
- To export all fields of the source record but override one or more default record properties, add new properties, or do both, specify:


```
record {record_properties} ()
```

 - Refer to "Import/Export Properties" to learn about setting up record and field properties.
 - To export selected fields of the source record, define them as part of the schema definition, as follows:


```
record ( field_definition0; ... field_definitionN; )
```

 - where *field_definition0*; ... *field_definitionN* are the fields chosen for export. No record-level properties have been defined and the default export schema is applied.
 - You can define properties for records and for fields, as in the following example:


```
record {delim = none, binary} (
    Name:string {delim = ','};
    Age:int8 {default = 127};
    Address:subrec (
        Street:string[64];
        City:string[24];
        State:string[4];
        Zip:string[12];
        Phone:string[12];
    )
)
```

Refer to "Import/Export Properties" to learn about setting up record and field properties.

Refer to the following sections for a discussion of schemas as they pertain to import/export operations: "Record Schemas" and "Complete and Partial Schemas".

Specifying destination files or named pipes for exported data

The destination of the data to be exported can be one of the following:

- One or more files or named pipes (see "Files and named pipes" on page 352)
- A file set (see "File sets" on page 352)

- A program's input (see "Destination program's input" on page 353)
- The input to several programs defined in a destination list (see "List of destination programs" on page 353).

You can also explicitly specify the nodes and directories to which the operator exports data (see "Nodes and directories" on page 353).

Files and named pipes

Specify the -file option and the path name of the file or named pipe, which might optionally be preceded by the node name. You might specify more than one file or named pipe. For each one, repeat the operation.

Note: For external sequential data files (as opposed to WebSphere DataStage file sets) that are to be processed by a UNIX custom operator that functions in parallel, you must first import the data sequentially. See "Import Operator". You then export the data in parallel by means of the file **file_name* option. The wild card (*) ensures that the operator writes to the file on every node on which it executes.

File sets

The export operator can generate and name exported files, write them to their destination, and list the files it has generated in a file whose extension is .fs. The data files and the file that lists them are called a file set. They are established by means of the fileset option. This option is especially useful because some operating systems impose a 2 GB limit on the size of a file and you must distribute the exported files among nodes to prevent overruns.

When you choose -fileset, the export operator runs on nodes that contain a disk in the export disk pool. If there is no export disk pool, the operator runs in the default disk pool, generating a warning when it does. However, if you have specified a disk pool other than export (by means of the -diskpool option), the operator does not fall back and the export operation fails.

The export operator writes its results according to the same mechanism. However, you can override this behavior by means of the -diskpool option.

The amount of data that might be stored in each destination data file is limited (typically to 2 GB) by the characteristics of the file system and the amount of free disk space available. The number of files created by a file set depends on:

- The number of processing nodes in the default node pool
- The number of disks in the export or default disk pool connected to each processing node in the default node pool
- The size of the partitions of the data set

You name the file set and can define some of its characteristics. The name of the file set has the form *file_name.fs*, where .fs identifies the file as a file set to WebSphere DataStage. Specify the -fileset option and the path name of the file set.

Note: When you choose -fileset, the export operator names the files it generates and writes the name of each one to the file set. By contrast, you have to name the files when you choose the -file option.

The names of the exported files created by the operator have the following form:

nodeName : dirName /prefixPXXXXXX_FYYYYsuffix

where:

- *nodeName* is the name of the node on which the file is stored and is written automatically by WebSphere DataStage.

- *dirName* is the directory path and is written automatically by WebSphere DataStage.
- *prefix* is by default *export.userName*, where *userName* is your login name, and is written automatically by WebSphere DataStage. However, you can either define your own prefix or suppress the writing of one by specifying "" as the prefix.
- XXXXXX is a 6-digit hexadecimal string preceded by P specifying the partition number of the file written by export. The first partition is partition 000000. The partition numbers increase for every partition of the data set but might not be consecutive.
- YYYY is a 4-digit hexadecimal string preceded by F specifying the number of the data file in a partition. The first file is 0000. The file numbers increase for every file but might not be consecutive.
- WebSphere DataStage creates one file on every disk connected to each node used to store an exported partition.
- *suffix* is a user-defined file-name suffix. By default, it is omitted.

For example, if you specify a *prefix* of "file" and a *suffix* of "_exported", the third file in the fourth partition would be named:

```
node1:dir_name/fileP000003_F0002_exported
```

Some data sets, such as sorted data sets, have a well-defined partitioning scheme. Because the files created by the export operator are numbered by partition and by data file within a partition, you can sort the exported files by partition number and file number to reconstruct your sorted data.

Destination program's input

You can specify the name of a program receiving the exported data. Such a program is called a destination program.

Specify the -destination option and the program name and program arguments, if any.

WebSphere DataStage creates a pipe to write data to each specified program.

List of destination programs

You can specify a file containing the names of multiple programs that receive data from the export operator. Specify the -destinationlist option and the file name.

The number of entries in -destinationlist determines the number of partitions on which the operator runs, regardless of the configuration file's contents. Incoming data is repartitioned as necessary.

Nodes and directories

Indicate the destination of exported data by specifying:

```
[ nodeName :]path_name
```

If you omit the optional node name, the operator exports files or named pipes only to the processing node on which the job was invoked.

If you specify only the file name the operator assumes that *path_name* is relative to the working directory from which the job was invoked.

You can also indicate one of these:

- A specific processing node to which the output is written. Do this by specifying the node's name. The *nodeName* must correspond to a node or fastname parameter of the WebSphere DataStage configuration file.

- All processing nodes on which the operator runs. Do this using the asterisk wild card character (*). When you do, WebSphere DataStage writes one file to every node on which the operator is running.

For example, you can supply the following as the exported file name on the osh command line:

```
*:outFile.data
```

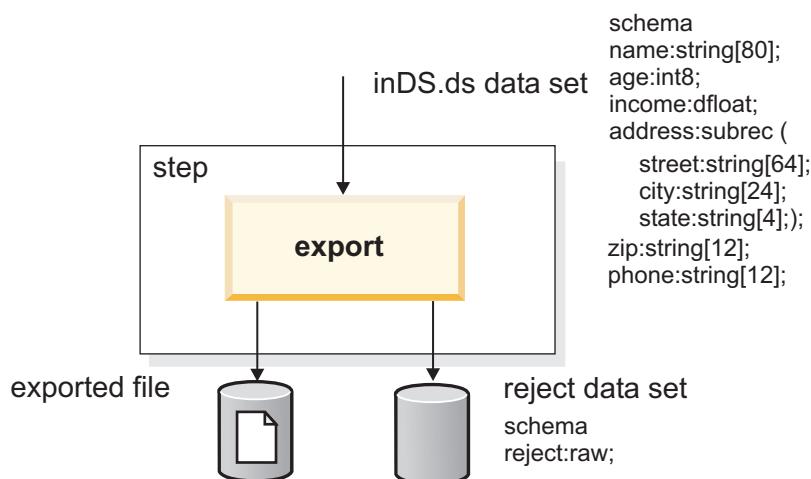
This exports data to files called outFile.data, which are created on all processing nodes on which the operator runs. The name outFile.data is a relative path name and the exporter writes files using the same current working directory as the one from which you invoked the job.

Export example 1: data set export to a single file

In this example, the export operator:

- Runs on a single processing node because it writes to only one file.
- Takes a persistent data set as its source.
- Writes records to a reject data set, if they cannot be successfully exported.
- The destination files created by the export operator are laid out according to the schema supplied in an argument to the operator.

The figure shows the data flow for Example 1.



The following osh code specifies the layout of the destination data file:

```
$ exp_example_1="record {delim = none, binary} (
    Name:string {delim = ','};
    Age:int8 {default = 127};
    Address:subrec (
        Street:string[64];
        City:string[24];
        State:string[4]);
    Zip:string[12];
    Phone:string[12]; )"
```

In this schema, the export operator automatically inserts a newline character at the end of each record according to the default. However, numeric data is exported in binary mode and no field delimiters are present in the exported data file, except for the comma delimiting the *Name* field. Both properties are non-default (see "The Default Export Schema"). They have been established through the definition of record and field properties. The record properties are *delim = none* and *binary*. The field *Name* is explicitly delimited by a comma (*delim = ','*) and this field property overrides the record-level property of *delim = none*. See "Import/Export Properties" for a full discussion of this subject.

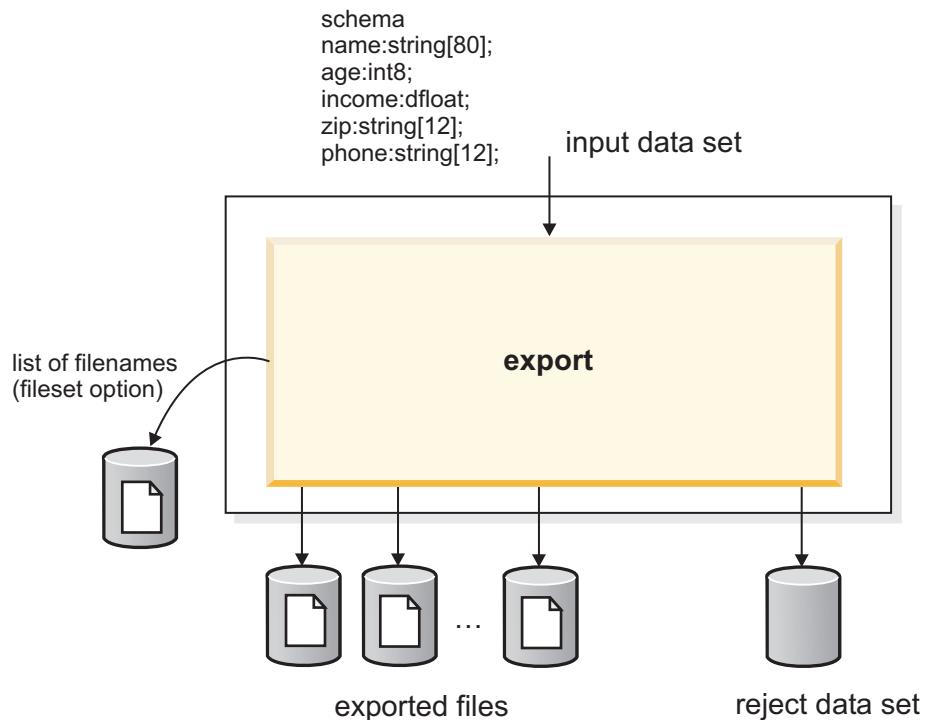
The following osh code uses the specified schema to export the data:

```
$ osh "export -file outFile.dat  
      -schema $exp_example_1  
      -rejects save < inDS.ds > errDS.ds"
```

Example 2: Data Set Export to Multiple files

In this example the operator exports a data set to multiple data files by means of the -fileset option and saves records to a reject data set if they cannot be successfully exported.

The figure shows the data flow of this example:



The following osh code specifies the layout of the destination data files:

```
$ exp_example_2="record {record_length = fixed, delim = none, binary} (  
    Name:string[64];  
    Age:int8;  
    Income:dfloat;  
    Zip:string[12];  
    Phone:string[12]; )"
```

In this example, all fields are of specified length and the schema defines the fixed record length property.

The following osh code uses the specified schema to export the data:

```
$ osh "export -fileset listFile.fs  
      -schema $exp_example_2  
      -rejects save < inDS.ds > errDS.ds"
```

Upon completion of the step, listFile.fs contains a list of every destination data file created by the operator, one file name per line, and contains the record schema used for the export.

Here are the contents of listFile.fs:

```
--Orchestrate File Set v1
--LFile
node0:/home/user1/sfiles/node0/export.user1.P000000_F0000
node0:/home/user1/sfiles/node0/export.user1.P000000_F0001
--LFile
node1:/home/user1/sfiles/node1/export.user1.P000001_F0000
node1:/home/user1/sfiles/node1/export.user1.P000001_F0001
node1:/home/user1/sfiles/node1/export.user1.P000001_F0002
--Schema
record {record_length = fixed, delim = none, binary} (
    Name:string[64];
    Age:int8;
    Income:dfloat;
    Zip:string[12];
    Phone:string[12]; )
```

For more information on file sets, see "File Sets" .

Import/export properties

You can add import/export properties to a schema when you import and export flat files. Properties define the layout of imported and exported data, including such things as how numbers, strings, times, and dates are represented. Properties can apply to records or to fields, and there are numerous default properties that you do not need to specify.

The property determines how the data is represented in the file from which the data is imported or to which the data is exported.

Note: Properties apply only when you import or export data. They are not part of the internal WebSphere DataStage representation of the data, which only needs to know the type of the data.

WebSphere DataStage assumes that data to import or export is formatted according to default properties if:

- You do not specify either record or field properties for imported or exported data.
- WebSphere DataStage performs an unmodified implicit import or export operation (see "Implicit Import/Export Operations with No Schemas Specified").

The defaults are discussed in "The Default Import Schema" and "The Default Export Schema" . You explicitly define properties to override these defaults.

Setting properties

You establish the properties of imported and exported data as part of defining the data's schema. Define record-level properties (which describe the format of the entire record) as follows:

```
record { prop1 [ = arg1 ], prop2 [ = arg2 ], ... propn
        [ = argn ] } (field_definitions ...;)
```

- Define record properties after the word record and before the enumeration of field definitions.
- Enclose the definition in braces ({ }).
- Define properties in a comma-separated list if there is more than one property.
- Attribute values to the properties with the attribution sign (=). If the attributed values are strings, enclose them in single quotes.
- Specify special characters by starting with a backslash escape character. For example, \t represents an ASCII tab delimiter character.

For example, the following defines a record in which all fields are delimited by a comma except the final one, which is delimited by a comma followed by an ASCII space character:

```

record {delim_string = ',', final_delim_string = ', '} (
    a:int32;
    b:string;
    c:int8;
    d:raw; )

```

Define field-level properties (which describe the format of a single field) as follows:

```

field_definition { prop1 [= arg1], prop2 [= arg2], ...
    propn [= argn] };

```

where *field_definition* is the name and data type of the field.

- Define field properties after the *field_definition* and before its final semi-colon.
- Enclose the definition of properties in braces ({}).
- Define properties in a comma-separated list if there is more than one property.
- Attribute values to the properties with the attribution sign (=). If the attributed values are strings, enclose them in single quotes.
- Specify special characters by starting with a backslash escape character. For example, \t represents an ASCII tab delimiter character.

For example, the following specifies that the width of a string field is 40 bytes and that the ASCII space pad character is used when it is exported:

```
record (a: string { width = 40, padchar = ' ' } );
```

You can specify many default values for properties at the record level. If you do, the defined property applies to all fields of that data type in the record, except where locally overridden. For example, if numeric data of imported or exported records are represented in binary format, you can define the binary property at record level, as in the following example:

```
record {binary, delim = none} (a:int32; b:int16; c:int8;)
```

With one exception, properties set for an individual field can override the default properties set at record level. The exception is the fill property, which is defined at record level but cannot be overridden by an individual field or fields.

For example, the following schema sets the record length as fixed with no field delimiters and the layout of all fields as binary. However, the definition of the properties of the *purposeOfLoan* field are as follows: the field is formatted as text, its length is 5 bytes (and not the 4 bytes of the int32 data type), and it is delimited by a comma:

```

record
    {record_length = fixed, delim = none, binary}
(
    checkingAccount Status:int32;
    durationOfAccount:sfloat;
    creditHistory:int32;
    purposeOfLoan:int32 {width = 5, text, delim = ','}
    creditAmount: sfloat;
    savingsAccountAmount:int32;
    yearsEmployed:int32;
    installmentRate:sfloat;
)

```

Properties

Certain properties are associated with entire records, and other properties with fields of specific data types, but most can be used with fields of all data types and the records that contain them. This section contains the following topics:

- "Record-Level Properties"

- "Numeric Field Properties"
- "String Field Properties"
- "Ustring Field Properties"
- "Decimal Field Properties"
- "Date Field Properties"
- "Time Field Properties"
- "Timestamp Field Properties"
- "Raw Field Properties"
- "Vector Properties"
- "Nullable Field Properties"
- "Tagged Subrecord Field Properties"

Each property that is described in the following section is discussed in detail in "Properties: Reference Listing".

Record-level properties

Some properties apply only to entire records. These properties, which establish the record layout of the imported or exported data or define partial schemas, can be specified only at the record level.

If you include no properties, default import and export properties are used; see "The Default Import Schema" and "The Default Export Schema".

The following table lists the record-level properties that cannot be set at the field level.

Keyword	Use
intact check_intact	Defines a partial schema and optionally verifies it
record_length	Defines the fixed length of a record
record_prefix	Defines a field's length prefix as being 1, 2, or 4 bytes long
record_format	With type = implicit, field length is determined by field content With type = varying, defines IBM blocked or spanned format
record_delim record_delim_string	Defines character(s) delimiting a record
fill	Defines the value used to fill gaps between fields of an exported record. This property applies to export only and cannot be set at the field level.

Field Properties

You can define properties of imported and exported fields. This section lists field properties according to the data types that they can be used with, under the following categories.

- "Numeric Field Properties" on page 359
- "String Field Properties" on page 360
- "Decimal Field Properties" on page 360
- "Date Field Properties" on page 361
- "Time Field Properties" on page 361
- "Timestamp Field Properties" on page 362

- “Raw Field Properties” on page 362
- “Vector Properties” on page 363
- “Nullable Field Properties” on page 363
- “Subrecord Field Properties” on page 363
- “Tagged Subrecord Field Properties” on page 364

There is some overlap in these tables. For example, a field might be both decimal and nullable, and so field properties in both the decimal and nullable categories would apply to the field. Furthermore many properties such as `delim` apply to fields of all (or many) data types, and to avoid tedious repetition are not listed in any of the categories. However, they all appear in “Properties: Reference Listing” .

Numeric Field Properties

Numeric data types can be signed or unsigned integers of 8, 16, 32, or 64 bits:

- `int8`, `int16`, `int32`, `int64`
- `uint8`, `uint16`, `uint32`, `uint64`

Numeric data types can also be single- or double-precision floating-point numbers:

- `sfloat`
- `dfloat`

By default, the import and export operators assume that numeric fields are represented as text. The import operator invokes the C functions `strtol()`, `strtoul()`, or `strtod()` to convert the text representation to a numeric format. The export operator invokes the C function `sprintf()` to convert the numeric representation to text.

When exported as text, numeric fields take up a varying number of bytes, based on the size of the actual field values. The maximum number of bytes is as follows:

- 8-bit signed or unsigned integers: 4 bytes
- 16-bit signed or unsigned integers: 6 bytes
- 32-bit signed or unsigned integers: 11 bytes
- 64-bit signed or unsigned integers: 21 bytes
- single-precision float: 14 bytes (sign, digit, decimal point, 7 fraction, "E", sign, 2 exponent)
- double-precision float: 24 bytes (sign, digit, decimal point, 16 fraction, "E", sign, 3 exponent)

Numeric Field Properties

Keyword	Use	See
<code>big_endian</code>	Specifies the byte ordering as big-endian	<code>big_endian</code>
<code>c_format</code>	Non-default format of text-numeric-text translation	<code>c_format</code>
<code>in_format</code>	Format of text translation to numeric field	<code>in_format</code>
<code>little_endian</code>	Specifies the byte ordering as little-endian	<code>little_endian</code>
<code>max_width</code>	Defines the maximum width of the destination field	<code>max_width</code>
<code>native_endian</code>	Specifies the byte ordering as native-endian	<code>native_endian</code>

Keyword	Use	See
out_format	Format of numeric translation to text field	out_format
padchar	Pad character of exported strings or numeric values	padchar
width	Defines the exact width of the destination field	width

String Field Properties

Keyword	Use	See
max_width	Defines the maximum width of the destination field	max_width
padchar	Defines the pad character of exported strings or numeric values	padchar
width	Defines the exact width of the destination field	width

Ustring Field Properties

Keyword	Use	See
charset	At the field level, it defines the character set to be used for ustring fields; at the record level it applies to the other import/export properties that support multi-byte Unicode character data.	charset
max_width	Defines the maximum width of the destination field	max_width
padchar	Defines the pad character of exported strings or numeric values	padchar
width	Defines the exact width of the destination field	width

Decimal Field Properties

Keyword	Use	See
decimal_separator	Specifies an ASCII character to separate the integer and fraction components of a decimal	decimal_separator
fix_zero	Treat a packed decimal field containing all zeros (normally illegal) as a valid representation of zero	fix_zero
max_width	Defines the maximum width of the destination field	max_width
packed	Defines the field as containing a packed decimal	packed
precision	Defines the precision of a decimal	precision
round	Defines the rounding mode	round

Keyword	Use	See
scale	Defines the scale of a decimal	scale
separate	Defines the field as containing an unpacked decimal with a separate sign byte	separate
width	Defines the exact width of the destination field	width
zoned	Defines the field as containing an unpacked decimal in text format	zoned

Date Field Properties

Keyword	Use	See
big_endian	Specifies the byte ordering as big-endian	big_endian
binary	In this context, a synonym for the julian property	binary
charset	Specifies the character set for the date	charset
date_format	Defines a text-string date format or uformat format other than the default; uformat can contain multi-byte Unicode characters	date_format
days_since	The field stores the date as a signed integer containing the number of days since date_in_ISO_format or uformat.	days_since
default_date_format	Provides support for international date components	default_date_format
julian	Defines the date as a binary numeric value containing the Julian day	julian
little_endian	Specifies the byte ordering as little-endian	little_endian
native_endian	Specifies the byte ordering as native-endian	native_endian
text	Specifies text as the data representation	text

Time Field Properties

Keyword	Use	See
big_endian	Specifies the byte ordering as big-endian.	big_endian
binary	In this context, a synonym for the julian property	binary
charset	Specifies the character set for the time field.	charset
default_time_format	Provides support for international time components	default_time_format

Keyword	Use	See
little_endian	Specifies the byte ordering as little_endian	little_endian
native_endian	Specifies the byte ordering as native_endian.	native_endian
midnight_seconds	Represent the time field as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.	midnight_seconds
time_format	Defines a text-string time format or uformat format other than the default	time_format
midnight_seconds	Defines the field as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight; see "midnight_seconds".	midnight_seconds
native_endian	Specifies the byte ordering as native-endian	native_endian
text	Specifies text as the data representation	text
time_format	Defines a text-string date format or uformat other than the default; uformat can contain multi-byte Unicode characters	time_format

Timestamp Field Properties

Keyword	Use	See
big_endian	Specifies the byte ordering as big-endian.	big_endian
binary	In this context, a synonym for the julian property	binary
charset	Specifies the character set for the timestamp field	charset
little_endian	Specifies the byte ordering as little_endian	little_endian
native_endian	Specifies the byte order as native_endian	native_endian
text	Specifies text as the data representation	text
timestamp_format	Defines a text-string date format or uformat other than the default; uformat can contain multi-byte Unicode characters	timestamp_format

Raw Field Properties

There are no type-specific properties for raw fields.

Vector Properties

WebSphere DataStage supports vector fields of any data type except subrecord and tagged subrecord. WebSphere DataStage vectors can be of either fixed or variable length.

No special properties are required for fixed-length vectors. The import and export operators perform the number of import/export iterations defined by the number of elements in the vector.

However, variable-length vectors can have the properties described in the following table.

Keyword	Use	See
link	This field holds the number of elements in a variable-length vector	link
prefix	Defines the length of prefix data, which specifies the length in bytes	prefix
vector_prefix	Defines the length of vector prefix data, which specifies the number of elements	vector_prefix

Nullable Field Properties

All WebSphere DataStage data types support null values. Fields might be declared to be nullable or not.

The following table describes the null field properties.

Keyword	Use	See
actual_length	Defines the number of bytes to skip in an imported record if the field contains a null; and the number of bytes to fill with null-field or specified pad-character, if the exported field contains a null	actual_length
null_field	Specifies the value representing null.	null_field
null_length	Specifies the value of the length prefix of a variable-length field that contains a null	null_length

Subrecord Field Properties

The following properties apply to subrecords. These properties establish the subrecord layout of the imported or exported data. They cannot be specified at the field level.

If you include no properties, default import and export properties are used; see "The Default Import Schema" and "The Default Export Schema".

Keyword	Use	See
fill	Defines the value used to fill gaps between fields of an exported record. This property applies to export only and cannot be set at the field level	fill
record_delim	Specifies a single ASCII or multi-byte Unicode character to delimit a record	record_delim

Keyword	Use	See
record_delim_string	Specifies an string or ustring to delimit a record	record_delim_string
record_format	With type = implicit, field length is determined by field content. With type = varying, defines IBM blocked or spanned format.	record_format
record_length	Defines the fixed length of a record	record_length
record_prefix	Defines field's length prefix as being 1, 2, or 4 bytes long	record_prefix
record_format	With type = implicit, field length is determined by field content. With type = varying, defines IBM blocked or spanned format.	record_format

Tagged Subrecord Field Properties

A tagged subrecord is a field whose type can vary. The subfields of the tagged subrecord are the possible types. The *tag value* of the tagged subrecord selects which of those types is used to interpret the field's value for the record. In memory, the tag is stored with the field. On import or export, it must be broken out as a separate field.

A tagged subrecord field in an imported or exported record must be preceded by an uint32 tag field whose value identifies which of the tagged aggregate's components is active. The tag field can be one of these:

- A prefix of the tagged aggregate
- A separate field in the record (that is, at the same level as the tagged subrecord field)

The following table describes the tagged subrecord properties.

Keyword	Use	See
prefix	Prefix of a tagged subrecord containing the value of the tag	prefix
reference	Holds the name of the field containing the tag value	reference
tagcase	Holds the tag value for a tag case field in a subrecord	tagcase

Properties: reference listing

The following table lists all record and field properties in alphabetical order.

Property	Specifies	See
actual_length	Bytes to skip in imported record if the field contains a null; bytes to fill with null-field or specified pad character, if exported field contains a null.	actual_length
ascii	Character set of data is in ASCII format.	ascii

Property	Specifies	See
big_endian	Multi-byte data types are formatted as big endian.	big_endian
binary	Field value represented in binary format; decimal represented in packed decimal format; julian day format; time represented as number of seconds from midnight; timestamp formatted as two 32-bit integers	binary
c_format	Format of text-numeric translation format to and from numeric fields	c_format
charset	Specifies a character set	charset
"record {text} (" " a:int32; " " b:int16 {binary}; " " c:int8;) "	Error checking of imported records with partial record schema (a suboption of intact)	check_intact
check_intact		
date_format	Defines a text-string date format or uformat format other than the default; uformat can contain multi-byte Unicode characters	date_format
days_since	The imported or exported field stores the date as a signed integer containing the number of days since <i>date_in_ISO_format</i> or <i>uformat</i> .	days_since
decimal_separator	Specifies an ASCII character to separate the integer and fraction components of a decimal	decimal_separator
default	Default value for a field that causes an error	default
default_date_format	Provides support for international date components in date fields	default_date_format
default_time_format	Provides support for international time components in time fields	default_time_format
delim	Trailing delimiter of all fields	delim
delim_string	One or more ASCII characters forming trailing delimiter of all fields	delim_string
drop	Field dropped on import	drop
ebcdic	Character set of data is in EBCDIC format	ebcdic
export_ebcdic_as_ascii	Exported field converted from EBCDIC to ASCII	export_ebcdic_as_ascii
fill	Byte value to fill in gaps in exported record	fill
final_delim	Delimiter character trailing last field of record	final_delim
final_delim_string	Delimiter string trailing last field of record	final_delim_string
fix_zero	A packed decimal field containing all zeros (normally illegal) is treated as a valid representation of zero	fix_zero
generate	Creation of exported field	generate

Property	Specifies	See
import_ascii_as_ebcdic	Translation of imported string field from ASCII to EBCDIC	import_ascii_as_ebcdic
in_format	Format of text translation to numeric field	in_format
intact	The record definition defines a partial record schema	intact
julian	The imported or exported field represents the date as a numeric value containing Julian day	julian
link	A field holds the length of another, variable-length field of the record; field might be a vector	link
little_endian	Multi-byte data types are formatted as little endian	little_endian
max_width	Maximum width of the destination field.	max_width
midnight_seconds	The field represents the time as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight	midnight_seconds
native_endian	Multi-byte data types are formatted as defined by the native format of the machine; this is the default for import/export operations	native_endian
nofix_zero	A packed decimal field containing all zeros generates an error (default)	nofix_zero
null_field	Value of null field in file to be imported; value to write to exported file if source field is null	null_field
null_length	Imported length meaning a null value; exported field contains a null	null_length
overpunch	The field has a leading or ending byte that contains a character which specifies both the numeric value of that byte and whether the number as a whole is negatively or positively signed	overpunch
out_format	Format of numeric translation to text field	out_format
packed	The imported or exported field contains a packed decimal	packed
padchar	Pad character of exported strings or numeric values	padchar
position	The byte offset of the field in the record	position
precision	The precision of the packed decimal	precision
prefix	Prefix of all fields of record; also length of prefix holding the length in bytes of a vector	prefix

Property	Specifies	See
"record {prefix = 2} (" " a:string; " " b:string {prefix = 1};)"	Each imported field generates message	print_field
print_field		
quote	Field is enclosed in quotes or another ASCII character; useful for variable-length fields	quote
record_delim	Record delimited by a single ASCII character	record_delim
record_delim_string	Record delimited by one or more ASCII characters	record_delim_string
record_format	Variable-length blocked records or implicit records	record_format
record_length	Fixed length records	record_length
record_prefix	Records with a length prefix stored as binary data	record_prefix
reference	Name of field containing field length	reference
round	Rounding mode of source decimal	round
scale	The scale of the decimal	scale
separate	The imported or exported field contains an unpacked decimal with a separate sign byte	separate
skip	Number of bytes skipped from the end of the previous field to the beginning of this one	skip
tagcase	Defines the active field in a tagged subrecord	tagcase
text	Field represented as text-based data; decimal represented in string format; date, time, and timestamp representation is text-based	text
time_format	The format of an imported or exported field representing a time as a string	time_format
timestamp_format	The format of an imported or exported field representing a timestamp as a string	timestamp_format
vector_prefix	Prefix length for element count	vector_prefix
width	The exact width of the destination field	width
zoned	The imported or exported field contains an unpacked decimal represented by either ASCII or EBCDIC text	zoned

The remainder of this topic consists of an alphabetic listing of all properties. In presenting the syntax, *field_definition* is the name and data type of the field whose properties are defined.

actual_length

Used with null_length ("null_length").

- On import, specifies the actual number of bytes to skip if the field's length equals the null_length
- On export, specifies the number of bytes to fill with the null-field or specified pad character if the exported field contains a null.

Applies to

Nullable fields of all data types; cannot apply to record, subrec, or tagged.

Syntax

```
field_definition {actual_length = length};
```

where *length* is the number of bytes skipped by the import operator or filled with zeros by the export operator when a field has been identified as null.

See also

"null_length" .

Example

In this example:

- On import, the import operator skips the next ten bytes of the imported data if the imported field has a length prefix of 255
- On export, the length prefix is set to 255 if source field a contains a null and the export operator fills the next ten bytes with one of these: zeros, the null-field specification, the pad-character specification for the field.

```
record {prefix = 2} (
    a:nullable string {null_length = 255, actual_length = 10}; )
```

ascii

Specifies that the ASCII character set is used by text-format fields of imported or exported data. This property is equivalent to specifying the US_ASCII character set and is the default.

Applies to

All data types except raw, ustring; record, subrec, or tagged containing at least one non-raw field. For ustring, the same functionality is available using the charset property.

Syntax

```
record { ascii }
field_definition { ascii };
```

This property is mutually exclusive with ebcdic.

See also

"ebcdic" .

Example

The following specification overrides the record-level property setting for field b:

```
record {ebcdic} (
  a:int32;
  b:string {ascii}); )
```

big_endian

Specifies the byte ordering of multi-byte data types in imported or exported data as big-endian.

Applies to

Fields of the integer, date, time, or timestamp data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { big_endian }
  field_definition { big_endian };
```

Restrictions

WebSphere DataStage ignores the endian property of a data type that is not formatted as binary.

This property is mutually exclusive with little_endian and native_endian.

Examples

The following specification defines a record schema using a big-endian representation:

```
record {big_endian, binary, delim = none} (
  a:int32;
  b:int16;
  c:int8; )
```

The following specification overrides the record-level property in a definition of field b.

```
record {big_endian, binary, delim = none} (
  a:int32;
  b:int16 {little_endian};
  c:int8; )
```

binary

Specifies the data representation format of a field in imported or exported data as binary.

Applies to

Fields of all data types except string, ustring, and raw; record, subrec or tagged containing at least one field that is neither string nor raw.

Syntax

```
record { binary }
  field_definition { binary };
```

This option specifies binary data; data is formatted as text by default (see "text").

Meanings

The binary property has different meanings when applied to different data types:

- For decimals, binary means packed (see "packed").
- For other numerical data types, binary means "not text".

- For dates, binary is equivalent to specifying the julian property for the date field (see "julian").
- For time, binary is equivalent to midnight_seconds (see "midnight_seconds").
- For timestamp, binary specifies that the first integer contains a Julian day count for the date portion of the timestamp and the second integer specifies the time portion of the timestamp as the number of seconds from midnight; on export, binary specifies to export a timestamp to two 32-bit integers.

Restrictions

If you specify binary as a property of a numeric field, the data type of an imported or exported field must be the same as the corresponding field defined in a record schema. No type conversions are performed among the different numeric data types (as would be the case if text was specified instead).

This property is mutually exclusive with text, c_format, in_format, and out_format.

Examples

For example, the following defines a schema using binary representation for the imported or exported numeric fields with no delimiter between fields:

```
record { binary, delim = none } (
    a:int32;
    b:int16;
    c:int8; )
"record {text} ( "
```

The following statement overrides the record-level setting of field b as text:

```
record {text} (
    a:int32;
    b:int16 {binary};
    c:int8; )
```

c_format

Perform non-default conversion either of imported data from string to integer or floating-point data or of exported data from integer or floating-point data to a string.

This property specifies a C-language format string used for both import and export. To specify separate import/export strings, see "in_format" and "out_format" .

Applies to

Imported or exported data of the integer or floating-point type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
field_definition { c_format= ' sscanf_or_sprintf_string ' };
```

sscanf_or_sprintf_string is a control string in single quotation marks containing formatting specification for both `sscanf()` to convert string data to floating-point or integer data and `sprintf()` to convert floating-point or integer data to strings. See the appropriate C language documentation for details of this formatting string.

Discussion

A UNIX data file can represent numeric values as strings. By default, on import WebSphere DataStage invokes the C `strtol()`, `strtoll()`, `strtoul()`, `strtoull()`, or `strtod()` function to convert a string to a numeric field.

a WebSphere DataStage data file can represent numeric data in integer and floating-point data types. By default, on export WebSphere DataStage invokes the C `sprintf()` function to convert a numeric field formatted as either integer or floating point data to a string.

If these functions do not output data in a satisfactory format, you can use the `c_format` property to specify a format that is used both by the `sscanf()` function to convert string data to integer or floating point data and by the `sprintf()` function to convert integral or floating point data to string data.

The associated imported/exported field must represent numeric data as text.

`Int64` values map to long long integers on all supported platforms.

For all platforms the `c_format` is:

```
'%[ padding_character ][ integer ]lld'
```

The `integer` component specifies a minimum field width. The output column is printed at least this wide, and wider if necessary. If the column has fewer digits than the field width, it is padded on the left with `padding_character` to make up the field width. The default padding character is a space.

Examples

- For this example `c_format` specification:

```
'%09lld'
```

the padding character is zero (0), and the integers 123456 and 12345678 are printed out as 000123456 and 123456789.

- For this example:

```
record (a:int32 {c_format = '%x', width = 8};)
```

the meaning of the expression varies:

- On import, the expression uses the `c_format` property to ensure that string data in the imported file is formatted in the WebSphere DataStage record as field `a`, a 32-bit integer formatted as an 8-byte hexadecimal string
- On export, the same expression ensures that the field `a`, consisting of a 32-bit integer, is exported from the WebSphere DataStage record as an 8-byte hexadecimal string.

charset

Specifies a character set defined by the International Components for Unicode (ICU).

Applies to

At the field level, this option applies only to `ustrings`. At the record level, this option applies to the fields that do not specify a character set and to these properties that support multi-byte Unicode character data:

```
delim delim_string record_delim record_delim_string final_delim final_delim_string quote default  
padchar null_field date_format time_format timestamp_format
```

Syntax

```
record { charset = charset }  
field_definition { charset = charset };
```

Example

```
record { charset=charset1 }  
(a:cstring { charset=ISO-8859-15 } {delim = xxx}, b:date { charset=ISO-8859-15 }, c:cstring)
```

Where the user defined record charset, charset1, applies to field c and to the delim specification for field a. ISO-8859-15 applies to field a and b. Notice that the field character setting for field a does not apply to its delim property.

check_intact

Used only with the intact property, performs error checking when records are imported with a partial record schema.

Applies to

record, if it is qualified by the intact record property. (See "intact").

Syntax

```
record { intact, check_intact }
```

By default, when WebSphere DataStage imports records with a partial schema, it does not perform error checking in order to maximize the import speed. This property overrides the default. Error-checking in this case verifies, during the import operation, that the record contents conform to the field description. In addition, downstream of this verification, the operator that acts on the input fields verifies their value.

Example

For example, the following statement uses intact to define a partial record schema and uses check_intact to direct WebSphere DataStage to perform error checking:

```
record {intact=rName, check_intact, record_length=82,  
record_delim_string='\r\n'}()
```

date_format

Specifies the format of an imported or exported text-format date.

Applies to

Field of date data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { date_format = format_string | uformat }
```

uformat is described in "default_date_format" .

The *format_string* can contain one or a combination of the following elements:

Table 72. Date format tags

Tag	Variable width availability	Description	Value range	Options
%d	import	Day of month, variable width	1...31	s
%dd		Day of month, fixed width	01...31	s
%ddd	with v option	Day of year	1...366	s, v
%m	import	Month of year, variable width	1...12	s
%mm		Month of year, fixed width	01...12	s

Table 72. Date format tags (continued)

Tag	Variable width availability	Description	Value range	Options
%mmm		Month of year, short name, locale specific	Jan, Feb ...	t, u, w
%mmmm	import/export	Month of year, full name, locale specific	January, February ...	t, u, w, -N, +N
%yy		Year of century	00...99	s
%yyyy		Four digit year	0001 ...9999	
%NNNNyy		Cutoff year plus year of century	yy = 00...99	s
%e		Day of week, Sunday = day 1	1...7	
%E		Day of week, Monday = day 1	1...7	
%eee		Weekday short name, locale specific	Sun, Mon ...	t, u, w
%eeee	import/export	Weekday long name, locale specific	Sunday, Monday ...	t, u, w, -N, +N
%W	import	Week of year (ISO 8601, Mon)	1...53	s
%WW		Week of year (ISO 8601, Mon)	01...53	s

When you specify a date format string, prefix each component with the percent symbol (%) and separate the string's components with a suitable literal character.

The default date_format is %yyyy-%mm-%dd.

Where indicated the tags can represent variable-width data elements. Variable-width date elements can omit leading zeroes without causing errors.

The following options can be used in the format string where indicated in the table:

s Specify this option to allow leading spaces in date formats. The s option is specified in the form:

`%(tag,s)`

Where *tag* is the format string. For example:

`%(m,s)`

indicates a numeric month of year field in which values can contain leading spaces or zeroes and be one or two characters wide. If you specified the following date format property:

`%(d,s)/%(m,s)/%yyyy`

Then the following dates would all be valid:

8/ 8/1958

08/08/1958

8/8/1958

v Use this option in conjunction with the %ddd tag to represent day of year in variable-width format. So the following date property:

`%ddd,v`

represents values in the range 1 to 366. (If you omit the v option then the range of values would be 001 to 366.)

- u** Use this option to render uppercase text on output.
- w** Use this option to render lowercase text on output.
- t** Use this option to render titlecase text (initial capitals) on output.

The u, w, and t options are mutually exclusive. They affect how text is formatted for output. Input dates will still be correctly interpreted regardless of case.

- N** Specify this option to left justify long day or month names so that the other elements in the date will be aligned.
- +N** Specify this option to right justify long day or month names so that the other elements in the date will be aligned.

Names are left justified or right justified within a fixed width field of N characters (where N is between 1 and 99). Names will be truncated if necessary. The following are examples of justification in use:

`%dd-%(mmmm,-5)-%yyyyy`

21-Augus-2006

`%dd-%(mmmm,-10)-%yyyyy`

21-August -2005

`%dd-%(mmmm,+10)-%yyyyy`

21- August-2005

The locale for determining the setting of the day and month names can be controlled through the locale tag. This has the format:

`%(L,'locale')`

Where *locale* specifies the locale to be set using the *language_COUNTRY.variant* naming convention supported by ICU. See *NLS Guide* for a list of locales. The default locale for month names and weekday names markers is English unless overridden by a %L tag or the APT_IMPEXP_LOCALE environment variable (the tag takes precedence over the environment variable if both are set).

Use the locale tag in conjunction with your time format, for example the format string:

`%(L,'es')%eeee, %dd %mmmm %yyyy`

Specifies the Spanish locale and would result in a date with the following format:

miércoles, 21 septiembre 2005

The format string is subject to the restrictions laid out in the following table. A format string can contain at most one tag from each row. In addition some rows are mutually incompatible, as indicated in the 'incompatible with' column. When some tags are used the format string requires that other tags are present too, as indicated in the 'requires' column.

Table 73. Format tag restrictions

Element	Numeric format tags	Text format tags	Requires	Incompatible with
year	%yyyy, %yy, %[nnnn]yy	-	-	-

Table 73. Format tag restrictions (continued)

Element	Numeric format tags	Text format tags	Requires	Incompatible with
month	%mm, %m	%mmm, %mmmm	year	week of year
day of month	%dd, %d	-	month	day of week, week of year
day of year	%ddd		year	day of month, day of week, week of year
day of week	%e, %E	%eee, %eeee	month, week of year	day of year
week of year	%WW		year	month, day of month, day of year

When a numeric variable-width input tag such as %d or %m is used, the field to the immediate right of the tag (if any) in the format string cannot be either a numeric tag, or a literal substring that starts with a digit. For example, all of the following format strings are invalid because of this restriction:

%d%m-%yyyy

%d%mm-%yyyy

%(d)%(%mm)-%yyyy

%h00 hours

The *year_cutoff* is the year defining the beginning of the century in which all two-digit years fall. By default, the year cutoff is 1900; therefore, a two-digit year of 97 represents 1997.

You can specify any four-digit year as the year cutoff. All two-digit years then specify the next possible year ending in the specified two digits that is the same or greater than the cutoff. For example, if you set the year cutoff to 1930, the two-digit year 30 corresponds to 1930, and the two-digit year 29 corresponds to 2029.

On import and export, the *year_cutoff* is the base year.

This property is mutually exclusive with *days_since*, *text*, and *julian*.

You can include literal text in your date format. Any Unicode character other than null, backslash, or the percent sign can be used (although it is better to avoid control codes and other non-graphic characters). The following table lists special tags and escape sequences:

Tag	Escape sequence
%%	literal percent sign
\%	literal percent sign
\n	newline
\t	horizontal tab
\\"	single backslash

If the format string does not include a day, the day is set to the first of the month in the destination field. If the format string does not include the month and day, they default to January 1. Note that the format string must contain a month if it also contains a day; that is, you cannot omit only the month.

When using variable-width tags, it is good practice to enclose the date string in quotes. For example, the following schema:

```
f1:int32; f2:date { date_format='%eeee %mmmm %d, %yyyy', quote='double' }; f3:int32;
```

would ensure that the following records are processed correctly:

```
01 "Saturday November 5, 2005" 1000  
02 "Sunday November 6, 2005" 1001
```

The quotes are required because the parallel engine assumes that variable-width fields are space delimited and so might interpret a legitimate space in the date string as the end of the date.

See also

See the section on WebSphere DataStage data types in the *WebSphere DataStage Parallel Job Developer Guide* for more information on date formats.

days_since

The imported or exported field stores the date as a signed integer containing the number of days since *date_in_ISO_format* or *uformat*.

Applies to

Fields of the date data type ; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { days_since = date_in_ISO_format | uformat }
```

where *date_in_ISO_format* is in the form %yyyy-%mm-%dd; and *uformat* is the default date format as described in "default_date_format". The imported or exported field is always stored as binary data.

This property is mutually exclusive with *date_format*, *julian*, and *text*.

decimal_separator

Specifies an ASCII character to separate the integer and fraction components of a decimal.

Applies to

record and decimal fields.

Syntax

```
record { decimal_separator = ASCII_character }  
field_definition { decimal_separator = ASCII_character };
```

Example

```
record { decimal_separator = ',' }  
(a:decimal, b:decimal { decimal_separator = '-' })
```

where the decimal separator for field a is ',' and is '-' for field b.

default

Sets the default value of an imported or exported field:

- On import if the generate property is set
- On both import and export if an error occurs

Applies to

Fields of integer, float, string, ustring, and decimal data types. Cannot apply to record, subrecord, or tagged.

Syntax

```
field_definition { default = field_value };
```

where *field_value* is the default value of an imported or exported field. For ustring fields, *field_value* might contain multi-byte Unicode characters.

On import, if you specify a default value, the import operator sets the destination field in the data set to the default value instead of rejecting an erroneous record. On export, the export operator sets the field in the destination data file to the default value instead of rejecting an erroneous record.

The property is mutually exclusive with link.

Example

The following defines the default value of field c as 127:

```
record (a:int32; b:int16; c:int8 {default = 127});
```

default_date_format

This property provides support for international date components in date fields.

Applies to

record and date fields.

Syntax

```
record {default_date_format = String%macroString%macroString%macroString}
```

where *%macro* is a formatting macro such as %mmm for a 3-character English month. The section "date_format" lists the date formatting macros. The String components can be strings or ustrings.

Example

```
record ( { default_date_format = jeudi%ddaoût%yyyy } );
```

default_time_format

This property provides support for international time components in time fields.

Applies to

record and time fields.

Syntax

```
record {default_time_format = String%macroString%macroString%macroString}
```

where *%macro* is a formatting macro such as %hh for a two-digit hour. The section "time_format" lists the time formatting macros. The String components can be strings or ustrings.

Example

```
record ( { default_time_format = %hh&nnA.M. } );
```

delim

Specifies the trailing delimiter of all fields in the record.

Applies to

record; any field of the record.

Syntax

```
record { delim = delim_char }  
field_definition { delim = delim_char };
```

where *delim_char* can be one of the following:

- ws to have the import operator skip all standard whitespace characters (space, tab, and newline) trailing after a field.
- end to specify that the last field in the record is composed of all remaining bytes until the end of the record.
- none to specify that fields have no delimiter.
- null to specify that the delimiter is the null character.
- You can specify an ASCII or multi-byte Unicode character enclosed in single quotation marks. To specify multiple characters, use *delim_string* (see "delim_string").

Discussion

By default, the import operator assumes that all fields but the last are whitespace delimited.

By default, the export operator inserts a space after every exported field except the last. On export, you can use *delim* to specify the trailing delimiter that the export operator writes after each field in the destination data file. Specify:

- ws to have the export operator insert an ASCII space (0x20) after each field
- none to write a field with no delimiter and no length prefix.

You can use a backslash (\) as an escape character to specify special characters. For example, \t represents an ASCII tab delimiter character.

You can use *final_delim* to specify a different delimiter for the last field of the record. See "final_delim" .

Mutually exclusive with *prefix*, *delim_string*, and *reference*.

Examples

The following statement specifies that all fields have no delimiter:

```
record {delim = none} (  
    a:int32;  
    b:string;  
    c:int8;  
    d:raw; )
```

The following statement specifies a comma as a delimiter at record-level but overrides this setting for field d, which is composed entirely of bytes until the end of the record:

```

record {delim = ','} (
    a:int32;
    b:string;
    c:int8;
    d:raw{delim = end}; )
"record {delim = ','} ( "

```

Note that in this example, the record uses the default record delimiter of a newline character. This means that:

- On import, field d contains all the bytes to the newline character.
- On export, a newline character is inserted at the end of each record.

delim_string

Like `delim`, but specifies a string of one or more ASCII or multi-byte Unicode characters forming a trailing delimiter.

Applies to

record; any field of the record.

Syntax

```

record {delim_string = ' ASCII_string ' | 'multi_byte_Unicode_string' }
field_definition { delim_string = ' ASCII_string ' | 'multi_byte_Unicode_string' };

```

You can use a backslash (\) as an escape character to specify special characters within a string. For example, \t represents an ASCII tab delimiter character. Enclose the string in single quotation marks.

This property is mutually exclusive with `prefix`, `delim`, and `reference`.

Note: Even if you have specified the character set as EBCDIC, `ASCII_string` is always interpreted as ASCII character(s).

Import and export behavior

Import behavior and export behavior differ:

- On import, the import operator skips exactly one instance of a trailing `ASCII_string`.
- On export, you use `delim_string` to specify the trailing delimiter that the export operator writes after each field in the destination data file. If you specify none, a field is written with no delimiter.

Examples

The following statement specifies that all fields are delimited by a comma followed by a space:

```

record {delim_string = ', '} (
    a:int32;
    b:string;
    c:int8;
    d:raw; )

```

In the following example, the delimiter setting of one comma is overridden for field b, which will be delimited by a comma followed by an ASCII space character, and for field d, which will be delimited by the end of the record:

```

record {delim = ','} (
    a:int32;
    b:string {delim_string = ', '};
    c:int8;
    d:raw {delim = end}; )

```

Note that in this example, the record uses the default record delimiter of a newline character. This means that:

- On import field d contains all the bytes to the newline character.
- On export a newline character is inserted at the end of each record.

drop

Specifies a field to be dropped on import and not stored in the WebSphere DataStage data set.

Applies to

Fields of all data types. Cannot apply to record, subrec, or tagged.

Syntax

```
field_definition { drop };
```

You can use this property when you must fully define the layout of your imported data in order to specify field locations and boundaries but do not want to use all the data in the WebSphere DataStage job.

Restrictions

This property is valid for import only and is ignored on export.

This property is mutually exclusive with link.

See also

"padchar" .

Example

In this example, the variable-length string field, b, is skipped as the record is imported.

```
record (a:int32; b:string {drop}; c:int16);
```

In the following example, all fields are written as strings to the WebSphere DataStage data set on import and are delimited by a comma except the last field; records are delimited by the default newline character. The last four fields are dropped from every record on import, and generated on export. This technique is useful when fields of the source data are not be processed in an WebSphere DataStage job but place holders for those fields must be retained in the resultant file containing the exported results. In addition, all bytes of the generated fields in the exported data file will be filled with ASCII spaces, as defined by the padchar property:

```
record { delim = ',' }
(
    first_name      :string[20];
    middle_init     :string[1];
    last_name       :string[40];
    street          :string[40];
    apt_num         :string[4];
    city            :string[30];
    state           :string[2];
    prev_street     :string[40] {drop, generate, padchar=' '};
    prev_apt_num    :string[4] {drop, generate, padchar=' '};
    prev_city       :string[30] {drop, generate, padchar=' '};
    prev_state      :string[2] {delim = end, drop, generate, padchar=' '};
```

ebcdic

Specifies that the EBCDIC character set is used by text-format fields of imported or exported data. Setting this property is equivalent to specifying the EBCDIC equivalent of the US_ASCII character set.

Applies to

All data types except raw and ustring; record, subrec, or tagged if it contains at least one field of this type. For ustring, the same functionality is available through the charset property.

Syntax

```
record { ebcdic }  
field_definition { ebcdic };
```

This property is mutually exclusive with ascii.

WebSphere DataStage's default character set is ASCII-formatted. (WebSphere DataStage supplies lookup tables for converting between ASCII and EBCDIC. See "ASCII and EBCDIC Conversion Tables" for more information.)

See also

"ascii" ."

export_ebcdic_as_ascii

Export a string formatted in the EBCDIC character set as an ASCII string.

Applies to

Fields of the string data type and records; record, subrec, or tagged if it contains at least one field of this type. This property does not apply to ustring; you can obtain the same functionality using the charset property.

Syntax

```
record { export_ebcdic_as_ascii }  
field_definition { export_ebcdic_as_ascii };
```

This property cannot be used on import.

fill

Specifies the byte value used to fill the gaps between fields of an exported record caused by field positioning.

Applies to

record; cannot be specified for individual fields.

Syntax

```
record {fill [ = fill_value ] }
```

where *fill_value* specifies a byte value that fills in gaps. By default the fill value is 0. The *fill_value* can also be one of these:

- a character or string in single quotation marks
- an integer between 0 and 255.

Restriction

You cannot override the record-level fill property with an individual field definition.

Examples

In the following example, the two-byte gap between fields a and b are to be filled with ASCII spaces:

```
record {fill = ' '} (a:int32; b:int16 {skip = 2};)
```

In the following example, the gaps are to be filled with the value 127:

```
record { fill = 127} (a:int32; b:int16 {skip = 2};)
```

final_delim

Specifies a delimiter for the last field. If specified, *final_delim* precedes the *record_delim* if both are defined.

Applies To

The last field of a record. When the last field in a record is a subrec, a space is added after the subrec instead of your non-space *delim_value*.

Syntax

```
record {final_delim = delim_value}
```

where *delim_value* is one of the following:

- ws (a white space)
- end (end of record, the default)
- none (no delimiter, field length is used)
- null (0x00 character)
- 'a' (specified ASCII or multi-byte Unicode character, enclosed in single quotation marks)

Example

In this example, commas delimit all fields except the last. Since end is specified as the *final_delim*, the record delimiter serves as the delimiter of the final field. This is the newline character. Note that it is not specified as the *record_delim*, because newline-delimited records are the default.

```
record
  {delim = ',', final_delim = end}
  (
    checkingAccountStatus :int32;
    durationOfAccount :sfloat;
    creditHistory :int32;
    purposeOfLoan :int32;
    creditAmount :sfloat;
    savingsAccountAmount :int32;
    yearsPresentlyEmployed :int32;
    installmentRate :sfloat;
  )
```

By default, on export a space is now inserted after every field except the last field in the record. Previous to this release, a space was inserted after every field, including the last field; or, when *delim* property was set, its value was used instead of the *final_delim* value.

Now when you specify the *final_delim* property for records that have a tagged or subrec field as the last field, your specification is correctly applied unless the subrec is a vector. By default, a space is added to the last field when it is a subrec vector.

You can set the APT_PREVIOUS_FINAL_DELIM_COMPATIBLE environment variable to obtain the *final_delim* behavior prior to this release.

final_delim_string

Like *final_delim*, but specifies a string of one or more ASCII or multi-byte Unicode characters that are the delimiter string of the last field of the record. The *final_delim_string* property precedes the *record_delim* property, if both are used.

Applies to

The last field of a record.

Syntax

```
record {final_delim_string = ' ASCII_string ' | ' multi_byte_Unicode_string ' }
```

where string characters are enclosed in single quotation marks.

Import and export

Behavior on import and export varies:

- On import, the import operator skips the delimiter in the source data file. The import operator skips exactly one instance of a trailing *ASCII_string* during import.
- On export, *final_delim_string* specifies the trailing delimiter that the export operator writes after the last field of each record.

Even if you have specified the character set as EBCDIC, '*ASCII_string*' is always interpreted as ASCII characters.

Example

For example, the following statement specifies that all fields are delimited by a comma, except the final field, which is delimited by a comma followed by an ASCII space character:

```
record {delim_string = ',', final_delim_string = ', ', } (
    a:int32;
    b:string;
    c:int8;
    d:raw; )
```

fix_zero

Treat a packed decimal field containing all zeros (normally illegal) as a valid representation of zero.

Applies to

Fields of the packed decimal data type on import; all decimal fields on export (exported decimals are always packed); record, subrec, or tagged if it contains at least one field of these types.

Syntax

```
field_definition { fix_zero };
```

Note: Omitting *fix_zero* causes the job to generate an error if it encounters a decimal containing all zeros.

This property overrides the nocheck option to packed.

See also

"packed" and "nofix_zero" . The latter option restores default behavior.

generate

On export, creates a field and sets it to the default value.

Applies to

Fields of all data types. Cannot apply to record, subrec, or tagged.

Syntax

```
field_definition { generate, default = default_value };
```

where:

- default is the ordinary default property (see "default") and not a sub-option of generate
- *default_value* is the optional default value of the field that is generated; if you do not specify a *default_value*, WebSphere DataStage writes the default value associated with that data type.

Discussion

You can specify both the drop property and the generate property in the same property list. When the schema is used for import it drops the field, and when it is used for export, it generates the field.

This type of statement is useful if the field is not important when the data set is processed but you must maintain a place holder for the dropped field in the exported data.

Restriction

This property is valid for export only and is ignored on import.

Examples

The following statement creates a new field in an exported data set:

```
record ( a:int32; b:int16 {generate, default=0}; c:int8; )
```

The following statement causes field b to be:

- dropped on import
- generated as a 16-bit integer field, which is initialized to 0, on export.

```
record ( a:int32; b:int16 {drop, generate, default =0}; c:int8; )
```

import_ascii_as_ebcdic

Translate a string field from ASCII to EBCDIC as part of an import operation. This property makes it possible to import EBCDIC data into WebSphere DataStage from ASCII text.

Applies to

Fields of the string data type; record, subrec, or tagged if it contains at least one field of this type. This property does not apply to ustring; you can obtain the same functionality using the charset property.

Syntax

```
field_definition { import_ascii_as_ebcdic };
```

Restriction

Use only in import operations. On export, you can use export_ebcdic_as_ascii to export the EBCDIC field back to ASCII. See "export_ebcdic_as_ascii".

Example

In the following example, the string field *x* contains an ASCII string that is converted to EBCDIC on import:

```
record (x:string[20] {import_ascii_as_ebcdic});
```

in_format

On import, perform non-default conversion of an input string to an integer or floating-point data.

This property specifies a C-language format string used for import. See also "out_format".

Applies to

Imported data of integer or floating-point type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
field_definition { in_format = ' sscanff_string' };
```

where *sscanf_string* is a control string in single quotation marks interpreted by the C scanf() function to convert character data to numeric data. See the appropriate C language documentation for details of this formatting string.

Discussion

A UNIX data file can represent numeric values as strings. By default, on import WebSphere DataStage invokes the C strtol(), strtoll(), strtoul(), strtoull(), or strtod() function to convert a string to a numeric field. If these functions do not output data in a satisfactory format, you can specify the out_format property. When you do, WebSphere DataStage invokes the sprintf() function to convert the string to numeric data. You pass formatting arguments to the function.

When strings are converted to 8-, 16-, or 32-bit signed integers, the *sscanf_string* must specify options as if the generated field were a 32-bit signed integer. WebSphere DataStage converts the 32-bit integer to the proper destination format.

Restrictions

The export operator ignores this property; use it only for import operations.

This property is mutually exclusive with binary.

Example

The following statement assumes that the data to be converted is represented as a string in the input flat file. The data is imported as field *a*, a 32-bit integer formatted as an 8-byte hexadecimal string:

```
record (a:int32 {in_format = '%x', width=8});
```

intact

Define a partial record schema.

Applies to

record

Syntax

```
record {intact[= rName] ... } ( field_definitions ; )
```

where *rName* is an optional identifier of the intact schema.

See also

"check_intact".

Example

For example, the following statement uses intact to define a partial record schema:

```
record {intact=rName, record_length=82, record_delim_string='\r\n'} ()
```

In this example, the record schema defines an 82-byte record (80 bytes of data, plus one byte each for the carriage return and line feed characters delimiting the record). On import, the two bytes for the carriage return and line feed characters are removed from the record and not saved; therefore, each record of the data set contains only the 80 bytes of data.

julian

Specifies that the imported or exported field represents the date as a binary numeric value denoting the Julian day.

Applies to

Fields of the date data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { binary, julian }
field_definition { binary, julian };
```

Note: The imported or exported value is always represented as binary data.

A Julian day specifies the date as the number of days from 4713 BCE January 1, 12:00 hours (noon) GMT. For example, January 1, 1998 is Julian day count 2,450,815. In this context, binary has the same meaning as julian.

This property is mutually exclusive with days_since, date_format, and text.

link

Specifies that a field holds either the length of another (variable-length) field in the record or the length of the tag value of a tagged.

Applies to

Numeric or string fields; cannot apply to a record, subrec, or tagged.

Syntax

```
field_definition { link };
```

Discussion

A variable-length field must specify the number of elements it contains by means either of a prefix of the vector (see "vector_prefix") or a link to another field.

- On import, the link field is dropped and does not appear in the schema of the imported data set.
- On export, the link field is not exported from the data set, but is generated from the length of the field referencing it and then exported.

The data type of a field defined by the link import/export property must:

- Be a uint32 or data type that can be converted to uint32; see "Modify Operator" for information on data type conversions.
- Have a fixed-length external format on export.
- Not be a vector.

This property is mutually exclusive with drop.

Examples

In this example, field a contains the element length of field c, a variable-length vector field.

```
record (a:uint32 {link}; b:int16; c[]:int16 {reference = a});  
"record (a:uint32 {link}; b:int16; c:string {reference =  
a});"
```

In this example, field a contains the length of field c, a variable-length string field.

```
record (a:uint32 {link}; b:int16; c:string {reference = a});
```

little_endian

Specify the byte ordering of multi-byte data types as little-endian. The default mode is native-endian.

Applies to

Fields of the integer, date, time, or timestamp data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { little_endian }  
field_definition { little_endian };
```

This property is mutually exclusive with big_endian and native_endian.

Note: WebSphere DataStage ignores the endian property if a data type is not formatted as binary.

Examples

The following specification defines a schema using a little-endian representation:

```
record {little_endian, binary, delim = none} (  
    a:int32;  
    b:int16;  
    c:int8; )
```

The following declaration overrides the big_endian record-level property in a definition of field b:

```
record {big_endian, binary, delim = none} (
    a:int32;
    b:int16 {little_endian};
    c:int8; )
```

max_width

Specifies the maximum number of 8-bit bytes of an imported or exported text-format field. Base your width specification on the value of your -impexp_charset option setting. If it's a fixed-width charset, you can calculate the maximum number of bytes exactly. If it's a variable length encoding, calculate an adequate maximum width for your fields.

Applies to

Fields of all data types except date, time, timestamp, and raw; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { max_width = n }
field_definition { max_width = n };
```

where n is the maximum number of bytes in the field; you can specify a maximum width of 255 bytes.

This property is useful for a numeric field stored in the source or destination file in a text representation.

If you specify neither width nor max_width, numeric fields exported as text have the following number of bytes as their maximum width:

- 8-bit signed or unsigned integers: 4 bytes
- 16-bit signed or unsigned integers: 6 bytes
- 32-bit signed or unsigned integers: 11 bytes
- 64-bit signed or unsigned integers: 21 bytes.
- single-precision float: 14 bytes (sign, digit, decimal point, 7 fraction, "E", sign, 2 exponent)
- double-precision float: 24 bytes (sign, digit, decimal point, 16 fraction, "E", sign, 3 exponent)

Restriction

On export, if you specify the max_width property with a dfloat field, the max_width must be at least eight characters long.

midnight_seconds

Represent the imported or exported time field as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.

Applies to

The time data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { binary, midnight_seconds };
field_definition { binary, midnight_seconds };
```

Note: The imported or exported integer is always stored as binary data.

This property is mutually exclusive with time_format and text.

native_endian

Specify the byte ordering of multi-byte data types as native-endian. This is the default mode for import/export operations.

Applies to

Fields of the integer, date, time, or timestamp data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { native_endian }  
field_definition { native_endian };
```

where native_endian specifies that all multi-byte data types are formatted as defined by the native format of the machine.

This property is mutually exclusive with big_endian and little_endian.

Note: WebSphere DataStage ignores the endian property of a data type that is not formatted as binary.

Examples

The following defines a schema using a native-endian representation:

```
record {native_endian, binary, delim = none} (  
    a:int32;  
    b:int16;  
    c:int8; )
```

nofix_zero

Generate an error when a packed decimal field containing all zeros is encountered. This is the default behavior of import and export.

Applies to

Import behavior and export behavior differ:

- Fields of the packed decimal data type on import
- All decimal fields on export (exported decimals are always packed); record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
field_definition { nofix_zero };
```

null_field

Specifies the value representing null.

Applies to

Fields whose data type is nullable; cannot apply to record, subrec, or tagged.

Syntax

```
field_definition { null_field = 'byte_value' | 'multi_byte_Unicode_value' };
```

where *byte_value* or *multi_byte_Unicode_value* is:

- On import, the value given to a field containing a null;
- On export, the value given to an exported field if the source field is set to null.

The *byte_value* or *multi_byte_Unicode_value* can take one of these forms:

- A number or string that defines the value to be written if the field contains a null. Enclose the string or *wstring* in single quotation marks.
- A standard C-style string literal escape character. For example, you can represent a byte value by \ooo, where each o is an octal digit 0 - 7 and the first o is < 4, or by \xhh, where each h is a hexadecimal digit 0 - F. You must use this form to encode non-printable byte values.

When you are specifying the *null_field* property at the record level, it is important to also specify the field-level *null_field* property to any nullable fields not covered by the record-level property.

For example:

```
record { null_field = 'aaaa' }
  ( field1:nullable int8 { null_field = '-127' };
    field2:nullable string[4];
    field3:nullable string;
    field4:nullable wstring[8] {null_field = 'ââââââââ' }; )
```

The record-level property above applies only to variable-length strings and fixed-length strings of four characters, *field2* and *field3*; *field1* and *field4* are given field-level *null_field* properties because they are not covered by the record property.

This property is mutually exclusive with *null_length* and *actual_length*.

The *null_field* parameter can be used for imported or exported data. For a fixed-width data representation, you can use *padchar* to specify a repeated trailing character if *byte_value* is shorter than the fixed width of the field.

null_length

Specifies the value of the length prefix of a variable-length field that contains a null.

Applies to

Fields whose data type is nullable; cannot apply to record, subrec, or tagged.

Syntax

```
field_definition { null_length= length }
```

where *length* is the length in bytes of a variable-length field that contains a null.

When a variable-length field is imported, a length of *null_length* in the source field indicates that it contains a null. When a variable-length field is exported, the export operator writes a length value of *null_length* if it contains a null.

This property is mutually exclusive with *null_field*.

Restriction

Specifying *null_length* with a non-nullable WebSphere DataStage field generates an error.

See also

"*actual_length*"

Example

For example, the following schema defines a nullable, variable-length string field, prefixed by a two-byte length:

```
record {prefix = 2} (
    a:nullable string {null_length = 255}; )
```

Import and export results differ:

- On import, the imported field is assumed to be zero length and to contain a null if the length prefix contains 255; field *a* of the imported data is set to null.
- On export, the length prefix is set to 255 and the length of the actual destination is zero if field *a* contains null.

out_format

On export, perform non-default conversion of an integer or floating-point data type to a string.

This property specifies a C-language format string used for export of a text field. See also "in_format".

Applies to

Integer or floating-point data to be exported to numeric strings.

Syntax

```
field_definition { out_format = ' sprintf_string '};
```

where *sprintf_string* is a control string in single quotation marks containing formatting specification for sprintf() to convert floating-point or integer data to strings. See the appropriate C language documentation for details of this formatting string.

When 8-, 16-, or 32-bit signed integers are converted to strings, the *sprintf_string* must specify options as if the source field were a 32-bit integer. WebSphere DataStage converts the source field to a 32-bit signed integer before passing the value to sprintf(). For 8-, 16-, and 32-bit unsigned integers, specify options as if the source field were a 32-bit unsigned integer.

Discussion

An WebSphere DataStage data file can represent numeric data in integer and floating-point data types. By default, on export WebSphere DataStage invokes the C sprintf() function to convert a numeric field formatted as either integer or floating point data to a string. If this function does not output data in a satisfactory format, you can specify the out_format property. When you do, WebSphere DataStage invokes the sprintf() function to convert the numeric data to a string. You pass formatting arguments to the function.

Restrictions

The import operator ignores this property; use it only for export operations.

The property is mutually exclusive with binary.

Example

The following statement defines an exported record containing two integer values written to the exported data file as a string:

```

record (
    a:int32 {out_format = '%x', width = 8};
    b:int16 {out_format = '%x', width = 4};
)

```

overpunch

Specifies that the imported or exported decimal field has a leading or ending byte that contains a character which specifies both the numeric value of that byte and whether the number as a whole is negatively or positively signed. This representation eliminated the need to add a minus or plus sign to the beginning or end of a number. All the digits besides the overpunched byte represent normal numbers.

Use one of these formats:

- To indicate that the overpunched value is in the leading byte, use this syntax:
`{overpunch}`
For example, in the overpunched number B567, B indicates that the leading byte has a value of 2 and that the decimal is positively signed. It is imported as 2567.
- To indicate that the overpunched value is in the last byte, use this syntax:
`{trailing, overpunch}`
For example, in the overpunched number 567K, K indicates that the last byte has a value of 2 and that the number is negatively signed. It is imported as -5672.

Applies to

Decimal fields.

Syntax

```

record { { overpunch | trailing, overpunch} option_1 .... option_n }
field_definition { { overpunch | trailing, overpunch} option_1 ,... option_n };

```

packed

Specifies that the imported or exported field contains a packed decimal.

Applies to

Fields of the decimal, string, and ustring data types; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```

record { packed option_1 .... option_n }
field_definition { packed option_1 ,...option_n };

```

where *option_1 ... option_n* is a list of one or more options; separate options with a comma if there are more than one. The options are:

- check. (default) perform the WebSphere DataStage data verification process on import/export. Note that this property is ignored if you specify fix_zero.
- nocheck. bypass the WebSphere DataStage data verification process on import/export. Note that this property is ignored if you specify fix_zero.

The options check and nocheck are mutually exclusive.

- signed (default) use the sign of the source decimal on import or export.
- unsigned generate a sign nibble of 0xf, meaning a positive value, regardless of the imported or exported field's actual sign. This format corresponds to the COBOL PICTURE 999 format (as opposed to S999).

The options signed and unsigned are mutually exclusive.

Discussion

The precision and scale of either the source decimal on import or the destination decimal on export defaults to the precision and scale of the WebSphere DataStage decimal field. Use the precision and scale properties to override these defaults.

For example, the following schema specifies the WebSphere DataStage decimal field has a precision of 5 and a scale of 2:

```
record {packed} ( a:decimal[5,2]; )
```

Import and export results differ:

- On import, field a is imported from a packed decimal representation three bytes long.
- On export, the field is written to three bytes in the destination.

padchar

Specifies a pad character used when WebSphere DataStage strings or numeric values are exported to an external string representation.

Applies to

string, ustring, and numeric data types on export; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { padchar = `char(s)` ` }  
field_definition { padchar = `char(s)` ` }
```

where *char(s)* is one or more pad characters, single byte for string fields and multi-byte Unicode for ustring fields. You can specify null to set the pad character to 0x00; the default pad character is 0x20 (ASCII space). Enclose the pad character in single quotation marks if it is not the value null.

The pad character is used when the external string representation is larger than required to hold the exported field. In this case, the external string is filled with the pad character to its full length.

Restrictions

This property is ignored on import.

Example

In the following example, the destination string will be padded with spaces to the full length, if the string field *a* is less than 40 bytes long.

```
record (a:string {width = 40, padchar = ' '};)
```

Notes

WebSphere DataStage fixed-length strings also have a padchar property, which is part of a schema rather than an import/export property, as in the following example. Here the exported fixed-length string is also padded with spaces to the length of the external field:

```
record (a:string[20, padchar = ' '] {width = 40};)
```

You can globally override the default pad character using the WebSphere DataStage environment variable APT_STRING_PADCHAR. The Option and Syntax of this environment variable is shown below:

```
export APT_STRING_PADCHAR='character'                                # ksh  
setenv APT_STRING_PADCHAR 'character'                                # csh
```

where *character* is the default pad character enclosed in single quotation marks.

position

Specifies the starting position of a field in the imported source record or exported destination record. The starting position can be either an absolute byte offset from the first record position (0) or the starting position of another field.

Applies to

Fields of all data types; cannot apply to record, subrec, or tagged.

Syntax

```
field_definition { position = byte_offset | field_name };
```

where:

- *byte_offset* is an integer value that is greater than or equal to 0 and less than the record length and that indicates the absolute byte offset of the field from position 0 in the record
- *field_name* specifies the name of another field in the record at whose beginning boundary the defined field also starts, as in the example below.

Discussion

Specifies the byte offset of a field in the imported source record or exported destination record, where the offset of the first byte is 0. If you omit this property, a field starts immediately after the preceding field. Note that a field can start at a position preceding the end of the previous field.

Examples

For example, the following defines a schema using this property:

```
record {binary, delim = none} (a:int32; b:int16 {position = 6};)
```

Import and export results differ:

- On import, field b starts at absolute byte offset 6 in the source record, that is, there is a 2-byte gap between fields a and b. Note that numeric values are represented in a binary format in this example. The default numeric format is text.
- On export, the export operator skips two bytes after field a, then writes field b. By default on export, any skipped bytes are set to zero in the destination record. You can use the record-level fill property ("fill") to specify a value for the skipped bytes.

You can specify a *field_name* as a position, as in the following example:

```
record {binary, delim = none} (  
    a:string {delim = ws};  
    b:int16;  
    c:raw[2] {position = b}; )  
"record {binary, delim = none} ( "
```

Field c starts at field b. Import behavior and export behavior differ:

- On import, the schema creates two fields from field b of the imported record, interpreting the same external value using two different field types.

- On export, the schema first writes the contents of field b to destination field b and then to destination field c.

precision

Import behavior and export behavior differ:

- On import, specifies the precision of a source packed decimal.
- On export, specifies the precision of a destination string when a decimal is exported in text format.

Applies to

Imported strings representing packed decimals; exported packed decimals to be written as strings; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { precision= p }
field_definition { precision= p };
```

where p is the precision of the source packed decimal on import and the precision of the destination string on export; p has no limit.

Discussion

When a source decimal is exported to a string representation, the export operator uses the precision and scale defined for the source decimal field to determine the length of the destination string. The precision and scale properties override this default ("precision" and "scale"). When they are defined, the export operator truncates or pads the source decimal to fit the size of the destination string. If you include the width property ("width"), the export operator truncates or pads the source decimal to fit the size specified by width.

Restriction

The precision property is ignored on export if you also specify text.

Example

The following example shows a schema used to import a source field with the same precision and scale as the destination decimal:

```
record ( a:decimal[6,2]; )
```

WebSphere DataStage imports the source field to a decimal representation with a 6-digit precision.

The following example shows a schema that overrides the default to import a source field with a 4-digit precision:

```
record ( a:decimal[6,2] {precision = 4}; )
```

prefix

Specifies that each imported/exported field in the data file is prefixed by 1, 2, or 4 bytes containing, as a binary value, either the field's length or the tag value for a tagged subrecord.

Applies to

All fields and record.

Syntax

```
record { prefix = prefix }  
  field_definition { prefix = prefix };
```

where *prefix* is the integer 1, 2, or 4, which denotes a 1-, 2-, or 4-byte prefix containing the field length or a character enclosed in single quotes.

Discussion

You can use this option with variable-length fields. Variable-length fields can be either delimited by a character or preceded by a 1-, 2-, or 4-byte prefix containing the field length.

Import behavior and export behavior differ:

- On import, the import operator reads the length prefix but does not include the prefix as a separate field in the imported data set.
- On export, the export operator inserts the prefix before each field. You can use this option with variable-length fields. Variable-length fields can be either delimited by a character or preceded by a 1-, 2-, or 4-byte prefix containing the field length.

This property is mutually exclusive with `delim`, `delim_string`, `quote`, `final_delim`, and `reference`.

Example

In the following example, fields *a* and *b* are both variable-length string fields preceded by a 2-byte string length:

```
record {prefix = 2} (  
  a:string;  
  b:string;  
  c:int32; )
```

In the following example, the 2-byte prefix is overridden for field *b*, where the prefix is a single byte:

```
record {prefix = 2} (  
  a:string;  
  b:string {prefix = 1}; )
```

For tagged subrecords, the tag field might be either a prefix of the tagged aggregate or another field in the record associated with the tagged aggregate by the link property. Shown below is an example in which the tagged aggregate is preceded by a one-byte unsigned integer containing the tag:

```
record (  
  ...  
  tagField:tagged {prefix=1} (  
    aField:string;  
    bField:int32;  
    cField:sfloat;  
  );  
)
```

print_field

For debugging purposes only; causes the import operator to display each value imported for the field.

Applies to

Fields of all data types and record.

Syntax

```
record { print_field }
field_definition { print_field };
```

Discussion

This property causes import to write out a message for either selected imported fields or all imported fields, in the form:

Importing *N*: *D*

where:

- *N* is the field name.
- *D* is the imported data of the field. Non-printable characters contained in *D* are prefixed with an escape character and written as C string literals; if the field contains binary data, it is output in octal format.

Restrictions

This property is ignored on export.

Example

For example, the following schema uses print_field:

```
record {print_field} (a:string; b:int32;)
```

By default, imported numeric fields represent data as text. In this case, the import operator issues the following message:

```
Importing a: "the string"
Importing b: "4660"
```

The following schema specifies that the numeric data is represented in binary form:

```
record {binary, print_field} (a:string; b:int32;)
```

In this case, the import operator prints the binary data in an octal format as shown below:

```
Importing a: "a string"
Importing b: "\000\000\022\064"
```

quote

Specifies that a field is enclosed in single quotes, double quotes, or another ASCII or multi-byte Unicode character or pair of ASCII or multi-byte Unicode characters.

Applies to

Fields of any data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { quote = ' quotechar ' | ' quotechars ' }
field_definition { quote = ' quotechar ' | ' quotechars ' };
```

where *quotechar* is one of the following: single, double, an ASCII or Unicode character, and *quotechars* is a pair of ASCII or multi-byte Unicode characters, for example, '[]'. Enclose *quotechar* or *quotechars* in single quotation marks.

This property is mutually exclusive with prefix and reference.

Discussion

This property is useful for variable-length fields contained either in quotes or in other characters marking the start and end of a field.

- On import, the leading quote character is ignored and all bytes up to but not including the trailing quote character are imported.
- On export, the export operator inserts the leading quote character, the data, and a trailing quote character.

Quote characters are not counted as part of a field's length.

Example

The following example specifies that the data imported into the variable-length string field b is contained in double quotes:

```
record (
    a:int32;
    b:string {quote = double};
    c:int8;
    d:raw; )
```

record_delim

Specifies a single ASCII or multi-byte Unicode character to delimit a record.

Applies to

record; cannot be a field property.

Syntax

```
record {record_delim [= 'delim_char']}
```

where *delim_char* can be a newline character, a null, or one ASCII or multi-byte Unicode character. If no argument is specified, the default is a newline character.

This property is mutually exclusive with record_delim_string, record_prefix, and record_format.

record_delim_string

Specifies an ASCII or multi-byte Unicode string to delimit a record.

Applies to

record; cannot be a field property.

Syntax

```
record { record_delim_string = 'ASCII_string' | 'multi_byte_Unicode_string' }
```

where *ASCII_string* or *multi_byte_Unicode_string* is the string that delimits the record.

Restrictions

You cannot specify special characters by starting with a backslash escape character. For example, specifying \t, which represents an ASCII tab delimiter character, generates an error.

This property is mutually exclusive with record_delim, record_prefix, and record_format.

record_format

Specifies that data consists of variable-length blocked records or implicit records.

Applies to

record; cannot be a field property.

Syntax

```
record { record_format = { type = type [, format = format ]}}
```

where *type* is either implicit or varying.

If you choose the implicit property, data is imported or exported as a stream with no explicit record boundaries. You might not use the property *delim* = *end* with this format.

On import, the import operator converts the input into records using schema information passed to it. Field boundaries are implied by the record schema passed to the operator. You cannot save rejected records with this record format.

On export, the records are written with no length specifications or delimiters marking record boundaries. The end of the record is inferred when all of the fields defined by the schema have been parsed.

The varying property is allows you to specify one of the following IBM blocked or spanned formats: V, VB, VS, VBS, or VR. Data is imported using that format.

This property is mutually exclusive with *record_length*, *record_delim*, *record_delim_string*, and *record_prefix*.

record_length

Import or export fixed-length records.

Applies to

record; cannot be a field property.

Syntax

```
record { record_length = fixed | nbytes }
```

where:

- *fixed* specifies fixed-length records; the record schema must contain only fixed-length elements so that WebSphere DataStage can calculate the record length.
- *nbytes* explicitly specifies the record length in bytes if the record contains variable-length elements.

On export, the export operator pads the records to the specified length with either zeros or the fill character if one has been specified.

This property is mutually exclusive with *record_format*.

record_prefix

Specifies that a variable-length record is prefixed by a 1-, 2-, or 4-byte length prefix.

Applies to

record; cannot apply to fields.

Syntax

```
record {record_prefix [ = prefix ]}
```

where *prefix* is 1, 2, or 4. If you do not specify a value for *prefix*, the variable defaults to 1.

This property is mutually exclusive with record_delim, record_delim_string, and record_format.

reference

Points to a link field containing the length of an imported/exported field.

Applies to

Variable-length vectors of all data types; cannot apply to record, subrec, or tagged.

Syntax

```
field_definition { reference = link_field};
```

where *link_field* is the name of a field of the same record that holds the length of the field defined by *field_definition*.

Variable-length fields can specify the number of elements they contain by means of a link to another field that contains their length or the tag of a tagged subrecord.

This property is mutually exclusive with prefix, delim_string, quote, and delim.

Example

The following statement specifies that the link field a contains the length of the variable-length string field c:

```
record {delim = none, binary}
       (a:int32 {link}; b:int16; c:string {reference = a});
```

round

Round decimals on import or export.

Applies to

Fields of the decimal data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { round = rounding_type }
field_definition { round = rounding_type }
```

where *rounding_type* can be one of the following:

- ceil: Round the source field toward positive infinity. This mode corresponds to the IEEE 754 Round Up mode.
Examples: 1.4 -> 2, -1.6 -> -1
- floor: Round the source field toward negative infinity. This mode corresponds to the IEEE 754 Round Down mode.

Examples: 1.6 -> 1, -1.4 -> -2

- round_inf: Round the source field toward the nearest representable value, breaking ties by rounding toward positive infinity or negative infinity. This mode corresponds to the COBOL ROUNDED mode.
Examples: 1.4 -> 1, 1.5 -> 2, -1.4 -> -1, -1.5 -> -2

- trunc_zero (default): Truncate the source field toward zero. Discard fractional digits to the right of the right-most fractional digit supported in the destination, regardless of sign. For example, if the destination is an integer, all fractional digits are truncated. If the destination is another decimal with a smaller scale, truncate to the scale size of the destination decimal. This mode corresponds to the COBOL INTEGER-PART function.

Examples: 1.6 -> 1, -1.6 -> -1

Import and export behavior

Import behavior and export behavior differ:

- On import, this property specifies how WebSphere DataStage rounds the source field to fit into the destination decimal when the source field is imported to a decimal.
- On export, this property specifies how to round a source decimal when its precision and scale greater than those of the destination.

scale

Specifies the scale of a packed decimal.

Applies to

Imported strings representing packed decimals; exported packed decimals to be written as strings; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { scale = s }  
field_definition { scale = s };
```

where *s* is the scale.

Discussion

By default, the import operator uses the scale defined for the WebSphere DataStage decimal field to import the source field. You can change this. On import, the scale property specifies the scale of the source packed decimal.

By default, when the export operator exports a source decimal to a string representation, it uses the precision and scale defined for the source decimal field to determine the length of the destination string. You can override the default by means of the precision and scale properties. When you do, the export operator truncates or pads the source decimal to fit the size of the destination string. If you include the width property, the export operator truncates or pads the source decimal to fit the size specified by width. See "width".

Restrictions

The scale property is ignored on export if you also specify text.

The value of scale must be less than the precision and greater than 0. The precision is specified by the precision property. See "precision".

Example

The following example is a schema used to import a source field with the same precision and scale as the destination decimal:

```
record ( a:decimal[6,2]; )
```

WebSphere DataStage imports the source field to a decimal representation with a 2-digit scale.

The following schema overrides this default to import a source field with a 4-digit precision and a 1-digit scale:

```
record ( a:decimal[6,2] {precision = 4, scale = 1}; )
```

separate

Specifies that the imported or exported field contains an unpacked decimal with a separate sign byte.

Applies to

Fields of the decimal data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
field_definition { separate[, option] };
```

where *option* can be one of these:

- leading (default)-the sign is contained in the first byte
- trailing-the sign is contained in the last byte

Discussion

By default, the sign of an unpacked decimal is contained in the first byte of the imported string. The following table defines the legal values for the sign byte for both ASCII and EBCDIC:

Sign	ASCII	EBCDIC
positive	0x2B (ASCII "+")	0x43 (EBCDIC "+")
negative	0x2D (ASCII "-")	0x60 (EBCDIC "-")

Example

For example, the following schema specifies that the WebSphere DataStage decimal field contains a leading sign and has a precision of 5 and a scale of 2:

```
record ( a:decimal[5,2] {separate}; )
```

Import and export results differ:

- On import, field a is imported from a decimal representation six bytes long with the sign in the first byte.
- On export, the field is written to six bytes in the destination: the five contained by the decimal and one byte to contain the sign.

skip

Skip a number of bytes from the end of the previous imported/exported field to the beginning of the field.

Applies to

Fields of all data types. Cannot apply to record, subrec, or tagged .

Syntax

```
field_definition { skip = nbytes };
```

where *nbytes* is the number of bytes to skip after the previous record. The value of *nbytes* can be negative but the absolute record offset computed from this and the previous field position must always be greater than or equal to 0.

On export, any skipped bytes are set to zero by default. The record-level fill property specifies an explicit value for the skipped bytes.

Example

For example, the following statement defines a record schema for the import or export operator:

```
record (a:int32 {position = 4}; b:int16 {skip = 2};)
```

Import and export results differ:

- On import, this schema creates each record from an input data file by importing a 32-bit integer, beginning at byte 4 in each input record, skipping the next 2 bytes of the data file, and importing the next two bytes as a 16-bit integer.
- On export, the export operator fills in the first four bytes with zeros, writes out the 32-bit integer, fills the next two bytes with zeroes, and writes the 16-bit integer.

tagcase

Explicitly specifies the tag value corresponding to a subfield in a tagged. By default the fields are numbered 0 to *N*-1, where *N* is the number of fields.

Applies to

Fields within tagged.

Syntax

```
field_definition { tagcase = n }
```

where *n* is an integer denoting the tag value.

text

Specifies the data representation type of a field as being text rather than binary. Data is formatted as text by default.

Applies to

Fields of all data types except ustring; record. For ustring, the same functionality is available through the charset property.

Syntax

```
record { text }
field_definition { text [, ascii | edcdic]
[,import_ebcdic_as_ascii]
[, export_ebcdic_as_ascii] }
```

This property is mutually exclusive with binary.

Discussion

Data is formatted as text by default, as follows:

- For the date data type, text specifies that the source data on import or the destination data on export, contains a text-based date in the form %yyyy-%mm-%dd or uformat. See "default_date_format" for a description of uformat.
- For the decimal data type: an imported or exported field represents a decimal in a string format with a leading space or '-' followed by decimal digits with an embedded decimal point if the scale is not zero. For import, the source string format is:

[+ | -]ddd[.ddd]

For export, the destination string format is:

[+ | -]ddd.[ddd]

Any precision and scale arguments are ignored on export if you specify text.

- For numeric fields (int8, int16, int32, uint8, uint16, uint32, sfloat, and dfloat): the import and export operators assume by default that numeric fields are represented as text; the import operator converts the text representation to a numeric format by means of C functions to. (See "c_format", "in_format", and "out_format".)
- For the time data type: text specifies that the imported or exported field represents time in the text-based form %hh:%nn:%ss or uformat. See "default_time_format" for a description of uformat.
- For the timestamp data type: text specifies a text-based timestamp in the form %yyyy-%mm-%dd %hh:%nn:%ss or uformat, which is default_date_format and default_time_format concatenated. Refer to "default_date_format" and "default_time_format".

Example

If you specify the following record schema:

```
record {text} (a:decimal[5,2];)
```

import and export results are as follows:

- On import, the source decimal is read from a 7-byte string (five bytes for the precision, one for the sign and one for the decimal point).
- On export, the field is written out as a 7-byte string.

time_format

Specifies the format of an imported or exported field representing a time as a string or ustring.

Applies to

Fields of the time data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
field_definition { time_format = time_format | uformat};
```

uformat is described in "default_time_format".

The possible components of the *time_format* string are given in the following table:

Table 74. Time format tags

Tag	Variable width availability	Description	Value range	Options
%h	import	Hour (24), variable width	0...23	s
%hh		Hour (24), fixed width	0...23	s
%H	import	Hour (12), variable width	1...12	s
%HH		Hour (12), fixed width	01...12	s
%n	import	Minutes, variable width	0...59	s
%nn		Minutes, fixed width	0...59	s
%s	import	Seconds, variable width	0...59	s
%ss		Seconds, fixed width	0...59	s
%s.N	import	Seconds + fraction ($N = 0...6$)	–	s, c, C
%ss.N		Seconds + fraction ($N = 0...6$)	–	s, c, C
%SSS	with v option	Milliseconds	0...999	s, v
%SSSSSS	with v option	Microseconds	0...999999	s, v
%aa	German	am/pm marker, locale specific	am, pm	u, w

By default, the format of the time contained in the string is %hh:%nn:%ss. However, you can specify a format string defining the format of the string field.

You must prefix each component of the format string with the percent symbol. Separate the string's components with any character except the percent sign (%).

Where indicated the tags can represent variable-fields on import, export, or both. Variable-width date elements can omit leading zeroes without causing errors.

The following options can be used in the format string where indicated:

s Specify this option to allow leading spaces in time formats. The s option is specified in the form:
%(tag,s)

Where *tag* is the format string. For example:

%(n,s)

indicates a minute field in which values can contain leading spaces or zeroes and be one or two characters wide. If you specified the following date format property:

%(h,s):\$(n,s):\$(s,s)

Then the following times would all be valid:

20: 6:58

20:06:58

20:6:58

- v Use this option in conjunction with the %SSS or %SSSSSS tags to represent milliseconds or microseconds in variable-width format. So the time property:

`%(SSS,v)`

represents values in the range 0 to 999. (If you omit the v option then the range of values would be 000 to 999.)

- u Use this option to render the am/pm text in uppercase on output.
- w Use this option to render the am/pm text in lowercase on output.
- c Specify this option to use a comma as the decimal separator in the %ss.N tag.
- C Specify this option to use a period as the decimal separator in the %ss.N tag.

The c and C options override the default setting of the locale.

The locale for determining the setting of the am/pm string and the default decimal separator can be controlled through the locale tag. This has the format:

`%(L,'locale')`

Where *locale* specifies the locale to be set using the *language_COUNTRY.variant* naming convention supported by ICU. See *NLS Guide* for a list of locales. The default locale for am/pm string and separators markers is English unless overridden by a %L tag or the APT_IMPEXP_LOCALE environment variable (the tag takes precedence over the environment variable if both are set).

Use the locale tag in conjunction with your time format, for example:

`%L('es')%HH:%nn %aa`

Specifies the Spanish locale.

The format string is subject to the restrictions laid out in the following table. A format string can contain at most one tag from each row. In addition some rows are mutually incompatible, as indicated in the 'incompatible with' column. When some tags are used the format string requires that other tags are present too, as indicated in the 'requires' column.

Table 75. Format tag restrictions

Element	Numeric format tags	Text format tags	Requires	Incompatible with
hour	%hh, %h, %HH, %H	-	-	-
am/pm marker	-	%aa	hour (%HH)	hour (%hh)
minute	%nn, %n	-	-	-
second	%ss, %s	-	-	-
fraction of a second	%ss.N, %s.N, %SSS, %SSSSSS	-	-	-

You can include literal text in your date format. Any Unicode character other than null, backslash, or the percent sign can be used (although it is better to avoid control codes and other non-graphic characters). The following table lists special tags and escape sequences:

Tag	Escape sequence
%%	literal percent sign
\%	literal percent sign
\n	newline
\t	horizontal tab
\\"	single backslash

When using variable-width tags, it is good practice to enclose the time string in quotes. For example, the following schema:

```
f1:int32; f2:time { time_format='%h:%n:%s', quote='double' }; f3:int32;
```

would ensure that the following records are processed correctly:

```
01 "23: 1: 4" 1000
02 " 3:21:22" 1001
```

The quotes are required because the parallel engine assumes that variable-width fields are space delimited and so might interpret a legitimate space in the time string as the end of the time.

Restrictions

You cannot specify time_format with either of the following: midnight_seconds and text. That is, you can specify only one of these options.

Example

For example, you define a format string as %hh:%nn:%ss.3 to specify that the string contains the seconds to three decimal places, that is, to milliseconds. Alternatively you could define this as %hh:%nn:%SSS.

timestamp_format

Specifies the format of an imported or exported field representing a timestamp as a string.

Applies to

Fields of the timestamp data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record ( {timestamp_format = timestamp_format | uformat } )
field_definition { timestamp_format = timestamp_format | uformat };
```

uformat is default_date_format and default_time_format concatenated. The two formats can be in any order and date and time elements can be mixed. The *uformat* formats are described in "default_date_format" and "default_time_format".

The *timestamp_format* is the date format and the time format. Again the two formats can be in any order and their elements can be mixed. The formats are described in "date_format" on page 372 and "time_format" on page 404.

You must prefix each component of the format string with the percent symbol (%). Enclose *timestamp_format* in single quotation marks.

Default

If you do not specify the format of the timestamp it defaults to the string %yyyy-%mm-%dd %hh:%nn:%ss.

vector_prefix

Specifies 1-, 2-, or 4-byte prefix containing the number of elements in the vector.

Applies to

Fields that are variable-length vectors, which are formatted accordingly.

Syntax

```
record { vector_prefix [= n ] }
field_definition { vector_prefix [= n ] };
```

where *n* is the optional byte size of the prefix containing the number of elements in the vector; *n* can be 1 (the default), 2, or 4.

If a vector_prefix is defined for the entire record, you can override the definition for individual vectors.

Discussion

Variable-length vectors must use either a prefix on the vector or a link to another field in order to specify the number of elements in the vector. If the variable-length vector has a prefix, you use the property vector_prefix to indicate the prefix length. By default, the prefix length is assumed to be one byte. Behavior on import differs from that on export:

- On import, the source data file must contain a prefix of each vector containing the element count. The import operator reads the length prefix but does not include the prefix as a separate field in the imported data set.
- On export, the export operator inserts the element count as a prefix of each variable-length vector field.

For multi-byte prefixes, the byte ordering is determined by the setting of the little_endian ("little_endian"), big_endian ("big_endian"), or native_endian ("native_endian") properties.

Examples

The following schema specifies that all variable-length vectors are prefixed by a one-byte element count:

```
record {vector_prefix} (a[]:int32; b[]:int32; )
```

In the following record schema, the vector_prefix of the record (1 byte long by default) is overridden for field b, whose vector_prefix is two bytes long:

```
record {vector_prefix} (a[]:int32; b[]:int32 {vector_prefix = 2} )
```

The schema shown below specifies that the variable-length vector a is prefixed by a one-byte element count, and vector b is prefixed by a two-byte element count:

```
record (a[]:int32 {vector_prefix}; b[]:int32 {vector_prefix = 2});
```

Import and export results differ:

- On import, the source data file must contain a prefix of each vector containing the element count.
- On export, the export operator inserts the element count as a prefix of each vector.

width

Specifies the number of 8-bit bytes of an imported or exported text-format field. Base your width specification on the value of your -impexp_charset option setting. If it's a fixed-width charset, you can calculate the number of bytes exactly. If it's a variable length encoding, base your calculation on the width and frequency of your variable-width characters.

Applies to

Fields of all data types except date, time, timestamp, and raw; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { width = n }
field_definition { width = n };
```

where *n* is the number of bytes in the field; you can specify a maximum width of 255 bytes.

Discussion

This property is useful for numeric fields stored in the source or destination file in a text representation.

If no width is specified and you do not use max_width to specify a maximum width, numeric fields exported as text have the following number of bytes of maximum width:

- 8-bit signed or unsigned integers: 4 bytes
- 16-bit signed or unsigned integers: 6 bytes
- 32-bit signed or unsigned integers: 11 bytes
- 64-bit signed or unsigned integers: 21 bytes
- single-precision float: 14 bytes (sign, digit, decimal point, 7 fraction, "E", sign, 2 exponent)
- double-precision float: 24 bytes (sign, digit, decimal point, 16 fraction, "E", sign, 3 exponent)

Restriction

On export, if you specify the width property with a dfloat field, the width must be at least eight bytes long.

zoned

Specifies that the field contains an unpacked decimal using either ASCII or EBCDIC text.

Applies to

Fields of the decimal data type; record, subrec, or tagged if it contains at least one field of this type.

Syntax

```
record { zoned[, option] }
field_definition { zoned[, option] };
```

where *option* can be either trailing or leading:

- trailing (default) specifies that the sign nibble is in the last byte
- leading specifies that the sign nibble is in the first byte

Discussion

Import and export behavior differ:

- On import, the file is read from a zoned representation of the same length, with zoning as defined by the property.
- On export the field is written to the destination.

The following table defines how the sign is represented in both the ASCII and EBCDIC formats:

Sign	ASCII	EBCDIC
positive	Indicated by representing the sign digit normally.	Upper nibble equal to: 0xA, 0xC, 0xE, 0xF
negative	Indicated by setting the 0x40 bit in the sign digit's byte. This turns "0" through "9" into "p" through "y".	Upper nibble equal to: 0xB, 0xD

Example

For example, the following schema specifies that the WebSphere DataStage decimal field has a precision of 5 and a scale of 2 and that its sign nibble is found in the last byte of the field:

```
record ( a:decimal[5,2]{zoned}; )
```

The precision and scale of the source decimal on import, or the destination decimal on export, defaults to the precision and scale of the WebSphere DataStage decimal field. You can use the precision ("precision") and scale ("scale") properties to override these defaults.

Chapter 9. The partitioning library

The partitioning library is a set of related operators that are concerned with partitioning your data.

The partitioning operators are not separate WebSphere DataStage stages, but rather appear as options on the Advanced tab of stage Input pages.

By default, WebSphere DataStage inserts partition and sort operators in your data flow to meet the partitioning and sorting needs of your job.

Use the partitioners described in this topic when you want to explicitly control the partitioning and sorting behavior of an operator. You can also create a custom partitioner using the C++ API.

The partitioning library contains seven partitioners. They are:

- The entire partitioner. Every instance of an operator on every processing node receives the complete data set as input. It is useful when you want the benefits of parallel execution but every instance of the operator needs access to the entire input data set.
- The hash partitioner. Records are partitioned based on a function of one or more fields (the hash partitioning keys) in each record.
- The modulus partitioner. This partitioner assigns each record of an input data set to a partition of its output data set as determined by the value of a specified key field modulo the number of partitions.
- The random partitioner. Records are randomly distributed across all processing nodes. Like roundrobin, random partitioning can rebalance the partitions of an input data set to guarantee that each processing node receives an approximately equal-sized partition.
- The range partitioner. Divides a data set into approximately equal size partitions based on one or more partitioning keys. It is used with the help of one of the following:
 - The writerangemap operator. This operator takes an input data set produced by sampling and partition sorting a data set and writes it to a file in a form usable by the range partitioner. The range partitioner uses the sampled and sorted data set to determine partition boundaries.
 - The makerangemap utility, which determines the approximate range of a data set by sampling the set.
- The roundrobin partitioner. The first record goes to the first processing node, the second to the second processing node, and so on. When WebSphere DataStage reaches the last processing node in the system, it starts over. This method is useful for resizing partitions of an input data set that are not equal in size.
- The same partitioner. No repartitioning is done. With this partitioning method, records stay on the same processing node. .

The entire partitioner

In entire partitioning, every instance of an operator on every processing node receives the complete data set as input. This partitioning method is useful when you want the benefits of parallel execution but every instance of the operator needs access to the entire input data set. For example, you can use this partitioning method to propagate an entire lookup table to each processing node.

When you use the entire partitioner, the output data set of the partitioner must be either:

1. A virtual data set connected to the input of a parallel operator using the any partitioning method
A virtual data set output by a partitioner overrides the partitioning method of an operator using the any partitioning method.
2. A persistent data set

For example, the following figure shows an operator that uses the any partitioning method.

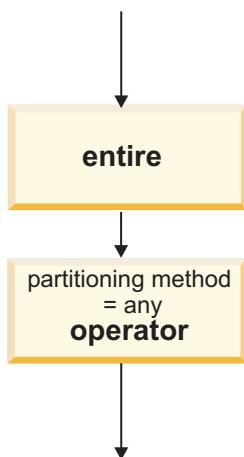


Figure 1. Using the entire Partitioner

To override the any partitioning method of the operator and replace it by the entire partitioning method, you place the entire partitioner into the step as shown.

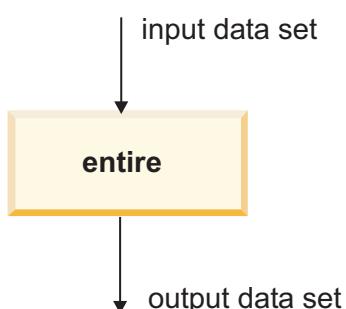
The osh command for this example is:

```
$ osh "... | entire | op ... "
```

Using the partitioner

The entire partitioner takes a single data set as input and repartitions it to create a single output data set, each partition of which contains a complete copy of the input data set.

Data flow diagram



entire: properties

Table 76. entire Partitioner Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema:	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec to outRec without modification
Execution mode	parallel

Table 76. entire Partitioner Properties (continued)

Property	Value
Partitioning method	entire
Preserve-partitioning flag in output set	set
Composite operator	no

Syntax

The entire partitioner has no options. its syntax is simply:

`entire`

The hash partitioner

The hash partitioner examines one or more fields of each input record, called hash key fields. Records with the same values for all hash key fields are assigned to the same processing node. This type of partitioning method is useful when grouping data to perform a processing operation.

When you remove duplicates, you can hash partition records so that records with the same partitioning key values are on the same node. You can then sort the records on each node using the hash key fields as sorting key fields, then remove duplicates, again using the same keys. Although the data is distributed across partitions, the hash partitioner ensures that records with identical keys are in the same partition, allowing duplicates to be found.

The hash partitioner guarantees to assign all records with the same hash keys to the same partition, but it does not control the size of each partition. For example, if you hash partition a data set based on a zip code field, where a large percentage of your records are from one or two zip codes, you can end up with a few partitions containing most of your records. This behavior can lead to bottlenecks because some nodes are required to process more records than other nodes.

For example, the following figure shows the possible results of hash partitioning a data set using the field age as the partitioning key. Each record with a given age is assigned to the same partition, so for example records with age 36, 40, or 22 are assigned to partition 0. The height of each bar represents the number of records in the partition.

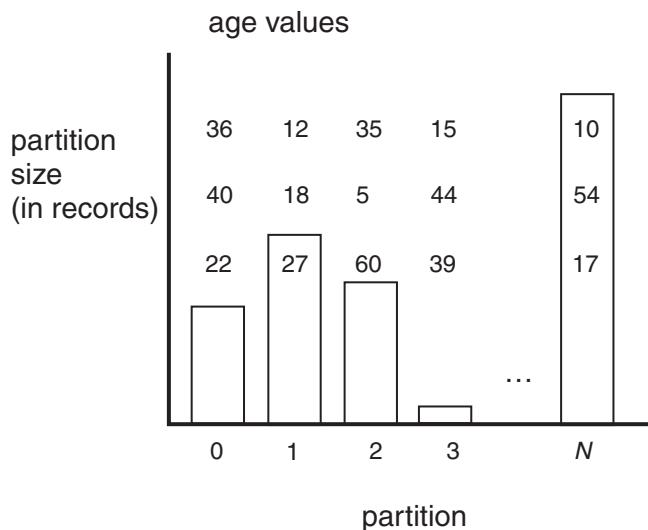


Figure 2. Hash Partitioning Example

As you can see in the diagram, the key values are randomly distributed among the different partitions. The partition sizes resulting from a hash partitioner are dependent on the distribution of records in the data set so even though there are three keys per partition, the number of records per partition varies widely, because the distribution of ages in the population is non-uniform.

When hash partitioning, you should select hashing keys that create a large number of partitions. For example, hashing by the first two digits of a zip code produces a maximum of 100 partitions. This is not a large number for a parallel processing system. Instead, you could hash by five digits of the zip code to create up to 10,000 partitions. You also could combine a zip code hash with an age hash (assuming a maximum age of 190), to yield 1,500,000 possible partitions.

Fields that can only assume two values, such as yes/no, true/false, male/female, are particularly poor choices as hash keys.

Specifying hash keys

Hash keys specify the criteria used to determine the partition into which the hash partitioner assigns a record. The hash partitioner guarantees to assign all records with identical hash keys to the same partition.

The hash partitioner lets you set a primary key and multiple secondary keys. You must define a single primary key, and you have the option of defining as many secondary keys as required by your job. Note, however, that each record field can be used only once as a key. Therefore, the total number of primary and secondary keys must be less than or equal to the total number of fields in the record.

The data type of a partitioning key might be any WebSphere DataStage data type except raw, subrecord, tagged aggregate, or vector.

By default, the hash partitioner uses a case sensitive hash function for strings. You can override this default to perform case insensitive hashing on string fields. In this case, records containing string keys which differ only in case are assigned to the same partition.

Example

This figure shows a step using the hash partitioner:

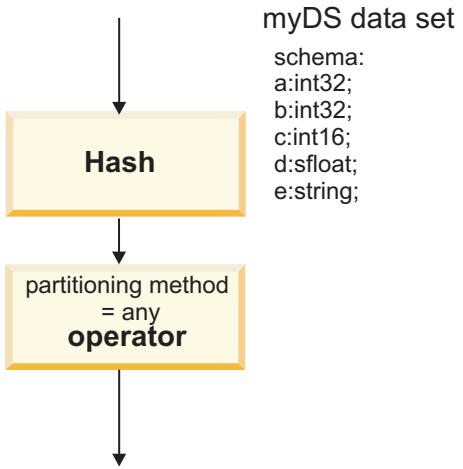


Figure 3. Using the hash Partitioner

In this example, fields a and b are specified as partitioning keys. Shown below is the osh command:

```
$ osh "... | hash -key a -key b | op ..."
```

By default, the hash partitioner uses a case-sensitive hashing algorithm. You can override this by using the -ci option to the partitioner, which is applied to the string field e in the following example:

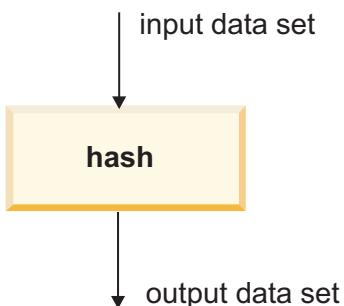
```
$ osh "... | hash -key e -ci | op ..."
```

To prevent the output of the hash partitioner from being repartitioned, the hash partitioner sets the preserve-partitioning flag in its output.

Using the partitioner

The hash partitioner takes a single data set as input and repartitions the input data set to create a single output data set. Each partition of the output data set contains only a subset of the records from the input data set. You must specify at least one key field to the partitioner.

Data flow diagram



hash: properties

Table 77. hash Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema:	inRec:*

Table 77. hash Operator Properties (continued)

Property	Value
Output interface schema	outRec:*
Transfer behavior	inRec to outRec without modification
Execution mode	parallel
Partitioning method	hash
Preserve-partitioning flag in output set	set
Composite operator	no

Hash: syntax and options

Here is the syntax for the hash partitioner in an osh command:

```
hash -key field [-ci | -cs]
[-param params]
[-key field
[-ci | -cs]
[-param params] ...]
[-collation_sequence locale | collation_file_pathname | OFF]
```

There is one required option, -key. You can specify it multiple times.

Table 78. hash Partitioner Option

Option	Use
-key	<p>-key <i>field</i> [-ci -cs] [-param <i>params</i>]</p> <p>Specifies that <i>field</i> is a partitioning key field for the hash partitioner. You can designate multiple key fields where the order is unimportant.</p> <p>The key <i>field</i> must be a field of the data set using the partitioner.</p> <p>The data type of aT partitioning key might be any WebSphere DataStage data type including nullable data types.</p> <p>By default, the hash partitioner uses a case sensitive algorithm for hashing. This means that uppercase strings are distinct from lowercase strings. You can override this default to perform case insensitive hashing, by using the -ci option after the field name.</p> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property</i> = <i>value</i> pairs separated by commas.</p>

Table 78. hash Partitioner Option (continued)

Option	Use
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> Specify a predefined IBM ICU locale Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>

The modulus partitioner

In data mining, data is often arranged in buckets, that is, each record has a tag containing its bucket number. You can use the modulus partitioner to partition the records according to this number. The modulus partitioner assigns each record of an input data set to a partition of its output data set as determined by a specified key field in the input data set. This field can be the tag field.

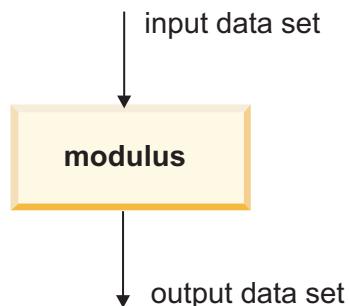
The partition number of each record is calculated as follows:

```
partition_number = fieldname  
mod  
number_of_partitions
```

where:

- *fieldname* is a numeric field of the input data set.
- *number_of_partitions* is the number of processing nodes on which the partitioner executes. If an partitioner is executed on three processing nodes it has three partitions. WebSphere DataStage automatically passes the number of partitions to partitioners, so you need not supply this information.

Data flow diagram



modulus: properties

Table 79. modulus Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema:	inRec:*;
Output interface schema	outRec:*
Transfer behavior	inRec to outRec without modification
Execution mode	parallel
Partitioning method	modulus
Preserve-partitioning flag in output set	set
Composite operator	no

Table 80. modulus Operator Option

Option	Use
key	<p>key <i>key_field</i></p> <p>Specifies the name of the key field on whose value the modulus will be calculated. The key field must be a numeric field, which is converted to a uint32 internally.</p>

Modulus: syntax and options

The syntax for the modulus operation in an osh command is shown below:

```
modulus -key fieldname
```

There is one option. It is required, and you can specify it only once.

Table 81. modulus Partitioner Option

Option	Use
-key	<p>-key <i>fieldname</i></p> <p>Supply the name of key field on whose value the modulus is calculated. The key field must be a numeric field, which is converted to an uint64 internally.</p>

Example

In this example, the modulus partitioner partitions a data set containing ten records. Four processing nodes run the partitioner, and the modulus partitioner divides the data among four partitions.

The input interface schema is as follows:

```
a:uint32; date:date;
```

Field a, of type uint32, is specified as the key field, on which the modulus operation is calculated.

Here is the input data set. Each line represents a record:

```

64123 1960-03-30
61821 1960-06-27
44919 1961-06-18
22677 1960-09-24
90746 1961-09-15
21870 1960-01-01
87702 1960-12-22
4705 1961-12-13
47330 1961-03-21
88193 1962-03-12

```

The following table shows the output data set divided among four partitions by the modulus partitioner.

Partition 0	Partition1	Partition2	Partition3
	61821 1960-06-27 <u>22677</u> 1960-09-24 4705 1961-12-13 4705 1961-12-13	21870 1960-01-01 87702 1960-12-22 <u>47330</u> 1961-03-21 90746 1961	<u>64123</u> 1960-03-30 44919 1961-06-18

Here are three sample modulus operations, corresponding to the values of the three key fields shown above with underscore:

22677 mod 4 = 1; the data is written to Partition 1.
47330 mod 4 = 2; the data is written to Partition 2.
64123 mod 4 = 3; the data is written to Partition 3.

None of the key fields can be divided evenly by 4, so no data is written to Partition 0.

The random partitioner

In random partitioning, records are randomly distributed across all processing nodes. Like round robin, random partitioning can rebalance the partitions of an input data set to guarantee that each processing node receives an approximately equal-sized part of the data. The random partitioning method has a slightly higher overhead than roundrobin because of the extra processing required to calculate a random value for each record.

When you use the random partitioner, the output data set of the partitioner must be either:

1. A virtual data set connected to the input of a parallel operator using the any partitioning method
A virtual data set output by a partitioner overrides the partitioning method of an operator using the any partitioning method.
2. A persistent data set

For example, the diagram shows an operator that uses the any partitioning method.

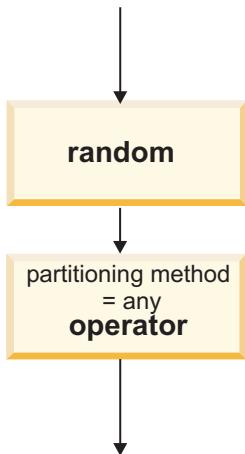


Figure 4. Using the random Partitioner

To override the any partitioning method of op and replace it by the random partitioning method, you place the random partitioner into the step.

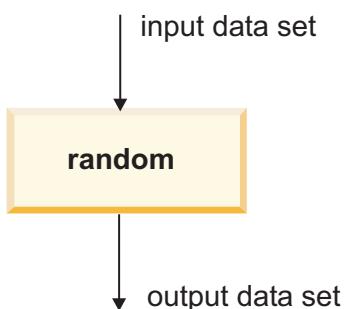
Here is the osh command for this example:

```
$ osh "... | random | op ... "
```

Using the partitioner

The random partitioner takes a single data set as input and repartitions the input data set to create a single output data set, each partition of which contains a random subset of the records in the input data set.

Data flow diagram



random: properties

Table 82. random Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema:	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec to outRec without modification
Execution mode	parallel

Table 82. random Operator Properties (continued)

Property	Value
Partitioning method	random
Preserve-partitioning flag in output set	cleared
Composite operator	no

Syntax

The syntax for the random partitioner in an osh command is:

```
random
```

The random partitioner has no options.

The range Partitioner

A range partitioner divides a data set into approximately equal size partitions based on one or more partitioning keys. Range partitioning is often a preprocessing step to performing a total sort on a data set.

This topic describes the range partitioner, the partitioner that implements range partitioning. It also describes the writerangemap operator, which you use to construct the range map file required for range partitioning, and the stand-alone makerangemap utility.

The range partitioner guarantees that all records with the same partitioning key values are assigned to the same partition and that the partitions are approximately equal in size so all nodes perform an equal amount of work when processing the data set.

The diagram shows an example of the results of a range partition. The partitioning is based on the age key, and the age range for each partition is indicated by the numbers in each bar. The height of the bar shows the size of the partition.

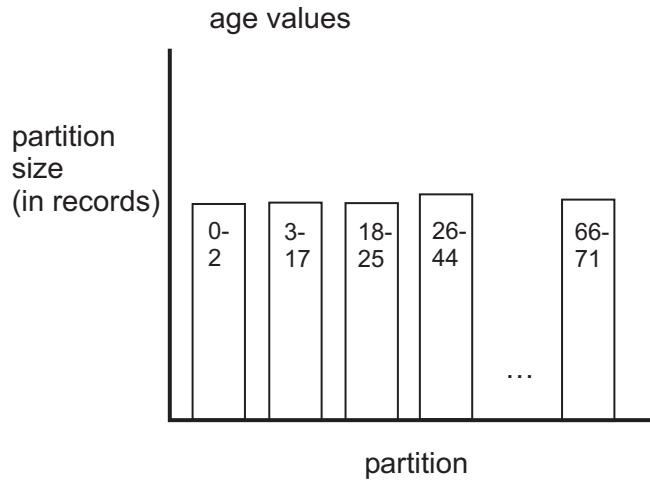


Figure 5. Range Partitioning Example

All partitions are of approximately the same size. In an ideal distribution, every partition would be exactly the same size. However, you typically observe small differences in partition size.

In order to size the partitions, the range partitioner orders the partitioning keys. The range partitioner then calculates partition boundaries based on the partitioning keys in order to evenly distribute records to the partitions. As shown above, the distribution of partitioning keys is often not even; that is, some partitions contain many partitioning keys, and others contain relatively few. However, based on the calculated partition boundaries, the number of records in each partition is approximately the same.

Range partitioning is not the only partitioning method that guarantees equivalent-sized partitions. The random and roundrobin partitioning methods also guarantee that the partitions of a data set are equivalent in size. However, these partitioning methods are keyless; that is, they do not allow you to control how records of a data set are grouped together within a partition.

Considerations when using range partitioning

The range partitioner creates an output data set with approximately equal size partitions where the partitions are ordered by the partitioning keys. This type of partitioning is often useful when you want to perform a total sort of a data set so that the individual records of a partition are sorted and the partitions themselves are sorted. See "The tsort Operator" for more information on performing a total sort.

In order to perform range partitioning your job requires two steps: one step to calculate the range partition boundaries, and a second step to actually use the partitioner. Thus the range partitioner adds processing overhead to your job.

If you only want to perform load balancing so that your partitions are approximately equal in size, you should use the random or roundrobin partitioners. These partitioners add little overhead in comparison to range partitioning. You should use the range partitioner only when you need ordered partitions, not as a general-purpose load-balancing partitioner.

The range partitioning algorithm

The range partitioner uses a probabilistic splitting technique to range partition a data set. This technique is described in *Parallel Sorting on a Shared-Nothing Architecture Using Probabilistic Splitting* by DeWitt, Naughton, and Schneider in *Query Processing in Parallel Relational Database Systems* by Lu, Ooi, and Tan, IEEE Computer Society Press, 1994.

Specifying partitioning keys

The range partitioner uses the partitioning keys to determine the partition boundaries of a data set. You must define at least one partitioning key, and you have the option of defining as many keys as required by your job. However, each record field can be used only once as a key. Therefore, the total number of keys must be less than or equal to the total number of fields in the record.

The data type of a key can be any one of the defined WebSphere DataStage data types except raw, subrecord, tagged aggregate, or vector. Specifying a partitioning key of one of these data types causes the range partitioner to issue an error and abort execution.

By default, the range partitioner does case-sensitive comparison. This means that uppercase strings appear before lowercase strings in a partitioned data set. You can override this default if you want to perform case-insensitive partitioning on string fields.

By default, the range partitioner uses an ascending order, so that smaller values appear before larger values in the partitioned data set. You can specify a descending order as well, so that larger values appear before smaller values in the partitioned data set.

Note: When you use the range partitioner in preparation for a total sort, it is important that the keys for the range partitioning and the sorting be specified the same way, including such attributes as case sensitivity and sort order.

Creating a range map

This section describes the procedure for creating and configuring a range map. Once you have created the range map, you can supply it to the range partitioner to partition a data set. See "Example: Configuring and Using range Partitioner" for the code required to configure and use the range partitioner.

To perform range partitioning, the range partitioner must determine the partition boundaries of a data set as determined by the partitioning keys. For example, if you specify age as the partitioning key, the range partitioner must be able to determine the low and high age values that are the boundaries for each partition in order to generate equally sized partitions. In this case, all records with the same age value between the first set of boundaries is assigned to partition 0; records within the next set of boundaries is assigned to partition 1, and so on.

In order for the range partitioner to determine the partition boundaries, you pass the range partitioner a sorted sample of the data set to be range partitioned. From this sample, the range partitioner can determine the appropriate partition boundaries for the entire data set.

To use a range partitioner:

1. Create a random sample of records from the data set to be partitioned using the sample partitioner. Your sample should contain at least 100 records per processing node in order to accurately determine the partition boundaries.
See "Sample Operator" for more information on the sample partitioner.
2. Use the tsort partitioner, in sequential mode, to perform a complete sort of the sampled records using the partitioning keys as sorting keys. Since your sample size should typically be less than 25,600 (assuming a maximum of 256 nodes in your system), the sequential-mode sort is quick.
You must sort the sampled data set using the same fields as sorting keys, and in the same order, as you specify the partitioning keys to the range partitioner. Also, the sorting keys must have the same characteristics for case sensitivity and ascending or descending ordering as you specified for the range partitioner.
3. Use the writerangemap operator to store the sorted, sampled data set to disk as a file. This file is called a range map. See "The writerangemap Operator" for more information on this operator.
4. Configure the range partitioner using the sorted, sampled data file. The range partitioner determines the partition boundaries for the entire data set based on this sample.
5. Use the -key argument to specify the partitioning keys to the range partitioner. Note that you must specify the same fields as the partitioning keys, and in the same order, as you specified as sorting keys above in Step 2. Also, the partitioning keys must have the same characteristics for case sensitivity and ascending or descending ordering as you specified for the sort.

The diagram shows a data flow where the second step begins with a range partitioner:

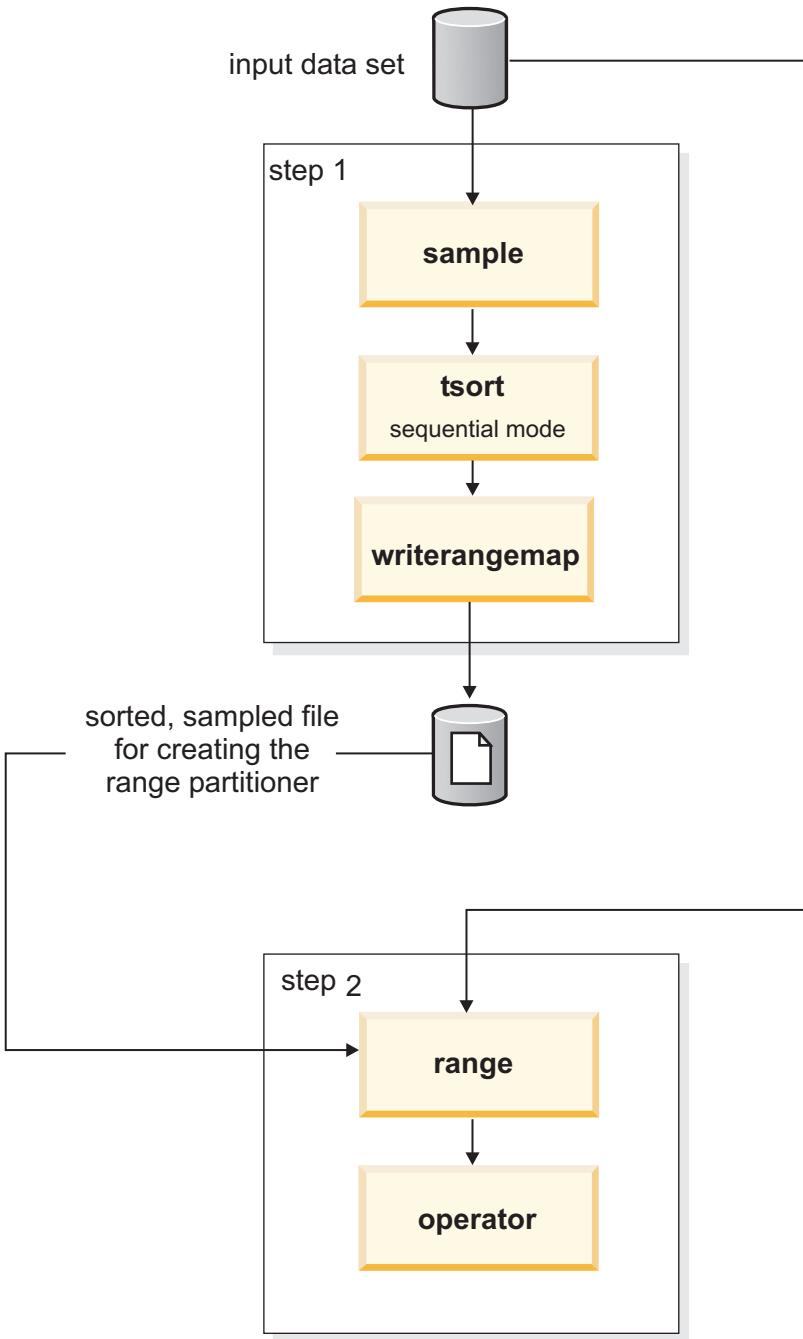


Figure 6. Two-Step Data Flow with Range Partitioner

Note the diagram shows that you sample and sort the data set used to configure the range partitioner in one WebSphere DataStage step, and use the range partitioner in a second step. This is because all the processing of the sorted sample is not complete until the first step ends.

This example shows a range partitioner configured from the same data set that you want to partition. However, you might have multiple data sets whose record distribution can be accurately modeled using a range partitioner configured from a single data set. In this case, you can use the same range partitioner for all the data sets.

Example: configuring and using range partitioner

This section gives an example of configuring a range partitioner. This example contains two steps: one to create the sorted, sampled data set used by the range partitioner and a second step containing an operator that uses the partitioner.

The input data set for this example has the following schema:

```
record (
    a:int32;
    b:int8;
    c:string[5];
    d:int16;
    e:string;
)
```

You decide to create a range partition using fields a and c as the range partitioning keys. Your system contains 16 processing nodes.

Here are the UNIX shell and osh commands for these two steps:

```
$ numSampled=1600                                # Line 1
$ numRecs=`dsrecords inDS.ds | cut -f1 -d' '`      # Line 2
$ percent=`echo "10 k $numSampled $numRecs / 100 * p q" |
dc`                                                 # Line 3

$ osh "sample $percent < inDS.ds |
  tsort -key a -key c [seq] |
  writerangemap
    -rangemap sampledData -overwrite
    -interface 'record(a:int32; c:string[5]);'""
$ osh "range -sample sampledData -key a -key c < inDS.ds | op1
..."
```

The sample size required by the range partitioner is at least 100 records per processing node. Since there are 16 processing nodes, you specify the sample size as 1600 records on Line 1.

On Lines 2 and 3 you calculate the sample size as a percentage of the total number of records in the data set. This calculation is necessary because the sample operator requires the sample size to be expressed as a percentage.

In order to calculate the percentage, you use the dsrecords utility to obtain the number of records in the input data set. The return value of dsrecords has the form "# records" where # is the number of records. Line 2 returns the record count and strips off the word "record" from the value.

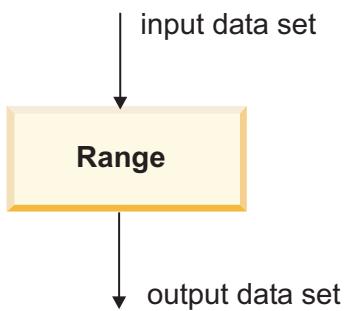
Line 3 then calculates a floating point value for the sample percentage from the 1600 records required by the sample and the number of records in the data set. This example uses the UNIX dc command to calculate the percentage. In this command, the term 10 k specifies that the result has 10 digits to the left of the decimal point. See the man page on dc for more information.

The range partitioner in this example partitions an input data set based on fields a and c. Therefore, the writerangemap operator only writes fields a and c to the output file used to generate the range partitioner.

Using the partitioner

The range partitioner takes a single data set as input and repartitions the input data set to create a single output data set, each partition of which contains a subset of the records from the input data set

Data flow diagram



range: properties

Table 83. range Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema:	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec to outRec without modification
Execution mode	parallel
Partitioning method	range
Preserve-partitioning flag in output set	set
Composite operator	no

Range: syntax and options

The syntax for the range partitioner is shown below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose the value in single quotes. At least one -key option is required.

```
range -key fieldname [-ci | -cs] [-asc | desc] [-nulls first | last] [-ebcdic] [-params params]  
[-key fieldname [-ci | -cs] [-asc | desc] [-nulls first | last] [-ebcdic] [-params params] ...]  
[-collation_sequence locale | collation_file_pathname | OFF]  
[-sample sorted_sampled_data_set]
```

Table 84. range Partitioner Options

Option	Use
-key	<p>-key <i>fieldname</i> [-ci -cs] [-asc desc] [-nulls first last] [-ebcdic [-params <i>params</i>]]</p> <p>Specifies that <i>fieldname</i> is a partitioning key field for the range partitioner. You can designate multiple partitioning key fields. You must specify the same fields, in the same order and with the same characteristics, for ascending/descending and case sensitive/insensitive order, as used to sort the sampled data set.</p> <p>The field name <i>fieldname</i> must be a field of the data set using the partitioner or a field created using an input field adapter on the data set.</p> <p>By default the range partitioner uses a case-sensitive algorithm. You can perform case-insensitive partitioning by using the -ci option after the field name.</p> <p>-ascending specifies ascending order sort; records with smaller values for <i>fieldname</i> are assigned to lower number partitions. This is the default.</p> <p>-descending specifies descending order sort; records with smaller values for <i>fieldname</i> are assigned to lower number partitions.</p> <p>-nulls {first last} specifies whether nulls appear first or last in the sorted partition. The default is first.</p> <p>-ebcdic specifies that the EBCDIC collating sequence is used.</p> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property = value</i> pairs separated by commas.</p>
-collation _ sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> • Specify a predefined IBM ICU <i>locale</i> • Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> • Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.htm</p>
-sample	<p>-sample <i>sorted_sampled_data_set</i></p> <p>Specifies the file containing the sorted, sampled data set used to configure the range partitioner.</p>

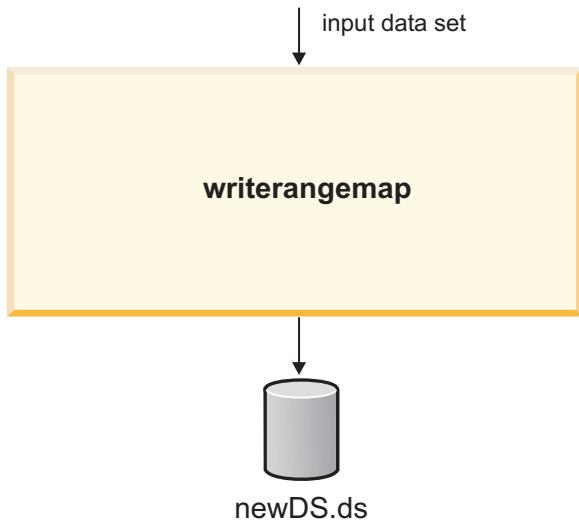
Table 85. range Operator Options

Option	Use
key	<p>key <i>field_name</i> [Case: Sensitive Insensitive] [Ascending Descending]</p> <p>Specifies that <i>field_name</i> is a partitioning key field for the range operator. You can designate multiple partitioning key fields. You must specify the same fields, in the same order, and with the same characteristics for ascending/descending order and case sensitive/insensitive matching, as used to sort the sampled data set.</p> <p>The field name <i>field_name</i> must be a field of the data set using the partitioner or a field created using an input field adapter on the data set.</p> <p>By default, the range operator is case sensitive. This means that uppercase strings come before lowercase strings. You can override this default to perform case-insensitive partitioning by using the Case: Insensitive radio button under the field name.</p> <p>By default, the range partitioning operator uses ascending order, so that records with smaller values for <i>field_name</i> are assigned to lower number partitions than records with larger values. You can specify descending sorting order, so that records with larger values are assigned to lower number partitions, where the options are:</p> <ul style="list-style-type: none"> Ascending specifies ascending order which is the default Descending specifies descending order
sample	<p>-sample <i>sorted_sampled_data_set</i></p> <p>Specifies the file containing the sorted, sampled data set used to create the operator.</p>

Writerangemap operator

The writerangemap operator takes an input data set produced by sampling and partition sorting a data set and writes it to a file in a form usable by the range partitioner. The range partitioner uses the sampled and sorted data set to determine partition boundaries.

Data flow diagram



The operator takes a single data set as input. You specify the input interface schema of the operator using the `-interface` option. Only the fields of the input data set specified by `-interface` are copied to the output file.

writerangemap: properties

Table 86. writerangemap properties

Property	Value
Number of input data sets	1
Number of output data sets	0 (produces a data file as output)
Input interface schema:	specified by the interface arguments
Output interface schema	none
Transfer behavior	inRec to outRec without modification
Execution mode	sequential only
Partitioning method	range
Preserve-partitioning flag in output set	set
Composite operator	no

Writerangemap: syntax and options

The syntax for the writerangemap operator is shown below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose the value in single quotes.

```

writerangemap
  [-key fieldname
   [-key fieldname ...]]
  | [-interface schema]
  -collation_sequence locale
  | collation_file_pathname | OFF
  [-overwrite]
  -rangemap filename

```

Table 87. Writerangemap options

Option	Use
-key	<p>-key <i>fieldname</i></p> <p>Specifies an input field copied to the output file. Only information about the specified field is written to the output file. You only need to specify those fields that you use to range partition a data set.</p> <p>You can specify multiple -key options to define multiple fields.</p> <p>This option is mutually exclusive with -interface. You must specify either -key or -interface, but not both.</p>
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> Specify a predefined IBM ICU locale Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. By default, WebSphere DataStage sorts strings using byte-wise comparisons. <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.htm</p>
-interface	<p>-interface <i>schema</i></p> <p>Specifies the input fields copied to the output file. Only information about the specified fields is written to the output file. You only need to specify those fields that you use to range partition a data set.</p> <p>This option is mutually exclusive with -key; that is, you can specify -key or -interface, but not both.</p>
-overwrite	<p>-overwrite</p> <p>Tells the operator to overwrite the output file, if it exists. By default, the operator does not overwrite the output file. Instead it generates an error and aborts the job if the file already exists.</p>
-rangemap	<p>-rangemap <i>filename</i></p> <p>Specifies the pathname of the output file which will contain the sampled and sorted data.</p>

Using the writerange operator

For an example showing the use of the writerange operator, see "Example: Configuring and Using range Partitioner".

The makerangemap utility

WebSphere DataStage supplies the makerangemap utility to generate the sorted, sampled data file, or range map, used to configure a range partitioner. The makerangemap utility determines the sample size based on the number of processing nodes in your system as defined by the WebSphere DataStage configuration file, or you can explicitly specify the sample size as a number of records or as a percentage of the input data set.

Makerangemap: syntax and options

The syntax for makerangemap is below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose the value in single quotes.

```
makerangemap -rangemap filename [-f]
-key fieldname [ci | cs][-asc | -desc][-ebcdic]
[-key fieldname ...]
-collation_sequence locale | collation_file_pathname | OFF [-percentage percent]
[-size samplesize]
```

where:

- *filename* specifies the name of the file containing the range map, a file containing the sorted, sampled records used to configure a range partitioner.
- *fieldname* specifies the field(s) of the input data used as sorting key fields. You can specify one or more key fields. Note that you must sort the sampled data set using the same fields as sorting keys, and in the same order, as you specify as partitioning keys. Also, the sorting keys must have the same ascending/descending and case sensitive/insensitive properties.

Table 88. makerangemap Utility Options

Option	Use
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none">• Specify a predefined IBM ICU <i>locale</i>• Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i>• Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.htm</p>
-f	<p>-f</p> <p>Specifies to overwrite the output file, if it exists. By default, the utility does not overwrite the output file. Instead, it generates an error and aborts if the file already exists.</p>

Table 88. makerangemap Utility Options (continued)

Option	Use
-key	<p>-key <i>fieldname</i> [-ci -cs][-asc -desc][-ebcdic]</p> <p>Specifies that <i>fieldname</i> is a sorting key field. You can designate multiple sorting key fields. You must specify the same fields, in the same order and with the same characteristics for ascending/descending and case sensitive/insensitive order, as used by the range partitioner.</p> <p>The field name <i>fieldname</i> must be a field of the input data set.</p> <p>By default, the sort uses a case-sensitive algorithm. This means that uppercase strings come before lowercase strings. You can override this default to perform case-insensitive sorting, where:</p> <ul style="list-style-type: none"> -cs specifies case sensitive (default) -ci specifies case insensitive <p>-ebcdic specifies that <i>fieldname</i> be sorted using the EBCDIC collating sequence.</p> <p>By default, the sort uses ascending order, so that records with smaller values for <i>fieldname</i> come before records with larger values. You can specify descending sorting order, so that records with larger values come first, where:</p> <ul style="list-style-type: none"> -asc specifies ascending, the default -desc specifies descending
-percentage or -p	<p>-percentage <i>percent</i></p> <p>Specifies the sample size of the input data set as a percentage.</p> <p>The sample size defaults to 100 records per processing node in the default node pool as defined by the WebSphere DataStage configuration file.</p> <p>If specified, <i>percent</i> should be large enough to create a sample of 100 records for each processing node executing the operator using the range partitioner.</p> <p>The options -size and -percentage are mutually exclusive.</p>
-rangemap or -rm	<p>-rangemap <i>filename</i></p> <p>Specifies the pathname of the output file containing the sampled and sorted data.</p>

Table 88. makerangemap Utility Options (continued)

Option	Use
-size or -s	<p>-size samplesize</p> <p>Specifies the size of the sample taken from the input data set. The size defaults to 100 records per processing node in the default node pool as defined by the WebSphere DataStage configuration file.</p> <p>If specified, <i>samplesize</i> should be set to at least 100 multiplied by the number of processing nodes executing the operator using the range partitioner.</p> <p>The options -size and -percentage are mutually exclusive.</p>

Using the makerangemap utility

As described in "Example: Configuring and Using range Partitioner", you need to write two WebSphere DataStage steps to configure and use a range partitioner. The first step configures the sorted, sampled file, or range map, and the second step includes the range partitioner that uses the range map.

WebSphere DataStage supplies a UNIX command line utility, makerangemap, that you can also use to create a range map. Using this utility eliminates the WebSphere DataStage step used to create the range map from your job.

The makerangemap utility determines the sample size equal to 100 times the number of processing nodes in your system in the default node pool as defined by the WebSphere DataStage configuration file. For example, if your configuration file contains 32 nodes in the default node pool, makerangemap creates a sample 3200 records in size.

You can explicitly specify the sample size as a total number of records for the entire sample or as a percentage of the input data set. This method might be useful when the operator using the range partitioner executes on a subset of processing nodes, not on all nodes in your system. For example, if you execute the operator on only eight processing nodes, your sample size need only be 800 records. In this case, you can explicitly specify the sample size in terms of a record count, or as a percentage of the input data set.

As an example of the use of makerangemap, consider "Example: Configuring and Using range Partitioner". In that example, two osh commands are used to configure and use the range partitioner. Instead, you could use the makerangemap utility and one osh command, as follows:

```
$ makerangemap -rangemap sampledData -key a -key c inDS.ds
$ osh "range -sample sampledData -key a -key c < inDS.ds | op1 ..."
```

The roundrobin partitioner

In round robin partitioning, the first record of an input data set goes to the first processing node, the second to the second processing node, and so on. When you reach the last processing node in the system, start over. This method is useful for resizing the partitions of an input data set that are not equal in size. Round robin partitioning always creates approximately equal-sized partitions.

When you use the roundrobin partitioner, the output data set of the partitioner must be either:

1. A virtual data set connected to the input of a parallel operator using the any partitioning method
 - A virtual data set output by a partitioner overrides the partitioning method of an operator using the any partitioning method.

2. A persistent data set

For example, Figure 7 shows an operator that uses the any partitioning method.

Figure 7. Using the random Partitioner

To override the any partitioning method of op and replace it by the random partitioning method, you place the roundrobin partitioner into the step as shown.

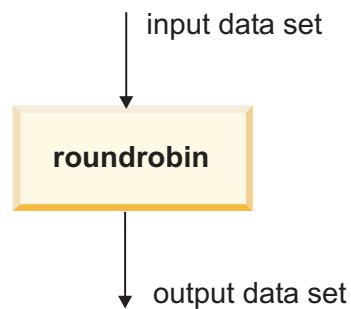
Shown below is the osh command for this example:

```
$ osh "... | roundrobin | op ... "
```

Using the partitioner

The roundrobin partitioner takes a single data set as input and repartitions the input data set to create a single output data set.

Data flow diagram



roundrobin: properties

Table 89. roundrobin Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema:	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec to outRec without modification
Execution mode	parallel
Partitioning method	roundrobin
Preserve-partitioning flag in output set	cleared
Composite operator	no

Syntax

The syntax for the roundrobin partitioner in an osh command is shown below. There are no options.

```
roundrobin
```

The same partitioner

In same partitioning, the operator using the data set as input performs no repartitioning. Records stay in the same partition.

When you use the same partitioner, the output data set of the partitioner must be either:

- A virtual data set connected to the input of a parallel operator using the any partitioning method
 - A virtual data set output by a partitioner overrides the partitioning method of an operator using the any partitioning method.
- A persistent data set

For example, The diagram shows an operator that uses the any partitioning method.

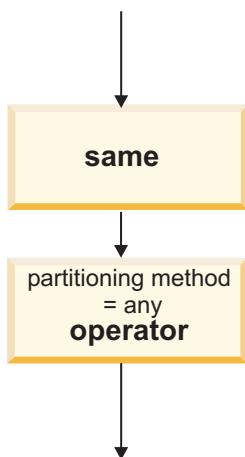


Figure 8. Using the random Partitioner

To override the any partitioning method of op and replace it by the same partitioning method, you place the same partitioner into the step as shown.

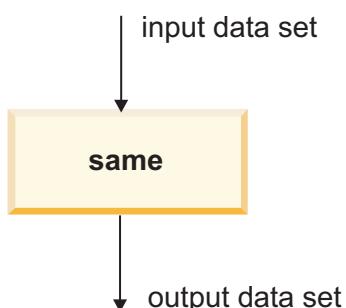
Shown below is the osh command for this example:

```
$ osh "... | same | op ... "
```

Using the partitioner

The same partitioner takes a single data set as input and repartitions the input data set to create a single output data set.

Data flow diagram



same: properties

Table 90. same Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema:	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec to outRec without modification
Execution mode	parallel
Partitioning method	same
Preserve-partitioning flag in output set	propagated
Composite operator	no

Syntax

The syntax for the same partitioner in an osh command is shown below. There are no options.

same

Chapter 10. The collection library

The collection library is a set of related operators that are concerned with collecting partitioned data.

The collection library contains three collectors:

- The ordered collector. Reads all records from the first partition, then all records from the second partition, and so on. This collection method preserves the sorted order of an input data set that has been totally sorted. In a totally sorted data set, the records in each partition of the data set, as well as the partitions themselves, are ordered. .
- The roundrobin collector. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the collector starts over. After reaching the final record in any partition, the collector skips that partition. .
- The sortmerge collector. Reads records in an order based on one or more fields of the record. The fields used to define record order are called collecting keys. .

The ordered collector

WebSphere DataStage allows you to use collectors to explicitly set the collection method of a data set. This section describes how to use the ordered collector.

Ordered collecting

In ordered collection, the collector reads all records from the first input partition, then all records from the second input partition, and so on, to create the output data set. This collection method preserves the record order of each partition of the input data set and might be useful as a preprocessing action before exporting a sorted data set to a single data file.

When you use the ordered collector, the output data set of the collector must be one of these:

- A virtual data set connected to the input of a sequential collector using the any collection method. A virtual data set output by a collector overrides the collection method of a collector using the any collection method.
- A persistent data set. If the data set exists, it must contain only a single partition unless a full overwrite of the data set is being performed.

For example, the diagram shows an operator using the any collection method, preceded by an ordered collector.

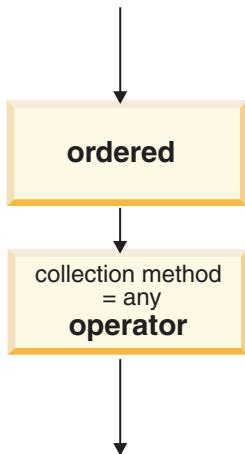


Figure 9. Using the ordered Collector

The any collection method of the other operator is overridden by inserting an ordered collector into the step.

The ordered collector takes a single partitioned data set as input and collects the input data set to create a single sequential output data set with one partition.

ordered Collector: properties

Table 91. ordered Collector Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec to outRec without modification
Execution mode	sequential
Collection method	ordered
Preserve-partitioning flag in output data set	propagated
Composite operator	no

Syntax

The syntax for the ordered collector in an osh command is:

```
$ osh " ... | ordered | ... "
```

The ordered collector has no options.

The roundrobin collector

WebSphere DataStage allows you to use collectors to explicitly set the collection method of a data set. This topic describes how to use the roundrobin collector.

Round robin collecting

In roundrobin collection, the collector reads one record from the first input partition, then one from the second partition, and so on. After reaching the last partition, the collector starts over. After the final record in any partition has been read, that partition is skipped.

When you use the roundrobin collector, the output data set of the collector must be either:

- A virtual data set connected to the input of a sequential operator using the any collection method. A virtual data set output by a collector overrides the collection method of an operator using the any collection method.
- A persistent data set. If the data set exists, it must contain only a single partition unless a full overwrite is being done.

For example, the diagram shows an operator using the any collection method, preceded by a roundrobin collector.

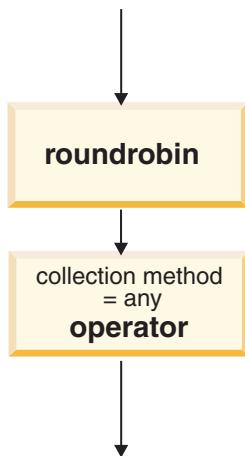


Figure 10. Using the roundrobin Collector

The any collection method of the other operator is overridden by inserting a roundrobin collector into the step.

The roundrobin collector takes a single data set as input and collects the input data set to create a single sequential output data set with one partition.

roundrobin collector: properties

Table 92. roundrobin Collector Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec to outRec without modification
Execution mode	sequential
Collection method	round robin
Preserve-partitioning flag in output data set	propagated

Table 92. roundrobin Collector Properties (continued)

Property	Value
Composite operator	no

Syntax

The syntax for the roundrobin collector in an osh command is:

```
osh " ... | roundrobin | ... "
```

The roundrobin collector has no options.

The sortmerge collector

The sortmerge collector reads records in an order based on one or more fields of the record. The fields used to define record order are called collecting keys. You use the sortmerge collector to implement the sorted merge collection method. This section describes the sortmerge collector.

Understanding the sortmerge collector

The sortmerge collector implements the sorted merge collection method for an input data set. The sortmerge collector determines the order of input records by examining one or more collecting key fields in the current record of each partition of an input data set. Records are collected base on the sort order of these keys.

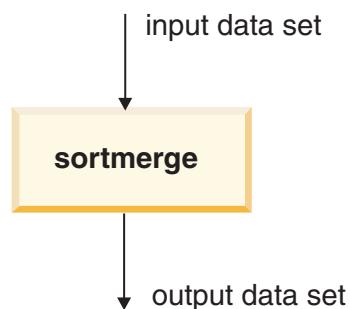
Typically, you use the sortmerge collector with a partition-sorted data set as created by the psort or tsort operator. In this case, you specify as the collecting key fields those fields you specified as sorting key fields to the sorting operator.

When you use the sortmerge collector, the output data set of the collector must be one of the following:

- A virtual data set connected to the input of a sequential operator using the any collection method. A virtual data set output by a collector overrides the collection method of an operator using the any collection method.
- A persistent data set. If the data set exists, it must contain only a single partition unless a full overwrite is being done.

The sortmerge collector takes a partitioned data set as input and collects the input data set to create a single sequential output data set with one partition.

Data flow diagram



Specifying collecting keys

Collecting keys specify the criteria for determining the order of records read by the sortmerge collector. The sortmerge collector allows you to set one primary collecting key and multiple secondary collecting keys. The sortmerge collector first examines the primary collecting key in each input record. For multiple records with the same primary key value, the sortmerge collector then examines any secondary keys to determine the order of records input by the collector.

For example, the diagram shows the current record in each of three partitions of an input data set to the collector:

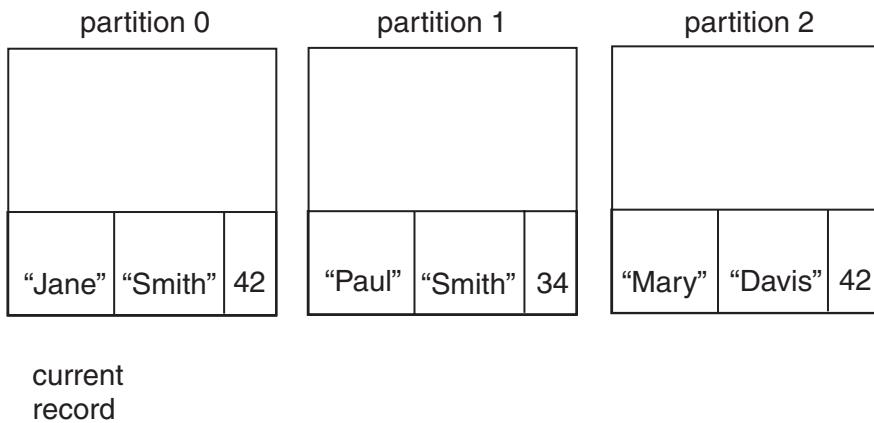


Figure 11. Sample Input to sortmerge Collector

In this example, the records consist of three fields. The first-name and last-name fields are strings, and the age field is an integer.

The following diagram shows the order of the three records read by the sortmerge collector, based on different combinations of collecting keys.

order read:

1	“Jane”	“Smith”	42
2	“Mary”	“Davis”	42
3	“Paul”	“Smith”	34

order read: primary collecting key
↓

1	“Paul”	“Smith”	34
2	“Mary”	“Davis”	42
3	“Jane”	“Smith”	42

↑ secondary collecting key

order read: primary collecting key
↓

1	“Jane”	“Smith”	42
2	“Paul”	“Smith”	34
3	“Mary”	“Davis”	42

↑ secondary collecting key

Figure 12. Sample Outputs from sortmerge Collector

You must define a single primary collecting key for the sortmerge collector, and you might define as many secondary keys as are required by your job. Note, however, that each record field can be used only once as a collecting key. Therefore, the total number of primary and secondary collecting keys must be less than or equal to the total number of fields in the record.

The data type of a collecting key can be any WebSphere DataStage type except raw, subrec, tagged, or vector. Specifying a collecting key of these types causes the sortmerge collector to issue an error and abort execution.

By default, the sortmerge collector uses ascending sort order and case-sensitive comparisons. Ascending order means that records with smaller values for a collecting field are processed before records with larger values. You also can specify descending sorting order, so records with larger values are processed first.

With a case-sensitive algorithm, records with uppercase strings are processed before records with lowercase strings. You can override this default to perform case-insensitive comparisons of string fields.

sortmerge: properties

Table 93. sortmerge Collector Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec:*
Output interface schema	outRec:*
Transfer behavior	inRec to outRec without modification
Execution mode	sequential
Collection method	sortmerge
Preserve-partitioning flag in output data set	propagated
Composite operator	no

Sortmerge: syntax and options

The sortmerge collector in osh uses the following syntax:

```
sortmerge
-key field_name [-ci | -cs] [-asc | -desc]
[-nulls first | last] [-ebcdic] [-param params]
[-key field_name [-ci | -cs] [-asc | -desc]
[-nulls first | last] [-ebcdic] [-param params] ...] [-collation_sequence locale |
collation_file_pathname | OFF]
```

You must specify at least one -key field to the sortmerge collector.

Table 94. sortmerge Collection Options

Option	Use
- key	<p>-key <i>field_name</i> [-ci -cs] [-asc -desc] [-nulls first last] [-ebcdic] [-param <i>params</i>]</p> <p>Specifies that <i>field_name</i> is a collecting key field for the sortmerge collector. You can designate multiple key fields. The first key you list is the primary key.</p> <p><i>field_name</i> must be a field of the data set using the collector or a field created using an input field adapter on the data set.</p> <p>By default, the sortmerge collector does case-sensitive comparisons. This means that records with uppercase strings are processed before records with lowercase strings. You can optionally override this default to perform case-insensitive collecting by using the -ci option after the field name.</p> <p>By default, the sortmerge collector uses ascending order, so that records with smaller values for <i>field_name</i> are processed before records with larger values. You can optionally specify descending sorting order, so that records with larger values are processed first, by using -desc after the field name.</p> <p>By default, the sortmerge collector sorts fields with null keys first. If you wish to have them sorted last, specify -nulls last after the field name.</p> <p>By default, data is represented in the ASCII character set. To represent data in the EBCDIC character set, specify the -ebcdic option.</p> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property = value</i> pairs separated by commas.</p>
- collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <p>Specify a predefined IBM ICU <i>locale</i></p> <p>Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i></p> <p>Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence.</p> <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu /userguide/Collate_Intro.htm</p>

Chapter 11. The restructure library

WebSphere DataStage features ten operators that modify the record schema of the input data set and the level of fields within records.

Table 95. Restructure Operators Listing

Operator	Short Description	See
aggtorec	groups records that have the same key-field values into an output record	The aggtorec Operator
field_export	combines the input fields specified in your output schema into a string- or raw-valued field	The field_export Operator
field_import	exports an input string or raw field to the output fields specified in your import schema	The field_import Operator
makesubrec	combines specified vector fields into a vector of subrecords	The makesubrec Operator
makevect	combines specified fields into a vector of fields of the same type	The makevect Operator
promotesubrec	converts input subrecord fields to output top-level fields	The promotesubrec Operator
splitsubrec	separates input subrecords into sets of output top-level vector fields	The splitsubrec Operator
splitvect	promotes the elements of a fixed-length vector to a set of similarly-named top-level fields	The splitvect Operator
tagbatch	converts tagged fields into output records whose schema supports all the possible fields of the tag cases.	The tagbatch Operator
tagswitch	The contents of tagged aggregates are converted to WebSphere DataStage-compatible records.	The tagswitch Operator

The aggtorec operator

The aggtorec operator takes input records that have been sorted on one or more key fields and groups same-valued key-field records into top-level output records containing subrecords. The number of top-level output records corresponds to the number of unique key-field values in the input data set.

It is important to first sort the data set on the same key fields that you use for the aggtorec operator.

The promotesubrec operator often performs the inverse operation. It is documented in "The promotesubrec Operator".

Output formats

The format of a top-level output record depends on whether you specify the -toplevel option. The following simplified example demonstrates the two output formats. In both cases, the -key option value is field-b, and the -subrecname option value is sub.

This is the input data set:

```
field-a:3 field-b:1  
field-a:4 field-b:2  
field-a:5 field-b:2
```

The two forms of output are:

- When the -toplevel option is not specified, each output top-level record consists entirely of subrecords, where each subrecord has exactly the same fields as its corresponding input record, and all subrecords in a top-level output record have the same values in their key fields.

```
sub:[0:(sub.field-a:3 sub.field-b:1)]  
sub:[0:(sub.field-a:4 sub.field-b:2) 1:(sub.field-a:5 sub.field-b:2)]
```

- When the -toplevel option is specified, the input key field or fields remain top-level fields in the output top-level record, and the non-key fields are placed in a subrecord.

```
field-b:1 sub:[0:(sub.field-a:3)]  
field-b:2 sub:[0:(sub.field-a:4) 1:(sub.field-a:5)]
```

aggrec: properties

Table 96. aggrec Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema:	inRec:*
Output interface schema	See the section "Output Formats".
Transfer behavior	See the section "Output Formats".
Execution mode	parallel (default) or sequential

Aggrec: syntax and options

The syntax of the aggrec operator is as follows. You must specify at least one -key option and the subrecname option.

```
aggrec -key key_field [-ci|-cs] [-param params ]  
[-key key_field [-ci|-cs] [-param params ] ...]  
[-collation_sequence locale | collation_file_pathname | OFF]  
-subrecname subrecname  
-toplevelkeys
```

Table 97. aggtorec Operator Options

-key	<p><code>-key <i>key_field</i> [-ci -cs] [-param <i>params</i>]</code></p> <p>This option is required. Specify one or more fields.</p> <p>If you do not specify <code>-toplevelkeys</code>, all records whose key fields contain identical values are gathered into the same record as subrecords. Each field becomes the element of a subrecord.</p> <p>If you specify the <code>-toplevelkeys</code> option, the key field appears as a top-level field in the output record. All non-key fields belonging to input records with that key field appear as elements of a subrecord in that key field's output record.</p> <p>You can specify multiple keys. For each one, specify the <code>-key</code> option and supply the key's name.</p> <p>By default, WebSphere DataStage interprets the value of key fields in a case-sensitive manner if the values are strings. Specify <code>-ci</code> to override this default. Do so for each key you choose. For example:</p> <pre>-key A -ci -key B -ci</pre> <p>The <code>-param</code> suboption allows you to specify extra parameters for a field. Specify parameters using <i>property =value</i> pairs separated by commas.</p>
-collation_sequence	<p><code>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</code></p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> Specify a predefined IBM ICU <i>locale</i> Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information reference this IBM ICU site:</p> <p>http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>
-subrecname	<p><code>-subrecname <i>subrecname</i></code></p> <p>This option is required. Specify the name of the subrecords that aggtorec creates.</p>
-toplevelkeys	<p><code>-toplevelkeys</code></p> <p>Optionally specify <code>-toplevelkeys</code> to create top-level fields from the field or fields you have chosen as keys.</p>

Aggtorec example 1: the aggtorec operator without the toplevelkeys option

This example shows an input data set and its processing by the aggtorec operator. In the example, the operation runs sequentially, and:

- The key is d, and the input data set has been sorted on d.
- The subrecord name is sub.
- The -toplevelkeys option has not been specified.

The osh command is:

```
$ osh "... aggtorec -key d -subrecname sub ..."
```

The input data set is:

```
a:1 b:00:11:01 c:1960-01-02 d:A
a:3 b:08:45:54 c:1946-09-15 d:A
a:1 b:12:59:01 c:1955-12-22 d:B
a:2 b:07:33:04 c:1950-03-10 d:B
a:2 b:12:00:00 c:1967-02-06 d:B
a:2 b:07:37:04 c:1950-03-10 d:B
a:3 b:07:56:03 c:1977-04-14 d:B
a:3 b:09:58:02 c:1960-05-18 d:B
a:1 b:11:43:02 c:1980-06-03 d:C
a:2 b:01:30:01 c:1985-07-07 d:C
a:2 b:11:30:01 c:1985-07-07 d:C
a:3 b:10:28:02 c:1992-11-23 d:C
a:3 b:12:27:00 c:1929-08-11 d:C
a:3 b:06:33:03 c:1999-10-19 d:C
a:3 b:11:18:22 c:1992-11-23 d:C
```

In the output, records with an identical value in field d have been gathered as subrecords in the same record, as shown here:

```
sub:[0:(sub.a:1 sub.b:00:11:01 sub.c:1960-01-02 sub.d:A)
     1:(sub.a:3 sub.b:08:45:54 sub.c:1946-09-15 sub.d:A)]
sub:[0:(sub.a:1 sub.b:12:59:01 sub.c:1955-12-22 sub.d:B)
     1:(sub.a:2 sub.b:07:33:04 sub.c:1950-03-10 sub.d:B)
     2:(sub.a:2 sub.b:12:00:00 sub.c:1967-02-06 sub.d:B)
     3:(sub.a:2 sub.b:07:37:04 sub.c:1950-03-10 sub.d:B)
     4:(sub.a:3 sub.b:07:56:03 sub.c:1977-04-14 sub.d:B)
     5:(sub.a:3 sub.b:09:58:02 sub.c:1960-05-18 sub.d:B)]
sub:[0:(sub.a:1 sub.b:11:43:02 sub.c:1980-06-03 sub.d:C)
     1:(sub.a:2 sub.b:01:30:01 sub.c:1985-07-07 sub.d:C)
     2:(sub.a:2 sub.b:11:30:01 sub.c:1985-07-07 sub.d:C)
     3:(sub.a:3 sub.b:10:28:02 sub.c:1992-11-23 sub.d:C)
     4:(sub.a:3 sub.b:12:27:00 sub.c:1929-08-11 sub.d:C)
     5:(sub.a:3 sub.b:06:33:03 sub.c:1999-10-19 sub.d:C)
     6:(sub.a:3 sub.b:11:18:22 sub.c:1992-11-23 sub.d:C)]
```

Example 2: the aggtorec operator with multiple key options

This example takes the same input data set as in "Example 1: The aggtorec Operator without the toplevelkeys option". Again, the -toplevelkeys option is not included; however, there are two key fields, a and d, instead of a single field, d.

The operation runs sequentially. The example outputs multiple records with a=1 and a=2 because the data is not sorted on a.

Here is the osh command:

```
$ osh "... aggtorec -key d -key a -subrecname sub ..."
```

The input data set is:

```
a:1 b:00:11:01 c:1960-01-02 d:A
a:3 b:08:45:54 c:1946-09-15 d:A
a:1 b:12:59:01 c:1955-12-22 d:B
a:2 b:07:33:04 c:1950-03-10 d:B
a:2 b:12:00:00 c:1967-02-06 d:B
a:2 b:07:37:04 c:1950-03-10 d:B
a:3 b:07:56:03 c:1977-04-14 d:B
a:3 b:09:58:02 c:1960-05-18 d:B
a:1 b:11:43:02 c:1980-06-03 d:C
a:2 b:01:30:01 c:1985-07-07 d:C
a:2 b:11:30:01 c:1985-07-07 d:C
a:3 b:10:28:02 c:1992-11-23 d:C
a:3 b:12:27:00 c:1929-08-11 d:C
a:3 b:06:33:03 c:1999-10-19 d:C
a:3 b:11:18:22 c:1992-11-23 d:C
```

In the output, records with identical values in both field d and field a have been gathered as subrecords in the same record:

```
sub:[0:(sub.a:1 sub.b:00:11:01 sub.c:1960-01-02 sub.d:A)]
sub:[0:(sub.a:3 sub.b:08:45:54 sub.c:1946-09-15 sub.d:A)]
sub:[0:(sub.a:1 sub.b:12:59:01 sub.c:1955-12-22 sub.d:B)]
sub:[0:(sub.a:2 sub.b:07:33:04 sub.c:1950-03-10 sub.d:B)
    1:(sub.a:2 sub.b:12:00:00 sub.c:1967-02-06 sub.d:B)
    2:(sub.a:2 sub.b:07:37:04 sub.c:1950-03-10 sub.d:B)]
sub:[0:(sub.a:3 sub.b:07:56:03 sub.c:1977-04-14 sub.d:B)
    1:(sub.a:3 sub.b:09:58:02 sub.c:1960-05-18 sub.d:B)]
sub:[0:(sub.a:1 sub.b:11:43:02 sub.c:1980-06-03 sub.d:C)]
sub:[0:(sub.a:2 sub.b:01:30:01 sub.c:1985-07-07 sub.d:C)
    1:(sub.a:2 sub.b:11:30:01 sub.c:1985-07-07 sub.d:C)]
sub:[0:(sub.a:3 sub.b:10:28:02 sub.c:1992-11-23 sub.d:C)
    1:(sub.a:3 sub.b:12:27:00 sub.c:1929-08-11 sub.d:C)
    2:(sub.a:3 sub.b:06:33:03 sub.c:1999-10-19 sub.d:C)
    3:(sub.a:3 sub.b:11:18:22 sub.c:1992-11-23 sub.d:C)]
```

Example 3: The aggtorec operator with the toplevelkeys option

This example shows the same input record as in the previous two examples. The specifications for this example are:

- The -toplevelkeys option is specified.
- The keys are a and d.
- The input schema is as follows:
a:uint8; b:time; c:date; d:string[1];
- The subrecord name is sub.

Here is the osh command:

```
$ osh "... aggtorec -key d -key a -subrecname sub -toplevelkeys ..."
```

In the output:

- Fields a and d are written as the top-level fields of a record only once when they have the same value, although they can occur multiple times in the input data set.
- Fields b and c are written as subrecords of the same record in which the key fields contain identical values.

Here is a small example of input records:

```
a:2 b:01:30:01 c:1985-07-07 d:C
a:2 b:11:30:01 c:1985-07-07 d:C
```

and the resulting output record:

```
a:2 d:C sub:[0:(sub.b:01:30:01 sub.c:1985-07-07)
           1:(sub.b:11:30:01 sub.c:1985-07-07)]
```

Here is the entire input data set:

```
a:1 b:00:11:01 c:1960-01-02 d:A
a:3 b:08:45:54 c:1946-09-15 d:A
a:1 b:12:59:01 c:1955-12-22 d:B
a:2 b:07:33:04 c:1950-03-10 d:B
a:2 b:12:00:00 c:1967-02-06 d:B
a:2 b:07:37:04 c:1950-03-10 d:B
a:3 b:07:56:03 c:1977-04-14 d:B
a:3 b:09:58:02 c:1960-05-18 d:B
a:1 b:11:43:02 c:1980-06-03 d:C
a:2 b:01:30:01 c:1985-07-07 d:C
a:2 b:11:30:01 c:1985-07-07 d:C
a:3 b:10:28:02 c:1992-11-23 d:C
a:3 b:12:27:00 c:1929-08-11 d:C
a:3 b:06:33:03 c:1999-10-19 d:C
a:3 b:11:18:22 c:1992-11-23 d:C
```

Here is the output data set:

```
a:1 d:A sub:[0:(sub.b:00:11:01 sub.c:1960-01-02)]
a:3 d:A sub:[0:(sub.b:08:45:54 sub.c:1946-09-15)]
a:1 d:B sub:[0:(sub.b:12:59:01 sub.c:1955-12-22)]
a:2 d:B sub:[0:(sub.b:07:33:04 sub.c:1950-03-10)
           1:(sub.b:12:00:00 sub.c:1967-02-06)
           2:(sub.b:07:37:04 sub.c:1950-03-10)]
a:3 d:B sub:[0:(sub.b:07:56:03 sub.c:1977-04-14)
           1:(sub.b:09:58:02 sub.c:1960-05-18)]
a:1 d:C sub:[0:(sub.b:11:43:02 sub.c:1980-06-03)]
a:2 d:C sub:[0:(sub.b:01:30:01 sub.c:1985-07-07)
           1:(sub.b:11:30:01 sub.c:1985-07-07)]
a:3 d:C sub:[0:(sub.b:10:28:02 sub.c:1992-11-23)
           1:(sub.b:12:27:00 sub.c:1929-08-11)
           2:(sub.b:06:33:03 sub.c:1999-10-19)
           3:(sub.b:11:18:22 sub.c:1992-11-23)]
```

The field_export operator

The field_export operator exports an input field or fields to a string, ustring, or raw valued output field, and transfers the remaining input fields to output unchanged. The fields incorporated in the string or raw output field are dropped.

You supply an export schema to guide the creation of the string, ustring, or raw field using the -schema or -schemafile option.

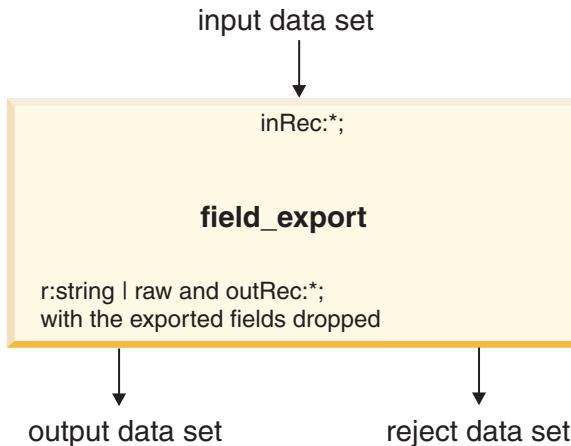
The export schema syntax is:

```
record{text,delim=' delimiter ',final_delim=' delimiter '} (field1:type;field2:type;...fieldn:type;)
```

The specified fields are combined into a single raw output field. You use the -field option to specify the name of the output field, and specify -type string to export to a string field.

You can optionally save rejected records in a separate reject data set. The default behavior is to continue processing and report a count of failures.

Data flow diagram



field_export: properties

Table 98. *field_export Operator Properties*

Property	Value
Number of input data sets	1
Number of output data sets	1, and optionally a reject data set
Input interface schema	inRec:*
Output interface schema	r:string OR ustring OR raw; outRec:/* without the exported fields
Transfer behavior	inRec -> outRec (exported fields are dropped)

Field_export: syntax and options

The `-field` option is required. You must also specify either the `-schema` or `-schemafile` option. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose the value in single quotes.

```
field_export
  -field field_name
  -schema schema_name | -schemafile file_name
  [-saveRejects]
  [-type string | ustring | raw]
```

Table 99. *field_export Operator Options*

Option	Use
<code>-field</code>	<p><code>-field field_name</code></p> <p>Specifies the name of the string- or raw-type field to which the input field or fields are exported.</p>
<code>-saveRejects</code>	<p><code>-saveRejects</code></p> <p>Specifies that the operator continues when it encounters a reject record and writes the record to an output reject data set. The default action of the operator is to continue and report a count of the failures to the message stream. If you specify this suboption, you must attach a second output data set to the operator.</p>

Table 99. *field_export* Operator Options (continued)

Option	Use
-schema	<p>-schema <i>schema_name</i></p> <p>Specifies the name of the export schema. The schema specifies how input fields are packed into the exported string- or raw-type field.</p> <p>You must specify either -schema or -schemafile.</p>
-schemafile	<p>-schemafile <i>file_name</i></p> <p>Specifies the name of a file containing the export schema. The schema specifies how input fields are packed into the exported string- or raw-type field.</p> <p>You must specify either -schemafile or -schema.</p>
-type	<p>-type string ustring raw</p> <p>Specifies the data type of the output field; this is the type to which the operator converts input data. The operator converts the input data to raw-type data by default.</p>

Example

The *field_export* operator exports an input data set whose record schema is:

```
record(value:decimal[5,2];SN:int16;time:time)
```

to an output data set whose record schema is:

```
record(exported:string;time:time;)
```

The operator exports the first two input fields to a single string output field in which the two items are displayed as text separated by a comma. Here is the export schema that guides this operation:

```
record{text,delim=',',final_delim=none}
      (SN:int16;value:decimal[5,2];)
```

The osh commands for this example are:

```
$ expsch="record{text,delim=',',final_delim=none}
      (SN:int16;value:decimal[5,2];) "
$ osh "... field_export -field exported -type string
      -schema $expsch ... "
```

Here is the input data set:

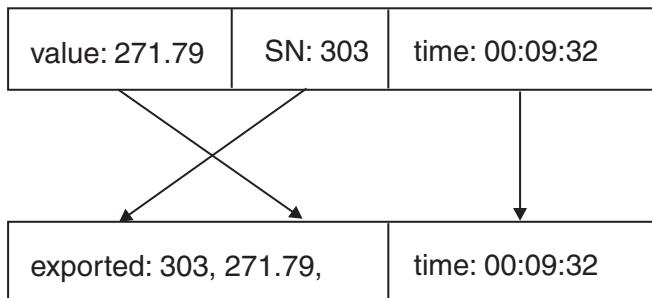
value:657.65	SN:153	time:00:08:36
value:652.16	SN:13	time:00:08:20
value:306.65	SN:391	time:00:08:52
value:292.65	SN:299	time:00:09:24
value:365.65	SN:493	time:00:08:28
value:449.46	SN:580	time:00:09:00
value:271.79	SN:303	time:00:09:32
value:098.15	SN:216	time:00:09:08
value:404.54	SN:678	time:00:08:44

value:379.31	SN:103	time:00:09:16
--------------	--------	---------------

Here is the output data set:

exported:153, 657.65	time:00:08:36
exported:13, 652.16	time:00:08:20
exported:391, 306.65	time:00:08:52
exported:299, 292.65	time:00:09:24
exported:493, 365.65	time:00:08:28
exported:580, 449.46	time:00:09:00
exported:303, 271.79	time:00:09:32
exported:216, 098.15	time:00:09:08
exported:678, 404.54	time:00:08:44
exported:103, 379.31	time:00:09:16

Note that the operator has reversed the order of the value and SN fields and combined their contents. The time field is transferred to the output with no change, as in the following diagram:

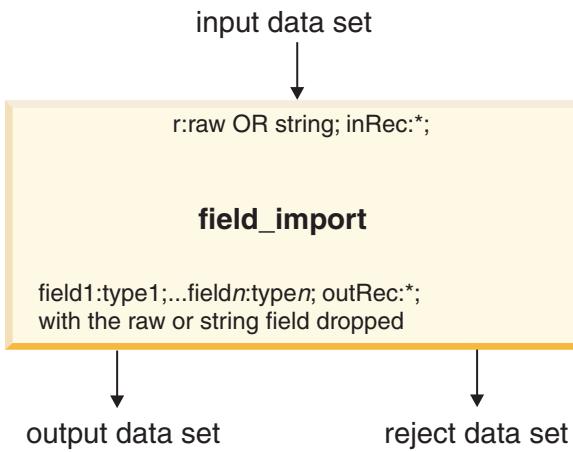


The `field_import` operator

The `field_import` operator exports an input string or raw field to the output fields specified in your import schema. The string or raw input field is dropped, and the other input fields are transferred to output without change.

You can optionally save rejected records in a separate reject data set. The default behavior is to continue processing and report a count of failures.

Data flow diagram



field_import: properties

Table 100. field_import Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1, and optionally a reject data set
Input interface schema	r:raw OR string;inRec:*
Output interface schema	field1:type1;...fieldn:typen;outRec:* with the string or raw-type field dropped.
Transfer behavior	inRec -> outRec (the string- or raw-type field is dropped)

Field_import: syntax and options

The -field option is required. You must also specify either the -schema or -schemafile.

```
field_import -field field_name-schema schema_name | -schemafile file_name
[-keepField]
[-failRejects] | [-saveRejects]
```

Table 101. field_import Operator Properties

Option	Use
-field	-field field_name Specifies the name of the field containing the string or raw data to import.
-keepField	-keepField Specifies that the operator continues when it encounters a reject record and writes the record to the output data set.
-failRejects	-failRejects Specifies that the operator fails when it encounters a record whose import is rejected. The default action of the operator is to continue and report a count of the failures to the message stream. This option is mutually exclusive with -saveRejects.

Table 101. *field_import* Operator Properties (continued)

Option	Use
-saveRejects	-saveRejects Specifies that the operator continues when it encounters a reject record and writes the record to an output reject data set. The default action of the operator is to continue and report a count of the failures to the message stream. This option is mutually exclusive with -failRejects.
-schema	-schema <i>schema_name</i> Specifies the name of schema that interprets the string or raw field's contents by converting them to another data type. You must specify either -schema or -schemafile.
-schemafile	-schemafile <i>file_name</i> Specifies the name of a file containing the schema that interprets the string or raw field's contents by converting them to another data type. You must specify either -schema or -schemafile.

Example

In this example, the *field_import* operator imports a 16-byte raw field to four 32-bit integer fields.

Here is the schema of the input data set:

```
record(rawfield[16]:raw;x:decimal[6,1];)
```

Here is the import schema that guides the import of the 16-byte raw field into four integer output fields. The schema also assures that the contents of the raw[16] field are interpreted as binary integers:

```
record {binary,delim=none}
  (a:int32;
   b:int32;
   c:int32;
   d:int32;
  )
```

Here is the schema of the output data set:

```
record (a:int32;
       b:int32;
       c:int32;
       d:int32;
       x:decimal[6,1];
      )
```

Here are the osh commands for this example:

```
$ intSch="record { binary, delim=none }
  (a:int32;
   b:int32;
   c:int32;
   d:int32;
  )"
$ osh " field_import -field raw -schema $intSch ... "
```

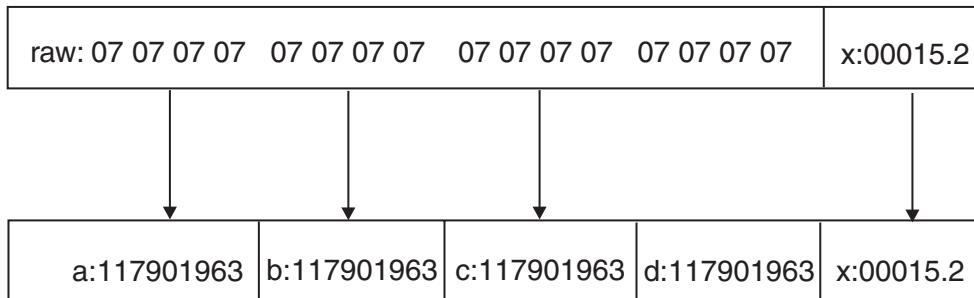
Here is the input data set:

rawfield:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	x:00087.2
rawfield:04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04	x:00004.8
rawfield:08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08	x:00042.7
rawfield:01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01	x:00091.3
rawfield:05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05	x:00075.9
rawfield:09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09	x:00081.3
rawfield:02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02	x:00040.6
rawfield:06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06	x:00051.5
rawfield:03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03	x:00061.7
rawfield:07 07 07 07 07 07 07 07 07 07 07 07 07 07 07 07 07 07 07 07	x:00015.2

Here is the output data set:

a:0	b:0	c:0	d:0	x:00087.2
a:67372036	b:67372036	c:67372036	d:67372036	x:00004.8
a:134744072	b:134744072	c:134744072	d:134744072	x:00042.7
a:16843009	b:16843009	c:16843009	d:16843009	x:00091.3
a:84215045	b:84215045	c:84215045	d:84215045	x:00075.9
a:151587081	b:151587081	c:151587081	d:151587081	x:00081.3
a:33686018	b:33686018	c:33686018	d:33686018	x:00040.6
a:101058054	b:101058054	c:101058054	d:101058054	x:00051.5
a:50529027	b:50529027	c:50529027	d:50529027	x:00061.7
a:117901063	b:117901063	c:117901063	d:117901063	x:00015.2

Note that the operator has imported four bytes of binary data in the input raw field as one decimal value in the output. The input field x (decimal[6,1]) is transferred to the output with no change, as in the following diagram:



In the example above each byte contributes the value shown for a total of 117901963:

byte	$07 * 224 +$	117440512
byte	$07 * 216 +$	458752
byte	$07 * 28 +$	1792
byte	$07 * 20 +$	7
Total		117901963

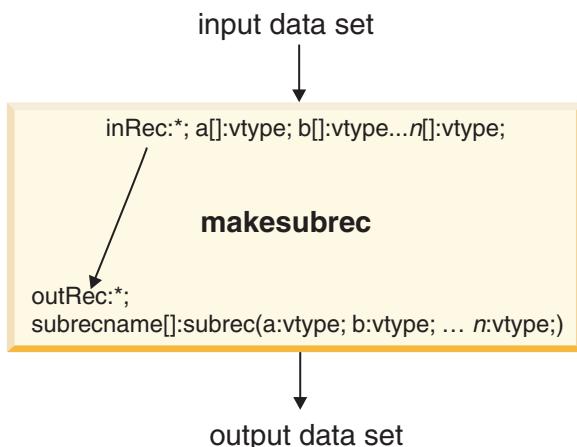
The makesubrec operator

The makesubrec operator combines specified vectors in an input data set into a vector of subrecords whose fields have the names and data types of the original vectors. You specify the vector fields to be made into a vector of subrecords and the name of the new subrecord.

The splitsubrec operator performs the inverse operation. See "The splitsubrec Operator" .

Data flow diagram

The following figure is a data flow diagram of the makesubrec operator, where *vtype* denotes any valid WebSphere DataStage data type that can figure in a vector, that is, any data type except tagged and * (schema variable). Different subrecord elements can be of different vtypes.



makesubrec: properties

Table 102. makesubrec Operator Options

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec: *; a[]:vtype; b[]:vtype; ...n[]:vtype;
Output interface schema	outRec: *; subrecname[]:subrec(a:vtype; b:vtype; ...n:vtype;)
Transfer behavior	See "Transfer Behavior".

Transfer behavior

The input interface schema is as follows:

```
record ( inRec: *; a[]:vtype; b[]:vtype; ...n[]:vtype; )
```

The inRec schema variable captures all the input record's fields except those that are combined into subrecords. In the interface, each field that is combined in a subrecord is a vector of indeterminate type.

The output interface schema is as follows:

```
record ( outRec: *;  
        subrecname[]:subrec(a:vtype; b:vtype; ...n:vtype; ) )
```

Subrecord length

The length of the subrecord vector created by this operator equals the length of the longest vector field from which it is created. If a variable-length vector field was used in subrecord creation, the subrecord vector is also of variable length.

Vectors that are smaller than the largest combined vector are padded with default values: NULL for nullable fields and the corresponding type-dependent value for non-nullable fields. For example, in the following record, the vector field a has five elements and the vector field b has four elements.

```
a:[0:0 1:2 2:4 3:6 4:8]  
b:[0:1960-08-01 1:1960-11-23 2:1962-06-25 3:1961-08-18]
```

The makesubrec operator combines the fields into a subrecord named *sub* as follows:

```
sub:[0:(sub.a:0 sub.b:1960-08-01) 1:(sub.a:2 sub.b:1960-11-23)  
     2:(sub.a:4 sub.b:1962-06-25) 3:(sub.a:6 sub.b:1961-08-18)  
     4:(sub.a:8 sub.b:1-0001)]
```

Subfield b of the subrecord's fifth element, shown in boldface type, contains the data type's default value.

When the makesubrec operator encounters mismatched vector lengths, it displays the following message:

When checking operator: Not all vectors are of the same length.

field_nameA and field_nameB differ in length.

Some fields will contain their default values or Null, if nullable

where *field_nameA* and *field_nameB* are the names of vector fields to be combined into a subrecord. You can suppress this message by means of the -variable option.

This operator treats scalar components as if they were a vector of length 1. This allows scalar and vector components to be combined.

Makesubrec: syntax and options

The -name and -subrecname options are required.

The syntax for the makesubrec operator is as follows:

```
makesubrec
  -name field_name [-name field_name ...]
  -subrecname subrecname  [-variable]
```

Table 103. makesubrec Operator Options

Option	Use
-name	<p><i>-name field_name</i></p> <p>Specify the name of the field to include in the subrecord. You can specify multiple fields to be combined into a subrecord. For each field, specify the option followed by the name of the field to include.</p>
-subrecname	<p><i>-subrecname subrecname</i></p> <p>Specify the name of the subrecord into which you want to combine the fields specified by the -name option.</p>
-variable	<p><i>-variable</i></p> <p>When the operator combines vectors of unequal length, it pads fields and displays a message to this effect. Optionally specify this option to disable display of the message. See "Subrecord Length".</p>

Here are the input and output data set record schemas:

```
a[5]:uint8; b[4]:decimal[2,1]; c[4]:string[1]
b[4]:decimal[2,1]; sub [5](a:uint8; c:string[1];)
```

Here is the osh command:

```
$ osh "... makesubrec -name a -name c -subrecname sub ..."
```

Here is the input data set:

```
a:[0:10 1:0 2:2 3:4 4:6] b:[0:-2.4 1:-0.7 2:0.0 3:0.0]
c:[0:A 1:B 2:C 3:D]
a:[0:6 1:8 2:10 3:0 4:2] b:[0:0.0 1:0.0 2:0.0 3:-2.0]
c:[0:A 1:B 2:C 3:D]
a:[0:0 1:2 2:4 3:6 4:8] b:[0:0.0 1:0.0 2:0.1 3:-3.0]
c:[0:A 1:B 2:C 3:D]
a:[0:4 1:6 2:8 3:10 4:0] b:[0:-3.5 1:0.0 2:4.9 3:0.0]
c:[0:A 1:B 2:C 3:D]
a:[0:8 1:10 2:0 3:2 4:4] b:[0:0.0 1:6.1 2:0.0 3:0.0]
c:[0:A 1:B 2:C 3:D]
```

Here is the output data set:

```
b:[0:0.0 1:0.0 2:0.0 3:-2.0] sub:[0:(sub.a:6 sub.c:A)
                                         1:(sub.a:8 sub.c:B)
                                         2:(sub.a:10 sub.c:C)
                                         3:(sub.a:0 sub.c:D)
                                         4:(sub.a:2 sub.c:\0)]
b:[0:0.0 1:6.1 2:0.0 3:0.0] sub:[0:(sub.a:8 sub.c:A)
                                         1:(sub.a:10 sub.c:B)
                                         2:(sub.a:0 sub.c:C)
                                         3:(sub.a:2 sub.c:D)
                                         4:(sub.a:4 sub.c:\0)]
```

```

b:[0:-2.4 1:-0.7 2:0.0 3:0.0] sub:[0:(sub.a:10 sub.c:A)
                                         1:(sub.a:0 sub.c:B)
                                         2:(sub.a:2 sub.c:C)
                                         3:(sub.a:4 sub.c:D)
                                         4:(sub.a:6 sub.c:\0)]
b:[0:0.0 1:0.0 2:0.1 3:-3.0] sub:[0:(sub.a:0 sub.c:A)
                                         1:(sub.a:2 sub.c:B)
                                         2:(sub.a:4 sub.c:C)
                                         3:(sub.a:6 sub.c:D)
                                         4:(sub.a:8 sub.c:\0)]
b:[0:-3.5 1:0.0 2:4.9 3:0.0] sub:[0:(sub.a:4 sub.c:A)
                                         1:(sub.a:6 sub.c:B)
                                         2:(sub.a:8 sub.c:C)
                                         3:(sub.a:10 sub.c:D)
                                         4:(sub.a:0 sub.c:\0)]

```

Note that vector field c was shorter than vector field a and so was padded with the data type's default value when the operator combined it in a subrecord. The default value is shown in boldface type.

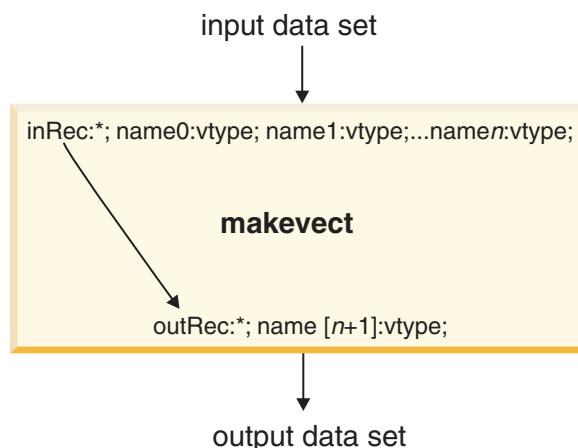
The makevect operator

The makevect operator combines specified fields of an input data record into a vector of fields of the same type. The input fields must be consecutive and numbered in ascending order. The numbers must increase by one. The fields must be named *field_name*₀ to *field_name*_n, where *field_name* starts the name of a field and 0 and n are the first and last of its consecutive numbers. All these fields are combined into a vector of the same length as the number of fields (n+1). The vector is called *field_name*.

The splitvect operator performs the inverse operation. See "The splitvect Operator".

Data flow diagram

The following figure is a conceptual diagram of the makevect operator, where *vtype* denotes any valid WebSphere DataStage data type that can figure in a vector, that is, any data type except tagged and * (schema variable).



makevect: properties

Table 104. makevect Operator Options

Property	Value
Number of input data sets	1

Table 104. *makevect* Operator Options (continued)

Property	Value
Number of output data sets	1
Input interface schema	inRec: *; name0: vtype; name1: vtype; ... namen: vtype;
Output interface schema	outRec: *; name[n+1]:vtype;
Transfer behavior	See "Transfer Behavior".
Execution mode	parallel (default) or sequential

Transfer Behavior

The input interface schema is:

```
record ( inRec: *; name0:vtype; name1:vtype; ... namen:vtype; )
```

The *inRec* schema variable captures all the input record's fields except those that are combined into the vector. The data type of the input fields determines that of the output vector. All fields to be combined in a vector must have compatible data types so that WebSphere DataStage type.

Data type casting is based on the following rules:

- Any integer, signed or unsigned, when compared to a floating-point type, is converted to floating-point.
- Comparisons within a general type convert the smaller to the larger size (sfloat to dfloat, uint8 to uint16, and so on)
- When signed and unsigned integers are compared, unsigned are converted to signed.
- Decimal, raw, string, time, date, and timestamp do not figure in type conversions. When any of these is compared to another type, *makevect* returns an error and terminates.

The output data set has the schema:

```
record ( outRec: *; name[n]:vtype; )
```

The *outRec* schema variable does not capture the entire input record. It does not include the fields that have been combined into the vector.

Non-consecutive fields

If a field between *field_name0* and *field_namen* is missing from the input, the operator stops writing fields to a vector at the missing number and writes the remaining fields as top-level fields. For example, data with this input schema:

```
record ( a0:uint8; a1:uint8; a2:uint8; a4:uint8; a5:uint8; )
```

is combined as follows:

```
record ( a4:uint8; a5:uint8; a[3]:uint8; )
```

The operator combines fields a0, a1, and a2 in a vector and writes fields a4 and a5 as top-level fields.

Makevect: syntax and options

There is one option, it is required, and you specify it only once. The syntax is as follows:

```
makevect
  -name field_name
```

Table 105. makevect Operator Option

Option	Use
-name	<p>-name <i>field_name</i></p> <p>Specifies the beginning <i>field_name</i> of the series of consecutively numbered fields <i>field_name0</i> to <i>field_namen</i> to be combined into a vector called <i>field_name</i>.</p>

Makevect example 1: The makevect operator

Here is the schema of a data set given as input to the makevect operator on fields whose names begin with a. In the example, the operator runs sequentially.

```
record ( a0:uint8; a1:uint8; a2:uint8; a3:uint8; a4:uint8; )
```

Here is the schema of the corresponding output data set:

```
record ( a[5]:uint8; )
```

The osh command is:

```
$ osh "... makevect -name a ..."
```

Here are the input and output of the operation:

Input Data Set	Output Data Set
a0:0 a1:6 a2:0 a3:2 a4:8	a:[0:0 1:6 2:0 3:2 4:8]
a0:20 a1:42 a2:16 a3:10 a4:28	a:[0:20 1:42 2:16 3:10 4:28]
a0:40 a1:78 a2:32 a3:18 a4:48	a:[0:40 1:78 2:32 3:18 4:48]
a0:5 a1:15 a2:4 a3:4 a4:13	a:[0:15 1:33 2:12 3:8 4:23]
a0:25 a1:51 a2:20 a3:12 a4:33	a:[0:5 1:15 2:4 3:4 4:13]
a0:45 a1:87 a2:36 a3:20 a4:53	a:[0:25 1:51 2:20 3:12 4:33]
a0:10 a1:24 a2:8 a3:6 a4:18	a:[0:45 1:87 2:36 3:20 4:53]
a0:30 a1:60 a2:24 a3:14 a4:38	a:[0:35 1:69 2:28 3:16 4:43]
a0:15 a1:33 a2:12 a3:8 a4:23	a:[0:10 1:24 2:8 3:6 4:18]
a0:35 a1:69 a2:28 a3:16 a4:43	a:[0:30 1:60 2:24 3:14 4:38]

Note that the operator did not write the output records in the same order as the input records, although all fields appear in the correct order.

Example 2: The makevect operator with missing input fields

Here is the schema of a data set given as input to the makevect operator on fields whose names begin with a. Field a3 is missing from the series of fields that are combined into a vector. One field (c) does not have the same name as those in the series.

```
record ( a0:uint8; a1:uint8; a2:uint8; a4:uint8; c:decimal[2,1]; )
```

Here is the output schema of the same data after the makevect operation.

```
record ( a4:uint8; c:decimal[2,1]; a[3]:uint8; )
```

Here is the osh command for this example:

```
$ osh "... makevect -name a ..."
```

Note that the operator:

- Stopped combining 'a' fields into a vector when it encountered a field that was not consecutively numbered.
- Wrote the remaining, non-consecutively numbered field (a4) as a top-level one.
- Transferred the unmatched field (c) without alteration.
- Did not write the output records in the same order as the input records, although all fields appear in the correct order.

Input Data Set	Output Data Set
a0:0 a1:6 a2:0 a4:8 c:0:0	a4:8 c:0:0 a:[0:0 1:6 2:0]
a0:20 a1:42 a2:16 a4:28 c:2:8	a4:28 c:2:8 a:[0:20 1:42 2:16]
a0:40 a1:78 a2:32 a4:48 c:5:6	a4:48 c:5:6 a:[0:40 1:78 2:32]
a0:5 a1:15 a2:4 a4:13 c:0:7	a4:13 c:0:7 a:[0:5 1:15 2:4]
a0:25 a1:51 a2:20 a4:33 c:3:5	a4:23 c:2:0 a:[0:15 1:33 2:12]
a0:45 a1:87 a2:36 a4:53 c:6:3	a4:33 c:3:5 a:[0:25 1:51 2:20]
a0:10 a1:24 a2:8 a4:18 c:1:4	a4:53 c:6:3 a:[0:45 1:87 2:36]
a0:30 a1:60 a2:24 a4:38 c:4:2	a4:18 c:1:4 a:[0:10 1:24 2:8]
a0:15 a1:33 a2:12 a4:23 c:2:0	a4:38 c:4:2 a:[0:30 1:60 2:24]
a0:35 a1:69 a2:28 a4:43 c:4:9	a4:43 c:4:9 a:[0:35 1:69 2:28]

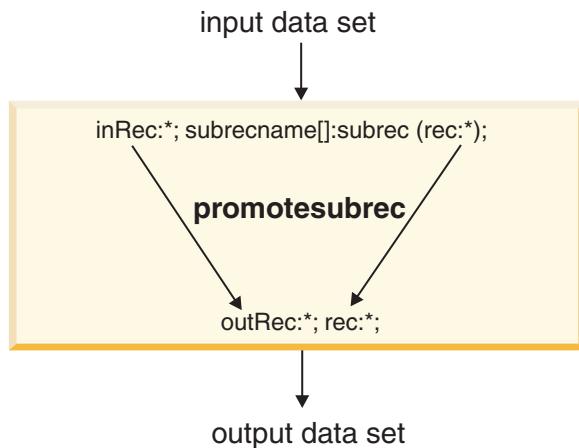
Here are the input and output of a makevect operation on the field a:

The promotesubrec Operator

The promotesubrec operator promotes the fields of an input subrecord to top-level fields. The number of output records equals the number of subrecord elements. The data types of the input subrecord fields determine those of the corresponding top-level fields.

The aggtorec operator often performs the inverse operation; see "The aggtorec Operator" .

Data Flow Diagram



promotesubrec: properties

Table 106. promotesubrec Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec: *; subrecname[]:subrec (rec: *);
Output interface schema	outRec: *; rec: *;
Transfer behavior	See below.

The input interface schema is as follows:

```
record ( inRec: *; subrecname[]:subrec (rec: *); )
```

where inRec does not include the subrecord to be promoted.

The output interface schema is as follows:

```
record ( outRec: *; rec: *; )
```

where outRec includes the same fields as inRec (top-level fields), and rec includes the fields of the subrecord.

Promotesubrec: syntax and options

There is one option. It is required. You might specify it only once.

```
promotesubrec -subrecname subrecname
```

Table 107. promotesubrec Operator Option

Option	Use
-subrecname	<pre>-subrecname subrecname</pre> <p>Specifies the name of the subrecord whose elements are promoted to top-level records.</p>

Example

Here is the schema of a data set given as input to the promotesubrec operation on the subrecord named sub. In the example, the operator runs sequentially.

```
record ( d:string[1]; sub []:subrec( a:uint8; b:time; c:date; ))
```

Here is the schema of the output data:

```
record (d:string[1]; a:uint8; b:time; c:date; )
```

Here is the osh command:

```
$ osh "... promotesubrec -subrecname sub..."
```

Here is the input of the operation:

```
d:A sub:[0:(sub.a:1 sub.b:00:11:01 sub.c:1960-01-02)
          1:(sub.a:3 sub.b:08:45:54 sub.c:1946-09-15)]
d:B sub:[0:(sub.a:1 sub.b:12:59:01 sub.c:1955-12-22)
          1:(sub.a:2 sub.b:07:33:04 sub.c:1950-03-10)
```

```

2:(sub.a:2 sub.b:12:00:00 sub.c:1967-02-06)
3:(sub.a:2 sub.b:07:37:04 sub.c:1950-03-10)
4:(sub.a:3 sub.b:07:56:03 sub.c:1977-04-14)
5:(sub.a:3 sub.b:09:58:02 sub.c:1960-05-18)]

```

Here is the output of the operation:

```

d:A a:1 b:00:11:01 c:1960-01-02
d:A a:3 b:08:45:54 c:1946-09-15
d:B a:1 b:12:59:01 c:1955-12-22
d:B a:2 b:07:33:04 c:1950-03-10
d:B a:2 b:12:00:00 c:1967-02-06
d:B a:2 b:07:37:04 c:1950-03-10
d:B a:3 b:07:56:03 c:1977-04-14
d:B a:3 b:09:58:02 c:1960-05-18

```

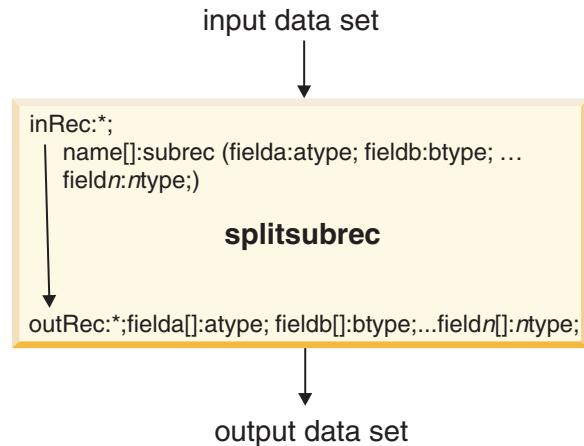
The splitsubrec Operator

The splitsubrec operator separates an input subrecord into a set of top-level vector fields. The operator creates one new vector field for each element of the original subrecord. That is, each top-level vector field that is created has the same number of elements as the subrecord from which it was created. The operator outputs fields of the same name and data type as those of the fields that comprise the subrecord.

Note: Input subrecord fields cannot be vectors.

The makesubrec operator performs the inverse operation. See "The makesubrec Operator".

Data Flow Diagram



splitsubrec properties

Table 108. splitsubrec Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec:*\nname[] subrec (a:atype; b:btype; ...n:ntype;)
Output interface schema	outRec:*\n a[]:atype; b[]:btype; ...n[]:nype;
Transfer behavior	See below.

The input interface schema is as follows:

```
record ( inRec:*; name[] subrec (subfielda:atype;  
subfieldb:btype; ...subfieldn:ntype;) )
```

The inRec schema variable omits the input field whose elements are separated and promoted. The subfields to be promoted can be of any data type except tagged or vector.

The output interface schema is as follows:

```
record ( outRec:*; fielda[]:atype; fieldb[]:btype; ... fieldn[]:nype; )
```

Each new vector field has the same name and data type as its corresponding input subfield.

This operator also works with scalar components, treating them as if they were vectors of length 1.

Splitsubrec: syntax and options

There is one option. It is required. You might specify it only once. The syntax is as follows:

```
splitsubrec -subrecname subrecname ... "
```

Table 109. splitsubrec Operator Options

Option	Use
-subrecname	-subrecname subrecname Required. Specifies the name of the subrecord whose elements you want to promote to a set of similarly named and typed top-level fields.

Example

In this example, which runs sequentially:

- The input schema is as follows:

```
record (s[5]:subrec( a:uint8; b:decimal[2,1]; ))
```
- The splitsubrec operator separates and promotes subfields a and b of subrecord s.
- The output schema is as follows:

```
record ( a[5]:uint8; b[5]:decimal[2,1]; )
```

Here is one record of the input data set:

```
s:[0:(s.a:8 s.b:-0.6)  
 1:(s.a:10 s.b:4.5)  
 2:(s.a:0 s.b:5.1)  
 3:(s.a:2 s.b:-8.5)  
 4:(s.a:4 s.b:2.1)]
```

Here is the corresponding output record:

```
a:[0:8 1:10 2:0 3:2 4:4] b:[0:-0.6 1:4.5 2:5.1 3:-8.5 4:2.1]
```

Here is the osh command for this example:

```
$ osh "... splitsubrec -subrecname s ..."
```

Here is the entire input data set:

```
s:[0:(s.a:8 s.b:-0.6)  
 1:(s.a:10 s.b:4.5)  
 2:(s.a:0 s.b:5.1)  
 3:(s.a:2 s.b:-8.5)
```

```

4:(s.a:4 s.b:2.1)]
s:[0:(s.a:10 s.b:5.8)
 1:(s.a:0 s.b:-1.7)
 2:(s.a:2 s.b:1.6)
 3:(s.a:4 s.b:-5.0)
 4:(s.a:6 s.b:4.6)]
s:[0:(s.a:6 s.b:9.1)
 1:(s.a:8 s.b:4.1)
 2:(s.a:10 s.b:-2.3)
 3:(s.a:0 s.b:9.0)
 4:(s.a:2 s.b:2.2)]
s:[0:(s.a:0 s.b:3.6)
 1:(s.a:2 s.b:-5.3)
 2:(s.a:4 s.b:-0.5)
 3:(s.a:6 s.b:-3.2)
 4:(s.a:8 s.b:4.6)]
s:[0:(s.a:4 s.b:8.5)
 1:(s.a:6 s.b:-6.6)
 2:(s.a:8 s.b:6.5)
 3:(s.a:10 s.b:8.9)
 4:(s.a:0 s.b:4.4)]

```

Here is the entire output data set:

```

a:[0:6 1:8 2:10 3:0 4:2] b:[0:9.1 1:4.1 2:-2.3 3:9.0 4:2.2]
a:[0:0 1:2 2:4 3:6 4:8] b:[0:3.6 1:-5.3 2:-0.5 3:-3.2 4:4.6]
a:[0:4 1:6 2:8 3:10 4:0] b:[0:8.5 1:-6.6 2:6.5 3:8.9 4:4.4]
a:[0:8 1:10 2:0 3:2 4:4] b:[0:-0.6 1:4.5 2:5.1 3:-8.5 4:2.1]
a:[0:10 1:0 2:2 3:4 4:6] b:[0:5.8 1:-1.7 2:1.6 3:-5.0 4:4.6]

```

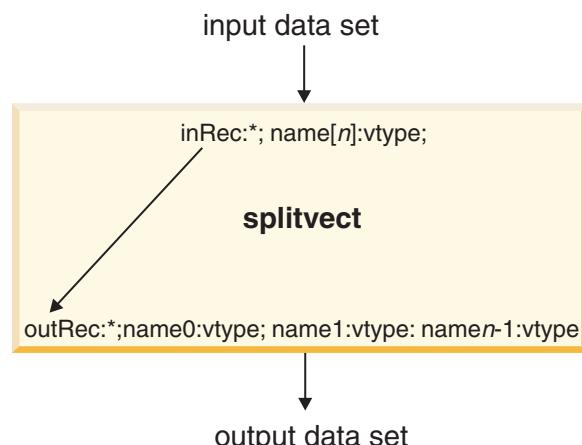
The splitvect operator

The splitvect operator promotes the elements of a fixed-length vector to a set of similarly named top-level fields. The operator creates fields of the format *name*₀ to *name*_{*n*}, where *name* is the original vector's name and 0 and *n* are the first and last elements of the vector.

The makevect operator performs the inverse operation. See "The makevect Operator".

Data flow diagram

The following figure is a conceptual diagram of the splitvect operator, where vtype denotes any valid WebSphere DataStage data type that can figure in a vector, that is, any data type except tagged and * (schema variable):



splitvect: properties

Table 110. splitvect Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec:/*; name[n]:vtype;
Output interface schema	outRec:/*; name0:vtype; name1:vtype; namen-1:vtype;
Transfer behavior	See below.
Execution mode	parallel (default) or sequential

The input interface schema is:

```
record ( inRec:/*; name[n]:vtype; )
```

The inRec schema variable captures all the input record's fields except the vector field whose elements are separated from each other and promoted. The data type of this vector is *vtype*, which denotes any valid WebSphere DataStage data type that can figure in a vector, that is, any data type except tagged and * (schema variable). This data type determines the data type of the output, top-level fields. The vector must be of fixed length.

The output data set has the schema:

```
record ( outRec:/*; name0:vtype; name1:vtype; namen-1:vtype; )
```

Splitvect: syntax and options

There is one option. It is required. You might specify it only once. The syntax is as follows:

```
splitvect  
-name name
```

Table 111. splitvect Operator Option

Option	Use
-name	<pre>-name name</pre> <p>Specifies the name of the vector whose elements you want to promote to a set of similarly named top-level fields.</p>

Example

Here is the input schema of a data set to be processed by splitvect:

```
record ( a[5]:uint8; )
```

Here is the output data set's schema:

```
record ( a0:uint8; a1:uint8; a2:uint8; a3:uint8; a4:uint8; )
```

Here is the osh command:

```
splitvect -name a ..."
```

Here are a sample input and output of the operator:

Input Data Set	Output Data Set
a:[0:0 1:6 2:0 3:2 4:8]	a0:0 a1:6 a2:0 a3:2 a4:8
a:[0:20 1:42 2:16 3:10 4:28]	a0:20 a1:42 a2:16 a3:10 a4:28
a:[0:40 1:78 2:32 3:18 4:48]	a0:40 a1:78 a2:32 a3:18 a4:48
a:[0:15 1:33 2:12 3:8 4:23]	a0:10 a1:24 a2:8 a3:6 a4:18
a:[0:5 1:15 2:4 3:4 4:13]	a0:5 a1:15 a2:4 a3:4 a4:13
a:[0:25 1:51 2:20 3:12 4:33]	a0:30 a1:60 a2:24 a3:14 a4:38
a:[0:45 1:87 2:36 3:20 4:53]	a0:15 a1:33 a2:12 a3:8 a4:23
a:[0:35 1:69 2:28 3:16 4:43]	a0:35 a1:69 a2:28 a3:16 a4:43
a:[0:10 1:24 2:8 3:6 4:18]	a0:25 a1:51 a2:20 a3:12 a4:33
a:[0:30 1:60 2:24 3:14 4:38]	a0:45 a1:87 a2:36 a3:20 a4:53

Here are a sample input and output of the operator:

Note that the operator did not write the output records in the same order as the input records, although all fields appear in the correct order.

The tagbatch operator

Tagged fields are types of aggregate fields that define a nested form of data representation. However, most WebSphere DataStage operators do not support operations on aggregate fields. This section describes an operator that converts tagged fields to a representation usable by other WebSphere DataStage operators.

The tagbatch operator reads input records containing a tagged aggregate and flattens the tag cases into an output record whose schema accommodates all possible fields of the various tag cases.

The tagswitch operator copies each input record to a separate output data set based on the active case of a tagged field in the input record. (See "The tagswitch Operator" .)

Tagged fields and operator limitations

Tagged fields are called aggregate fields in WebSphere DataStage. Most WebSphere DataStage operators do not directly support operations on aggregate fields but act on toplevel fields only. For example, the WebSphere DataStage statistics operator can calculate statistical information only on the top-level fields of an input data set. It cannot process subrecords or nested fields.

The tagbatch operator removes this limitation, because it extracts the cases of a tagged field and raises them to the top level of a record schema.

Tagged fields let you access the data storage of a single record field as different data types. For example, you might have one record where the tagged field contains string data and another record where the field contains a floating-point value. By using a tagged field, each record can use the data type for the field that matches the data stored within it.

You typically use tagged fields when importing data from a COBOL data file, if the COBOL data definition contains a REDEFINES statement. A COBOL REDEFINES statement specifies alternative data types for a single field.

A tagged field has multiple data types, called tag cases. Each case of the tagged is defined with both a name and a data type. The following example shows a record schema that defines a tagged field that has three cases corresponding to three different data types:

```
record ( tagField:tagged
  (
    aField:string;          // tag case 0
    bField:int32;           // tag case 1
    cField:sfloat;          // tag case 2
  );
)
```

The content of a tagged field can be any one of the subfields defined for the tagged field. In this example, the content of tagField is one of the following: a variable-length string, a 32-bit integer, a single-precision floating-point value. A tagged field in a record can have only one item from the list. This case is called the active case of the tagged.

"Dot" addressing refers to the components of a tagged field: For example, tagField.aField references the aField case, tagField.bField references the bField case, and so on.

Tagged fields can contain subrecord fields. For example, the following record schema defines a tagged field made up of three subrecord fields:

```
record ( tagField:tagged
  (
    aField:subrec(
      aField_s1:int8;
      aField_s2:uint32);
    bField:subrec(
      bField_s1:string[10];
      bField_s2:uint32);
    cField:subrec(
      cField_s1:uint8;
      cField_s2:dfloat);
  );
)
```

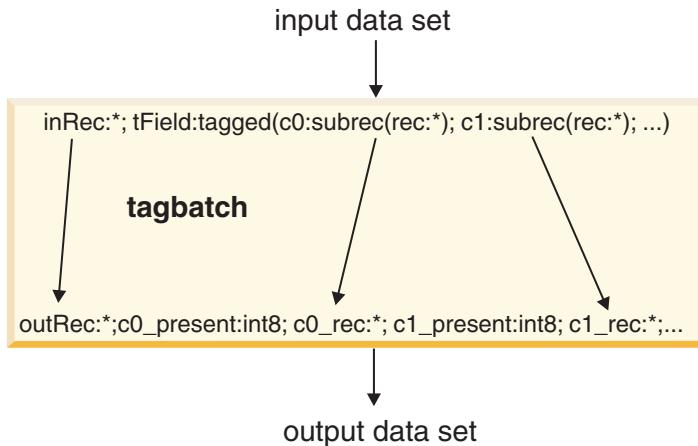
In this example, each subrecord is a case of the tagged. You reference cField_s1 of the cField case as tagField.cField.cField_s1.

See the topic on WebSphere DataStage data sets in the *WebSphere DataStage Parallel Job Developer Guide* for more information on tagged fields.

Operator action and transfer behavior

The tagbatch operator reads input records containing a tagged aggregate and flattens the tag cases into an output record whose schema accommodates all possible fields of the various tag cases. The operator groups input records based on one or more key fields and copies all records in the group to a single output record. The output record contains both all non-tagged fields of the input record and all cases of the tagged field as top-level fields.

Data flow diagram



tagbatch: properties

Table 112. tagbatch Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec: *; tField:tagged(c0:subrec(rec: *); c1:subrec(rec: *); ...);
Output interface schema	outRec: *; c0_present:int8; c0_rec: *; c1_present:int8; c1_rec: *; ...
Transfer behavior	See below.

The input interface schema is:

`inRec: *; tField:tagged(c0:subrec(rec: *); c1:subrec(rec: *); ...);`

By default, the inRec schema variable does not include the original tagged field.

The output interface schema is:

`outRec: *; c0_present:int8; c0_rec: *; c1_present:int8; c1_rec: *; ...`

Added, missing, and duplicate fields

When tagged fields are promoted to top-level fields, each is preceded by an int8 field called case_present, where case is the name of the case. If the promoted field contains a value, the case_present field is set to 1. If the promoted field does not contain a value, the case_present field is set to 0 and the promoted field is set to the default value for its data type.

If two records with the same key value have the same case for the tagged field, the operator outputs the case value of the first record processed by the operator. However, you can modify this behavior by means of the -ifDuplicateCase option.

If a tagged case is missing from a record, the operator outputs a record with the corresponding present field set to 0 and the field of the tag case is set to the default value for its data type. However, you can modify this behavior by means of the -ifMissingCase option.

"Example 2: The tagbatch Operator, Missing and Duplicate Cases" shows the behavior described in the three preceding paragraphs.

Input data set requirements

The input data set must meet these requirements:

- The input data set must contain a tagged field.
- Each case of the tag must consist of a subrecord field.
- Input records with identical key field values must be adjacent in the input data set. That is, you must first hash partition and sort the input data set by the key fields you then specify to the tagbatch operator.

Tagbatch: syntax and options

The -key option is required. The syntax is as follows:

```
tagbatch
  -key field_name [-ci | -cs] [-param params]
  [-key field_name [-ci | -cs] [-param params] ...]
  [-collation_sequence locale | collation_file_pathname | OFF]
  [-ifDuplicateCase drop | fail | keepFirst | keepLast]
  [-ifMissingCase continue | drop | fail]
  [-nobatch]
  [-tag tag_field]
```

Table 113. tagbatch Operator Options

Option	Use
-key	<p>-key <i>field_name</i> [-ci -cs] [-param <i>params</i>]</p> <p>Specifies the name of the key field used to combine input records. The key option might be repeated for multiple key fields. See "Example 3: The tagbatch Operator with Multiple Keys".</p> <p>-ci or -cs are optional arguments for specifying case-insensitive or case-sensitive keys. By default, the operator uses case-sensitive matching. Be sure to specify the same case sensitivity here as you did when you sorted the input data set.</p> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property = value</i> pairs separated by commas.</p>

Table 113. tagbatch Operator Options (continued)

Option	Use
-collation_sequence	<p><code>-collation_sequence locale collation_file_pathname OFF</code></p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> • Specify a predefined IBM ICU locale • Write your own collation sequence using ICU syntax, and supply its <code>collation_file_pathname</code> • Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.htm</p>
-ifDuplicateCase	<p><code>-ifDuplicateCase drop fail keepFirst keepLast</code></p> <p>Controls how the operator handles two records with the same key value that have the same active case for the tagged field. Suboptions include:</p> <p>drop: Drop the output record. This means that no record is output for that key value. See "Example 2: The tagbatch Operator, Missing and Duplicate Cases" .</p> <p>fail: Fail the operator when two or more records have the same active case.</p> <p>keepFirst: Default. The operator outputs the case value from the first record processed by the operator.</p> <p>keepLast: Output the case from the last record. See "Example 2: The tagbatch Operator, Missing and Duplicate Cases" .</p>
-nobatch	<p><code>-nobatch</code></p> <p>Optionally specify that a record is output for every input record. The default is to place input records into groups based on key values and output them as a single record.</p>

Table 113. tagbatch Operator Options (continued)

Option	Use
-ifMissingCase	<p>-ifMissingCase continue drop fail</p> <p>Optionally specifies how the operator handles missing tag cases for a specified key field. By default, the operator outputs a record with the corresponding present field set to 0 and the field of the tag case set to their default values (0 for numeric fields, 0 length for variable-length strings, and so on).</p> <p>The suboptions are:</p> <ul style="list-style-type: none"> continue: Default. Outputs a record with the corresponding present field set to 0 and the field of the tag case set to their default values (0 for numeric fields, 0 length for variable-length strings, and so on). drop: Drop the record. See "Example 2: The tagbatch Operator, Missing and Duplicate Cases". fail: Fail the operator and step.
-tag	<p>-tag <i>tag_field</i></p> <p>Specifies the name of the tagged field in the input data set used by the operator. If you do not specify this option, the input data set should have only one tagged field. If it has more than one tagged field, the operator processes only the first one and issues a warning when encountering others.</p>

Tagbatch example 1: simple flattening of tag cases

This example shows an input data set and its processing by the tagbatch operator. The name of the key field used to combine input records is key. The data set has been hash partitioned and sorted.

The operator runs sequentially. Here are the input and output schemas:

Input Schema	Output Schema
key:int32;	key:int32;
t:tagged	
(A:subrec(fname:string; lname:string;);	A_present:int8; fname:string; lname:string;
B:subrec(income:int32;);	B_present:int8; income:int32;
C:subrec(birth:date; retire:date;);)	C_present:int8; birth:date; retire:date;

In the following data set representations, the elements of a tagged field are enclosed in angle brackets (<>); the elements of a subrecord are enclosed in parentheses (()).

Here are three tagged records whose key field is equal:

```
key:11 t:<t.A:(t.A.fname:booker t.A.lname:faibus)>
key:11 t:<t.B:(t.B.income:27000)>
key:11 t:<t.C:(t.C.birth:1962-02-06 t.C.retire:2024-02-06)>
```

Here is the equivalent output record:

```
key:11 A_present:1 fname:booker lname:faubus B_present:1 income:27000 C_present:1 birth:1962-02-06 retire:2024-02-06
```

The fields A_present, B_present, and C_present now precede each output field and are set to 1, since the field that follows each contains a value.

Here is the osh command for this example:

```
$ osh "... tagbatch -key key ..."
```

Here are the input and output data sets without field names:

Input Data Set	Output Data Set
11 <(booker faubus)> 11 <(27000)> 11 <(1962-02-06 2024-0206)>	11 1 booker faubus 1 27000 1 1962-02-06 2024-02-06
22 <(marilyn hall)> 22 <(35000)> 22 <(1950-09-20 2010-0920)>	22 1 marilyn hall 1 35000 1 1950-09-20 2010-09-20
33 <(gerard devries)> 33 <(50000)> 33 <(1944-12-23 2009-1223)>	33 1 gerard devries 1 50000 1 1944-12-23 2009-12-23
44 <(ophelia oliver)> 44 <(65000)> 44 <(1970-04-11 2035-0411)>	44 1 ophelia oliver 1 65000 1 1970-04-11 2035-04-11
55 <(omar khayam)> 55 <(42000)> 55 <(1980-06-06 2040-0606)>	55 1 omar khayam 1 42000 1 1980-06-06 2040-06-06

Example 2: The tagbatch operator, missing and duplicate cases

This example shows an input data set and its processing by the tagbatch operator. The name of the key field used to combine input records is key. The data set has been hash-partitioned and sorted. The operator runs sequentially.

The input and output schemas are identical to those shown in "Example 1: The tagbatch Operator, Simple Flattening of Tag Cases". The input data set is identical to that shown in the same example, except that one field is missing from the input data set and one occurs twice. Accordingly:

- The -ifMissingCase option has been set to drop.
- The -ifDuplicateCase option has been set to keepLast.

Here is the osh command for this example:

```
$ osh "... tagbatch -key key -ifMissingCase drop  
-ifDuplicateCase keepLast ..."
```

Here is the input data set without field names. The income field is missing from those containing a -key whose value is 33. The corresponding fields are shown in boldface type. The income field occurs twice in those containing a key whose value is 44. The corresponding fields are shown in italic type.

```
11 <(booker faubus)>  
11 <(27000)>  
11 <(1962-02-06 2024-02-06)>  
22 <(marilyn hall)>  
22 <(35000)>  
22 <(1950-09-20 2010-09-20)>  
33 <(gerard devries)>  
33 <(1944-12-23 2009-12-23)>  
44 <(ophelia oliver)>
```

```

44 <(65000)>
44 <(60000)>
44 <(1970-04-11 2035-04-11)>55 <(omar khayam)>
55 <(42000)>
55 <(1980-06-06 2040-06-06)>

```

Here is the output data set without field names.

```

11 1 booker faibus 1 27000 1 1962-02-06 2024-02-06
22 1 marilyn hall 1 35000 1 1950-09-20 2010-09-20
44 1 ophelia oliver 1 60000 1 1970-04-11 2035-04-1155 1 omar khayam 1 42000 1 1980-06-06 2040-06-06

```

The operator has dropped the field containing a key whose value is 33 and has written the last value of the duplicate field (shown in italic type).

Example 3: The tagbatch operator with multiple keys

This example shows an input data set and its processing by the tagbatch operator. Two key fields are used to combine input records, *fname* and *lname*. The data set has been hash partitioned and sorted. Here are the input and output schemas:

Input Schema	Output Schema
<pre> fname:string; lname:string; t:tagged (emp_info:subrec (start_date:date; emp_id:int32;); compensation:subrec (title:string {delim=' OR '} ; grade:int16; salary:int32;); personal:subrec (birth:date; ssn:string;);) </pre>	<pre> fname:string; lname:string; emp_info_present:int8; start_date:date; emp_id:int32; compensation_present:int8; title:string; grade:int16; salary:int32; personal_present:int8; birth:date; ssn:string; </pre>

Here is the osh command for this example:

```
$ osh "... tagbatch -key fname -key lname ..."
```

Here are three fields of the input data set:

```

Rosalind Fleur <(VP sales 1 120000)>
Rosalind Fleur <(1998-01-03 456)>
Rosalind Fleur <(1950-04-16 333-33-3333)>

```

Here is their corresponding output:

```
Rosalind Fleur 1 1998-01-03 456 1 VP sales 1 120000 1 1950-04-16 333-33-3333
```

Here is the entire input data set:

```

Jane Doe <(principal engineer 3 72000)>
Jane Doe <(1991-11-01 137)>
Jane Doe <(1950-04-16 111-11-111100)>
John Doe <(senior engineer 3 42000)>
John Doe <(1992-03-12 136)>
John Doe <(1960-11-12 000-00-0000)>
Rosalind Elias <(senior marketer 2 66000)>
Rosalind Elias <(1994-07-14 208)>
Rosalind Elias <(1959-10-16 222-22-2222)>
Rosalind Fleur <(VP sales 1 120000)>
Rosalind Fleur <(1998-01-03 456)>
Rosalind Fleur <(1950-04-16 333-33-3333)>

```

Here is the entire output data set:

```

Jane Doe 1 1991-11-01 137 1 principal engineer 3 72000 1 1950-04-16 111-11- 111100
John Doe 1 1992-03-12 136 1 senior engineer 3 42000 1 1960-11-12 000-00-0000
Rosalind Elias 1 1994-07-14 208 1 senior marketer 2 66000 1 1959- 10-16 222-22-2222
Rosalind Fleur 1 1998-01-03 456 1 VP sales 1 120000 1 1950-04-16 333-33-3333

```

The tagbatch operator has processed the input data set according to both the fname and lname key fields, because choosing only one of these as the key field gives incomplete results.

If you specify only the fname field as the key, the second set of tagged fields whose fname is Rosalind is dropped from the output data set. The output data set, which is as follows, does not contain the data corresponding to Rosalind Fleur:

```

Jane Doe 1 1991-11-01 137 1 principal engineer 3 72000 1 1950-04- 16 111-11- 111100
John Doe 1 1992-03-12 136 1 senior engineer 3 42000 1 1960-11-12 000-00-0000
Rosalind Elias 1 1994-07-14 208 1 senior marketer 2 66000 1 1959- 10-16 222-22- 2222

```

If you specify only the lname field as the key, the second set of tagged fields whose lname is Doe is dropped from output data set. The output data set, which follows, does not contain the data corresponding to John Doe.

```

Jane Doe 1 1991-11-01 137 1 principal engineer 3 72000 1 1950-04- 16 111-11- 111100
Rosalind Elias 1 1994-07-14 208 1 senior marketer 2 66000 1 1959- 10-16 222-22- 2222
Rosalind Fleur 1 1998-01-03 456 1 VP sales 1 120000 1 1950-04-16 333-33-3333

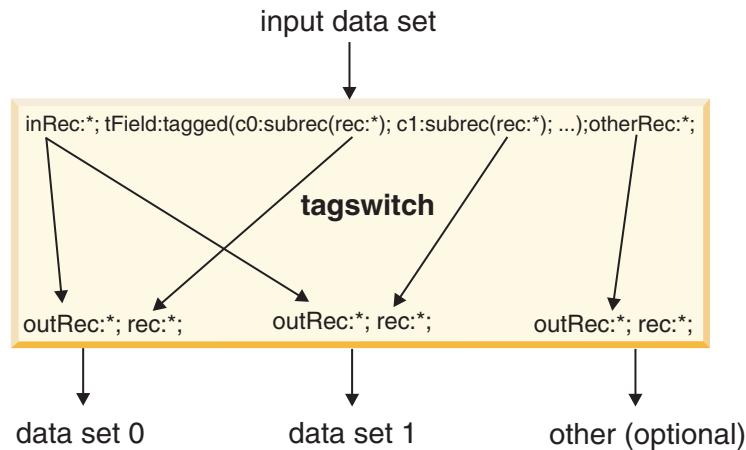
```

The tagswitch operator

"Tagged Fields and Operator Limitations" discusses the characteristics and limitations of tagged fields. WebSphere DataStage operators do not support operations on this type of field. This section describes an operator that converts tagged fields to a representation usable by other WebSphere DataStage operators.

The tagswitch operator transfers the contents of an input data set containing tagged aggregates to one or more output data sets. The output set or sets to which input data is transferred is determined by the tag value of the tagged aggregate. The output data sets contain the cases of the tagged field promoted to the top level of the data set's record schema. Top-level fields of the input records are transferred to all output data sets.

Data flow diagram



tagswitch: properties

Table 114. tagswitch Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1 <= n, where n is the number of cases; 1 <= n+1, if the operator transfers unlisted cases.
Input interface schema	inRec: *; tField:tagged(c0:subrec(rec: *); c1:subrec(rec: *); ..); otherRec: *;
Output interface schema	Output data set N: outRec: *; rec: *; Optional other output data set: otherRec: *;
Transfer behavior	See below.

Input and output interface schemas

The input interface schema is:

```
inRec: *; tField:tagged(c0:subrec(rec: *); c1:subrec(rec: *); ..);  
otherRec: *;
```

By default, the inRec schema variable does not include the original tagged field or fields. However, you can override this default by means of the modify operator to force the operator to retain the tagged field, so that it is transferred from the input data set to the output data sets.

The output interface schema is:

```
Output dataset N: outRec: *; rec: *;  
Optional other output dataset: otherRec: *;
```

Top-level fields are always copied from the input data set to all output data sets.

The case option

By default, all cases of the tagged field are copied to the output data sets. However, you can override this default by means of the -case option. When you specify this option, only the cases that you identify are copied to the output. The order in which you specify the -case option is important: the first instance of -case defines the tag case copied to output data set 0, the next defines the tag case copied to data set 1, and so on.

When you specify the -case option, by default the operator drops any record containing an unspecified case. You can override the default behavior by means of the -ifUnlistedCase option. This option controls operator behavior when an unspecified case is encountered: the option lets you either terminate the operator and the step that contains it or copy the record to the last data set attached to the operator.

Note: If you choose -ifUnlistedCase without first specifying -case, the operator ignores the option and defaults to all cases.

Using the operator

The input data set must contain a tagged field. Each case of the tag must consist of a subrecord field.

There are no required options. By default, the operator processes all cases of the input tagged field. In this event, connect one output data set for each case of the tagged field in the input data set.

You can override the default and explicitly list tag cases to be processed by means of the `-tag` option. If you choose the `-tag` option, you can control how the operator behaves when it encounters an unlisted case.

Tagswitch: syntax and options

The syntax is as follows. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose the value in single quotes.

```
tagswitch
[-tag tag_field ]
[-allCases] | [-case case ...]
[-ifUnlistedCase drop | fail | other]
```

Table 115. tagswitch Operator Options

Option	Use
<code>-allCases</code>	<p><code>-allCases</code></p> <p>Default. The operator processes all cases of the input tagged field. Specify this option only to explicitly set the action of the operator. This option is mutually exclusive with the <code>-case</code> option.</p>
<code>-case</code>	<p><code>-case <i>case</i></code></p> <p>Specifies the case of the tagged field that you want to extract from the input tagged field. You can specify multiple <code>-case</code> options. For each, specify both the option and its argument. The operator produces one output data set for each <code>-case</code> option.</p> <p>The order in which you specify the <code>-case</code> option is important: the first instance of <code>-case</code> defines the tag case copied to output data set 0, the next defines those records copied to data set 1, and so on.</p> <p>Connect one output data set for each specified case. However, if you specify the <code>-other</code> suboption of the <code>-ifUnlistedCase</code> option (see below), connect $n+1$ data sets to the operator, where n is the number of cases that you have specified.</p> <p>This option is mutually exclusive with the <code>-allCases</code> option.</p>
<code>-ifUnlistedCase</code>	<p><code>-ifUnlistedCase drop fail other</code></p> <p>If you specify the <code>-case</code> option to explicitly set the cases of the tagged processed by the operator, you can also specify <code>-ifUnlistedCase</code> to control how the operator treats unspecified cases. The options are:</p> <ul style="list-style-type: none"> -drop: Default. Drop the record and issue a warning message. -fail: Fail the operator and step when an unspecified tag is encountered. -other: Copy the record to the other output data set. This is the last output data set of the operator. If you specify this option, attach $n+1$ data sets to the operator, where n is the number of cases that you have specified.

Table 115. *tagswitch* Operator Options (continued)

Option	Use
-tag	<p><code>-tag tag_field</code></p> <p>Specifies the name of the tagged field in the input data set used by the operator. If you do not specify a -tag, the operator acts on the first tag field it encounters.</p>

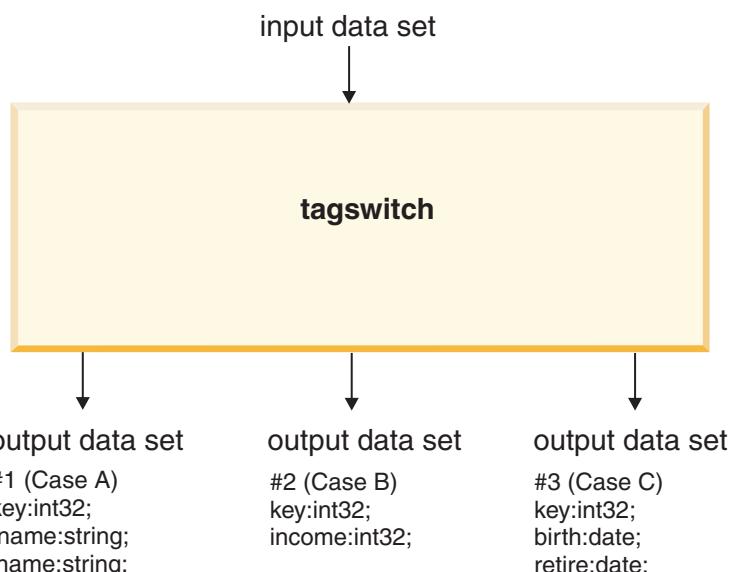
Tagswitch example 1: default behavior

This example shows the default behavior of the *tagswitch* operator. No option has been chosen.

The input data set contains three tag cases. Accordingly, three output data sets have been attached to the operator. The operator automatically writes the same number of data sets as there are tagged cases. The input data set has been sorted on the -key field for the purpose of demonstration, although the field does not have to be. The operator runs sequentially.

input record schema:

```
key:int32;
t:tagged
(A:subrec(fname:string;
    lname:string;
    );
B:subrec(income:int32;
    );
C:subrec(birth:date;
    retire:date;
    );
```



Here are the input and output schemas:

Input Schema	Output Schemas
<pre>key:int32; t:tagged (A:subrec(fname:string; lname:string;); B:subrec(income:int32;); C:subrec(birth:date; retire:date;);</pre>	<pre>dataset # 1 (Case A) key:int32; fname:string; lname:string; dataset # 2 (Case B) key:int32; income:int32; dataset # 3 (Case C) key:int32; birth:date; retire:date;</pre>

Here is the osh command for this example:

```
tagswitch -tag tag < in.ds > out0.ds > out1.ds > out2.ds"
```

Here are the input data set and output data sets without field names:

Input Data Set	Output Data Sets
11 <(booker faubus)> 11 <(27000)> 11 <(1962-02-06 2024-02-06)> 22 <(marilyn hall)> 22 <(35000)> 22 <(1950-09-20 2010-09-20)> 33 <(gerard devries)> 33 <(50000)> 33 <(1944-12-23 2009-12-23)> 44 <(ophelia oliver)> 44 <(65000)> 44 <(1970-04-11 2035-04-11)> 55 <(omar khayam)> 55 <(42000)> 55 <(1980-06-06 2040-06-06)>	Data Set # 1 (Case A) 11 booker faubus 22 marilyn hall 33 gerard devries 44 ophelia oliver 55 omar khayam Data Set # 2 (Case B) 11 27000 55 42000 22 35000 44 60000 44 65000 Data Set # 3 (Case C) 11 1962-02-06 2024-02-06 55 1980-06-06 2040-06-06 33 1944-12-23 2009-12-23 22 1950-09-20 2010-09-20 44 1970-04-11 2035-04-11

Each case (A, B, and C) has been promoted to a record. Each subrecord of the case has been promoted to a top-level field of the record.

Example 2: the tagswitch operator, one case chosen

This example shows the action of the tagswitch operator on the same input data set as in "Example 1: The tagswitch Operator, Default Behavior".

In this program, one case of three possible cases (case B) is chosen as the case of the tagged aggregate from which output is extracted. All other cases, which are unspecified, are copied to an other output data set. Accordingly, two output data sets are attached to the operator, one for the specified case and one for any unspecified cases.

The input data set has been sorted on the key field for the purpose of demonstration, although the field does not have to be. The operator runs sequentially.

input record schema:

```
key:int32;
t:tagged
(A:subrec(fname:string;
    lname:string;
);
B:subrec(income:int32;
);
C:subrec(birth:date;
    retire:date;
);
```

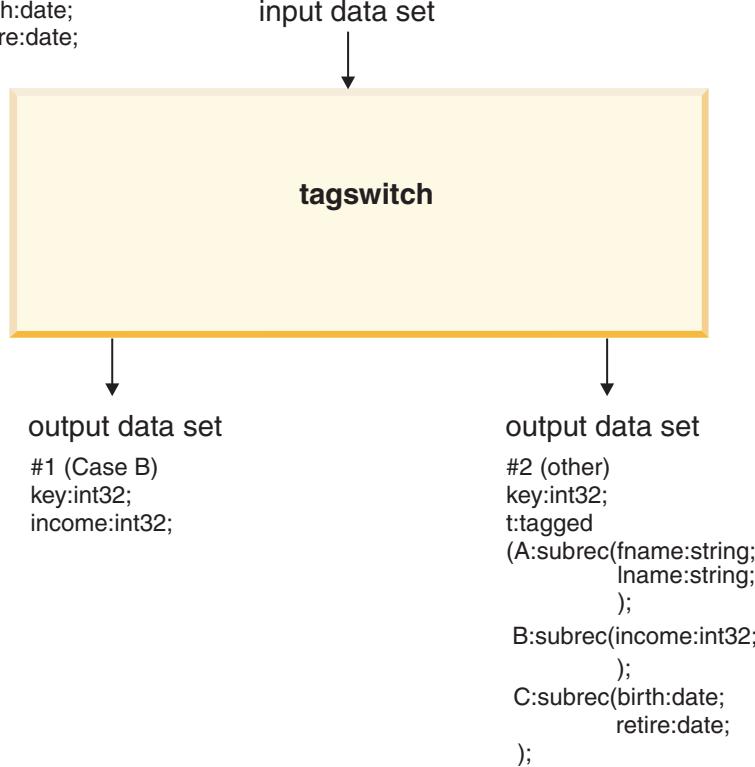


Table 116. Input and output schemas

Input Schema	Output Schemas
<pre>key:int32; t:tagged (A:subrec(fname:string; lname:string;); B:subrec(income:int32;); C:subrec(birth:date; retire:date;);</pre>	<pre>Data set # 1 (Case B) key:int32; income:int32; Data set # 2 (other) key:int32; t:tagged (A:subrec(fname:string; lname:string;); B:subrec(income:int32;); C:subrec(birth:date; retire:date;);</pre>

Here are the input and output schemas:

Note that the input schema is copied intact to the output schema. Case B has not been removed from the output schema. If it were, the remaining tags would have to be rematched.

Here is the osh command for this example:

```
osh "tagswitch -case B -ifUnlistedCase other < in.ds > out0.ds 1> out1.ds >"
```

Here are the input and output data sets.

Table 117. Input and output data sets

Input Data Set	Output Data Sets
11 <(booker faubus)> 11 <(27000)> 11 <(1962-02-06 2024-02-06)> 22 <(marilyn hall)> 22 <(35000)> 22 <(1950-09-20 2010-09-20)> 33 <(gerard devries)> 33 <(50000)> 33 <(1944-12-23 2009-12-23)> 44 <(ophelia oliver)> 44 <(65000)> 44 <(1970-04-11 2035-04-11)> 55 <(omar khayam)> 55 <(42000)> 55 <(1980-06-06 2040-06-06)>	Data set # 1 11 27000 22 35000 33 50000 44 65000 55 42000 Data set # 2 11 <(booker faubus)> 11 <(1962-02-06 2024-02-06)> 22 <(marilyn hall)> 22 <(1950-09-20 2010-09-20)> 33 <(gerard devries)> 33 <(1944-12-23 2009-12-23)> 44 <(ophelia oliver)> 44 <(1970-04-11 2035-04-11)> 55 <(omar khayam)> 55 <(1980-06-06 2040-06-06)>

Chapter 12. The sorting library

Use the sort operators when you want to explicitly control the sorting behavior of an operator.

By default, WebSphere DataStage inserts partition and sort components in your data flow to meet the partitioning and sorting needs of your use of WebSphere DataStage's predefined operators. But you can use these operators to control the sorting operation more specifically.

The sorting library contains these operators:

- The tsort operator, which requires no additional sorting software to run. This operator is described on page [The tsort Operator](#). Because of the additional capabilities in tsort, it is recommended that you use this operator rather than psort.
- The psort operator, which requires UNIX sort. This operator is described on page [The psort Operator](#).

The sorting operators sort the records of a data set by the values of one or more key fields in each record. The operators can be used in either parallel or sequential mode.

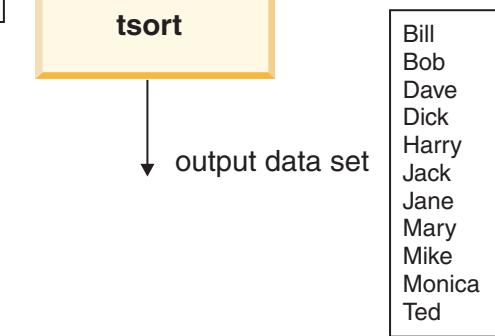
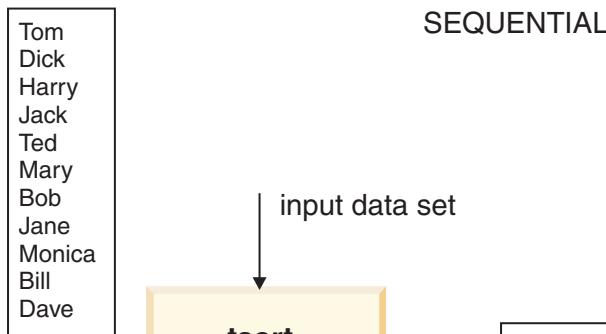
In sequential mode, the entire data set is sorted, but the sort is executed on a single node. In parallel mode, the data is partitioned and sorted on multiple nodes, but the resulting data set is sorted only within each partition. This is adequate for many tasks, such as removing duplicate records, if an appropriate partitioning method is used.

The tsort operator

WebSphere DataStage provides the sort operator, tsort that you can use to sort the records of a data set. The tsort operator can run as either a sequential or a parallel operator. The execution mode of the tsort operator determines its action:

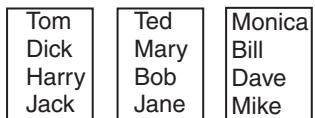
- Sequential mode: The tsort operator executes on a single processing node to sort an entire data set. On completion, the records of the data set are sorted completely.
- Parallel mode: The tsort operator executes on multiple processing nodes in your system. On completion, the records within each partition of a data set are sorted. This type of sort is called a partition sort.

The following figure shows the difference between these two operating modes:

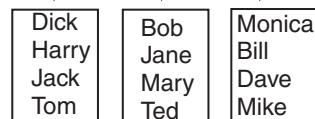


PARALLEL

input data set partitions



tsort



output data set partitions

The tsort operator on the left side of this figure runs sequentially to completely sort the records of a data set. Typically, you use the tsort operator sequentially when you need a total sort of a data set, or when you create a sequential job.

The tsort operator on the right side of this figure executes in parallel and sorts records within each partition. Remember that a parallel operator executes on multiple nodes in a system, where each node receives a partition of a data set. A parallel tsort operator outputs multiple partitions, where the records are sorted within the partition.

Typically, you use a parallel tsort operator as part of a series of operators that requires sorted partitions. For example, you can combine a sort operator with an operator that removes duplicate records from a data set. After the partitions of a data set are sorted, duplicate records in a partition are adjacent.

To perform a parallel sort, you insert a partitioner in front of the tsort operator. This lets you control how the records of a data set are partitioned before the sort. For example, you could hash partition records by a name field, so that all records whose name field begins with the same one-, two-, or three-letter sequence are assigned to the same partition. See "Example: Using a Parallel tsort Operator" for more information.

If you combine a parallel sort with a sort merge collector, you can perform a total sort. A totally sorted data set output is completely ordered, meaning the records in each partition of the output data set are ordered, and the partitions themselves are ordered. See "Performing a Total Sort" for more information.

Configuring the tsort operator

The tsort operator uses temporary disk space when performing a sort. The operator looks in the following locations, in the following order, for this temporary space:

1. Scratch disks in the disk pool sort.

You can create these pools by editing the WebSphere DataStage configuration file.

2. Scratch disks in the default disk pool.

All scratch disks are normally included in the default disk pool.

3. The directory specified by the TMPDIR environment variable.

4. The /tmp directory.

Using a sorted data set

You might perform a sort for several reasons. For example, you might want to sort a data set by a zip code field, then by last name within the zip code. Once you have sorted the data set, you can filter the data set by comparing adjacent records and removing any duplicates.

However, you must be careful when processing a sorted data set: many types of processing, such as repartitioning, can destroy the sort order of the data. For example, assume you sort a data set on a system with four processing nodes and store the results to a persistent data set. The data set will therefore have four partitions. You then use that data set as input to an operator executing on a different number of nodes, possibly due to node constraints.

WebSphere DataStage automatically repartitions a data set to spread out the data set to all nodes in the system, possibly destroying the sort order of the data. To prevent an accidental repartitioning of the sorted output data set, the tsort operator sets the preserve-partitioning flag in its output data set. If set, the preserve-partitioning flag prevents an WebSphere DataStage operator using a partitioning method of any from repartitioning the data set.

An operator that takes a sorted data set as input could also use the partitioning method same. An operator using this partitioning method does not perform any repartitioning as it reads the input data set; that is, the partitions of an input data set are unchanged by the processing nodes executing the operator.

Using a sorted data set with a sequential operator

You must also be careful when using a sequential operator to process a sorted data set. A sequential operator executes on a single processing node to perform its action. Sequential operators always repartition a data set when the data set has more than one partition; therefore, a sequential operator might also destroy the sorting order of its input data set.

If the input to tsort was partitioned using a range partitioner, you can use a partition-sorted data set as input to a sequential operator using an ordered collector to preserve the sort order of the data set. Using this collection method causes all the records from the first partition of a data set to be read first, then all records from the second partition, and so on.

If the input to tsort was partitioned using a hash partitioner, you can use a partition-sorted data set as input to a sequential operator using a sortmerge collector with the keys. Using this collection method causes the records to be read in sort order.

Passing a sorted data set to an RDBMS: general information

WebSphere DataStage allows you to read RDBMS data into a data set and to write a data set to an RDBMS table. Note that an RDBMS does not guarantee deterministic ordering behavior unless an SQL operation constrains it to do so. For example, if you write a sorted data set (using tsort or psort) to DB2, then read the data set back, the records of the data set are not guaranteed to be in sorted order. Also, if you read the same DB2 table multiple times, DB2 does not guarantee to deliver the records in the same order every time.

See the topic on WebSphere DataStage's interface to your particular RDBMS for more information.

Specifying sorting keys

Sorting keys specify the criteria used to perform the sort. The tsort operator allows you to set a primary sorting key and multiple secondary sorting keys.

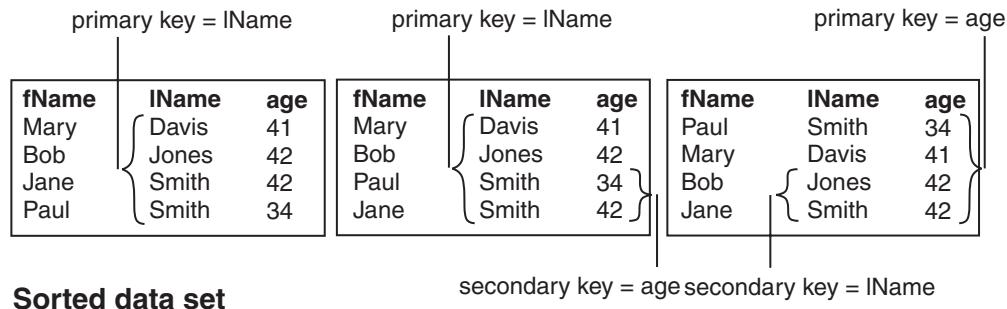
The tsort operator uses the sorting keys to determine the sorting order of a data set. The sort operator first sorts the records by the primary sorting key. If multiple records have the same primary key value, the tsort operator then sorts these records by any secondary keys. By default, if all the keys are identical in two records their sort order is undefined. Specifying the -stable option to tsort guarantees the records are output in the same order in which they were received, at a slight loss in speed.

You must define a single primary sorting key for the tsort operator. You might optionally define as many secondary keys as required by your job. Note, however, that each record field can be used only once as a sorting key. Therefore, the total number of primary and secondary sorting keys must be less than or equal to the total number of fields in the record.

The following figure shows four records whose schema contains three fields:

Unsorted data set

	fName	IName	age
unsorted data set	Jane	Smith	42
	Paul	Smith	34
	Mary	Davis	41
	Bob	Jones	42

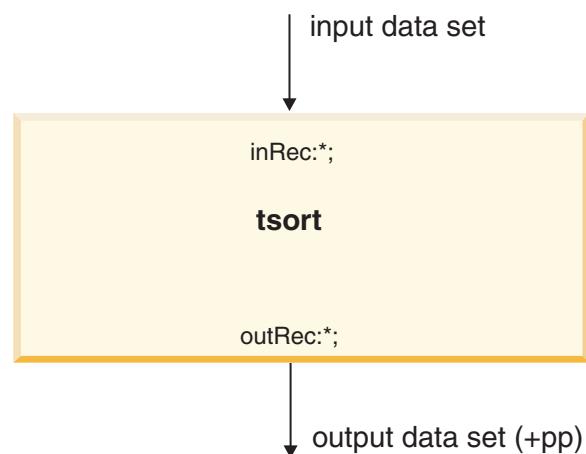


This figure also shows the results of three sorts using different combinations of sorting keys. In this figure, the fName and lName fields are string fields and the age field represents an integer.

Here are tsort defaults and optional settings:

- By default, the tsort operator APT_TSortOperator uses a case-sensitive algorithm for sorting. This means that uppercase strings appear before lowercase strings in a sorted data set. You can use an option to the tsort operator to select case-insensitive sorting. You can use the member function APT_TSortOperator::setKey() to override this default, to perform case-insensitive sorting on string fields.
- By default, the tsort operator APT_TSortOperator uses ascending sort order, so that smaller values appear before larger values in the sorted data set. You can use an option to the tsort operator to perform descending sorting. You can use APT_TSortOperator::setKey() to specify descending sorting order as well, so that larger values appear before smaller values in the sorted data set.
- By default, the tsort operator APT_TSortOperator uses ASCII. You can also set an option to tsort to specify EBCDIC.
- By default, nulls sort low. You can set tsort to specify sorting nulls high.

Data flow diagram



The tsort operator:

- Takes a single data set as input.
- Writes its results to a single output data set and sets the preserve-partitioning flag in the output data set.
- Has a dynamic input interface schema that allows you to specify as many input key fields as are required by your sort operation.
- Has an input interface schema containing the schema variable sortRec and an output interface schema consisting of a single schema variable sortRec.
- Does not add any new fields to the sorted records.
- Transfers all key and non-key fields to the output.

tsort: properties

Table 118. tsort Operator Properties

Property	Value
Number of input data sets	1

Table 118. *tsort Operator Properties (continued)*

Property	Value
Number of output data sets	1
Input interface schema	sortRec:*; The dynamic input interface schema lets you specify as many input key fields as are required by your sort operation.
Output interface schema	sortRec:*
Transfer behavior	sortRec -> sortRec without record modification
Execution mode	parallel (default) or sequential
Partitioning method	parallel mode: any
Collection method	sequential mode: any
Preserve-partitioning flag in output data set	set
Composite operator	no
Combinable operator	yes

Tsort: syntax and options

The syntax for the tsort operator in an osh command is shown below:

```
tsort
  -key field [ci | cs] [-ebcdic] [-nulls first | last] [-asc | -desc]
    [-sorted | -clustered] [-param params ]
    [-key field [ci | cs] [-ebcdic] [-nulls first | last] [-asc | -desc]
      [-sorted | -clustered] [-param params ] ...]
    [-collation_sequence locale | collation_file_pathname | OFF]
    [-flagKey]
    [-flagCuster]
    [-memory num_megabytes ]
    [-stable | -nonstable]
    [-stats]
    [-unique]
```

You must use -key to specify at least one sorting key to the operator.

Table 119. tsort Operator Options

Option	Use
-key	<p>-key <i>field</i> [ci cs] [-ebcdic] [-nulls first last] [-asc -desc] [-sorted -clustered] [-param <i>params</i>]</p> <p>Specifies a key field for the sort. The first -key defines the primary key field for the sort; lower-priority key fields are supplied on subsequent key specifications.</p> <p>-key requires that field be a field of the input data set.</p> <p>-ci -cs are optional arguments for specifying case-sensitive or case insensitive sorting. By default, the operator does case-sensitive sorting. This means that uppercase strings appear before lowercase strings in a sorted data set. You can override this default to perform case-insensitive sorting on string fields only.</p> <p>-asc -desc are optional arguments for specifying ascending or descending sorting. By default, the operator uses ascending sort order, so that smaller values appear before larger values in the sorted data set. You can specify descending sorting order instead, so that larger values appear before smaller values in the sorted data set.</p> <p>-ebcdic (string fields only) specifies to use EBCDIC collating sequence for string fields. Note that WebSphere DataStage stores strings as ASCII text; this property only controls the collating sequence of the string.</p> <p>For example, using the EBCDIC collating sequence, lowercase letters sort before uppercase letters (unless you specify the -ci option to select case-insensitive sorting). Also, the digits 0-9 sort after alphabetic characters. In the default ASCII collating sequence used by the operator, numbers come first, followed by uppercase, then lowercase letters.</p> <p>-sorted specifies that input records are already sorted by this field. The operator then sorts on secondary key fields, if any. This option can increase the speed of the sort and reduce the amount of temporary disk space when your records are already sorted by the primary key field(s) because you only need to sort your data on the secondary key field(s).</p> <p>-sorted is mutually exclusive with -clustered; if any sorting key specifies -sorted, no key can specify -clustered.</p> <p><i>continued</i></p>

Table 119. *tsort Operator Options (continued)*

Option	Use
- key (continued)	<p>If you specify -sorted for all sorting key fields, the operator verifies that the input data set is correctly sorted, but does not perform any sorting. If the input data set is not correctly sorted by the specified keys, the operator fails.</p> <p>-clustered specifies that input records are already grouped by this field, but not sorted. The operator then sorts on any secondary key fields. This option is useful when your records are already grouped by the primary key field(s), but not necessarily sorted, and you want to sort your data only on the secondary key field(s) within each group.</p> <p>-clustered is mutually exclusive with -sorted; if any sorting key specifies -clustered, no key can specify -sorted.</p> <p>-nulls specifies whether null values should be sorted first or last. The default is first.</p> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property =value</i> pairs separated by commas.</p>
- collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> Specify a predefined IBM ICU locale Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.htm</p>
- flagCluster	<p>-flagCluster</p> <p>Tells the operator to create the int8 field <i>clusterKeyChange</i> in each output record. The <i>clusterKeyChange</i> field is set to 1 for the first record in each group where groups are defined by the -sorted or -clustered argument to -key. Subsequent records in the group have the <i>clusterKeyChange</i> field set to 0.</p> <p>You must specify at least one sorting key field that uses either -sorted or -clustered to use -flagCluster, otherwise the operator ignores this option.</p>

Table 119. *tsort Operator Options (continued)*

Option	Use
-flagKey	Optionally specify whether to generate a flag field that identifies the key-value changes in output.
-memory	<p>-memory <i>num_megabytes</i></p> <p>Causes the operator to restrict itself to <i>num_megabytes</i> megabytes of virtual memory on a processing node.</p> <p>-memory requires that $1 < \text{num_megabytes} <$ the amount of virtual memory available on any processing node. We recommend that <i>num_megabytes</i> be smaller than the amount of physical memory on a processing node.</p>
-stable	<p>-stable</p> <p>Specifies that this sort is stable. A stable sort guarantees not to rearrange records that are already sorted properly in a data set.</p> <p>The default sorting method is unstable. In an unstable sort, no prior ordering of records is guaranteed to be preserved by the sorting operation.</p>
-stats	<p>-stats</p> <p>Configures tsort to generate output statistics about the sorting operation.</p>
-unique	<p>-unique</p> <p>Specifies that if multiple records have identical sorting key values, only one record is retained. If -stable is set, the first record is retained.</p>

Example: using a sequential tsort operator

This section contains an example using a sequential tsort operator, as shown in the following figure:

input data set

schema:

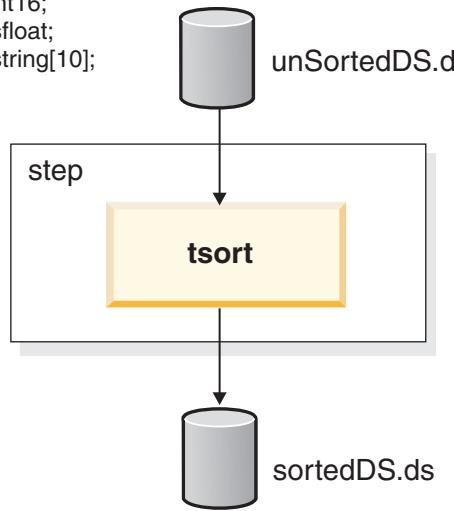
a:int32;

b:int32;

c:int16;

d:sfloat;

e:string[10];



This step uses a sequential tsort operator to completely sort the records of an input data set. The primary key field is a, the secondary sorting key field is e.

Since the tsort operator runs by default in parallel, you use the [seq] framework argument to configure the operator to run sequentially.

Shown below is the osh command line for this step:

```
$ osh "tsort -key a -key e -stable [seq] < unSortedDS.ds  
      > sortedDS.ds "
```

Example: using a parallel tsort operator

A parallel tsort operator runs on multiple processing nodes in your system to sort the records within each partition of a data set. The default execution mode of the operator is parallel, using the any partitioning method. However, you typically use a partitioning operator with the tsort operator to set an explicit partitioning method.

Choose a partitioning method that is correct for the sorting operation. For example, assume that you are sorting records in a data set based on the last name field of each record. If you randomly allocate records into any partition, records with similar last names are not guaranteed to be in the same partition and are not, therefore, processed by the same node. Similar records can be sorted by an operator only if they are in the same partition of the data set.

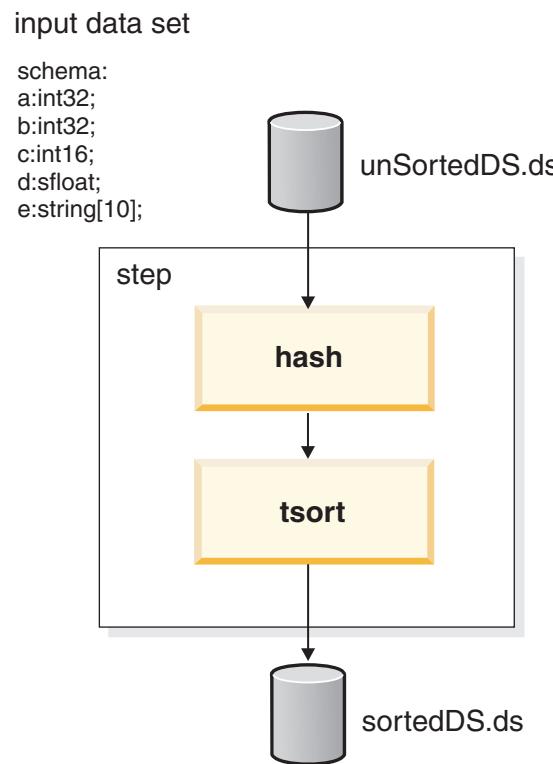
A better method of partitioning data in this case is to hash the records by the first five or six characters of the last name. All records containing similar names would be in the same partition and, therefore, would be processed by the same node. The tsort operator could then compare the entire last names, first names, addresses, or other information in the records, to determine the sorting order of the records.

For example:

```
record ( fname:string[30]; lname:string[30]; )  
... | modify -spec "lname_hash:string[6] = substring[0,6](lname)"  
    | hash -key lname_hash | tsort -key lname | ...
```

WebSphere DataStage supplies a hash partitioner operator that allows you to hash records by one or more fields. See "The hash Partitioner" for more information on the hash operator. You can also use any one of the supplied WebSphere DataStage partitioning methods.

The following example is a modification of the previous example, "Example: Using a Sequential tsort Operator", to execute the tsort operator in parallel using a hash partitioner operator. In this example, the hash operator partitions records using the integer field a, the primary sorting key. Thus all records containing the same value for field a will be assigned to the same partition. The figure below shows this example:



Shown below is the osh command corresponding to this example:

```
$ osh " hash -key a -key e < unSortedDS.ds |
      tsort -key a -key e > sortedDS.ds"
```

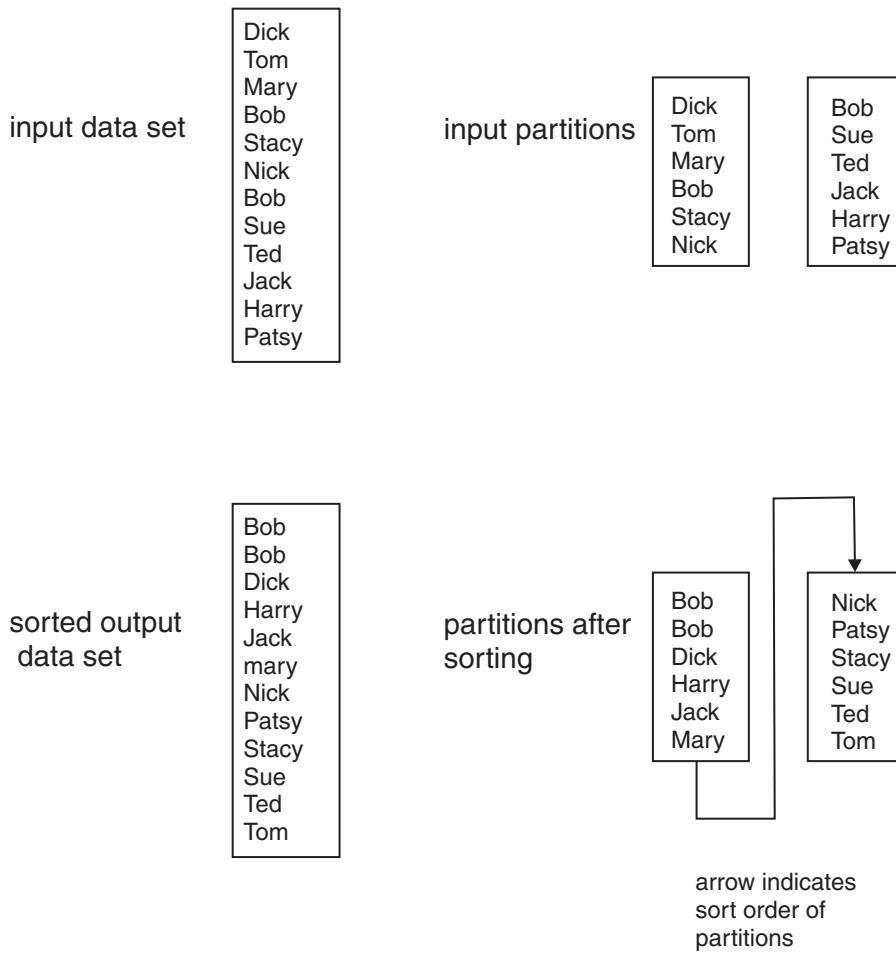
Performing a total sort

The previous section showed an example of a parallel sorter using the hash partitioner to sort the partitions of a data set. When using a hash partitioner, all records containing the same key values would be in the same partition and therefore are processed and sorted by the same node. The partition-sorted data set can be collected into a totally sorted sequential data set using a sort merge collector with the same keys.

The osh command for this is:

```
$ osh "hash -key a -key e < unsortedDS.ds | tsort -key a -key e | sortmerge -key a -key e > sorted.ds"
```

In contrast to a partition sort, you can also use the tsort operator to perform a total sort on a parallel data set. A totally sorted parallel data set means the records in each partition of the data set are ordered, and the partitions themselves are ordered.



The following figure shows a total sort performed on a data set with two partitions:

In this example, the partitions of the output data set contain sorted records, and the partitions themselves are sorted. A total sort requires that all similar and duplicate records be located in the same partition of the data set. Similarity is based on the key fields in a record.

Because each partition of a data set is sorted by a single processing node, another requirement must also be met before a total sort can occur. Not only must similar records be in the same partition, but the partitions themselves should be approximately equal in size so that no one node becomes a processing bottleneck.

To meet these two requirements, use the range partitioner on the input data set before performing the actual sort. The range partitioner guarantees that all records with the same key fields are in the same partition, and calculates partition boundaries based on the key fields, in order to evenly distribute records across all partitions. See the next section for more information on using the range partitioner.

You need to perform a total sort only when your job requires a completely ordered data set, either as output or as a preprocessing step. For example, you might want to perform a total sort on a mailing list so that the list is sorted by zip code, then by last name within zip code.

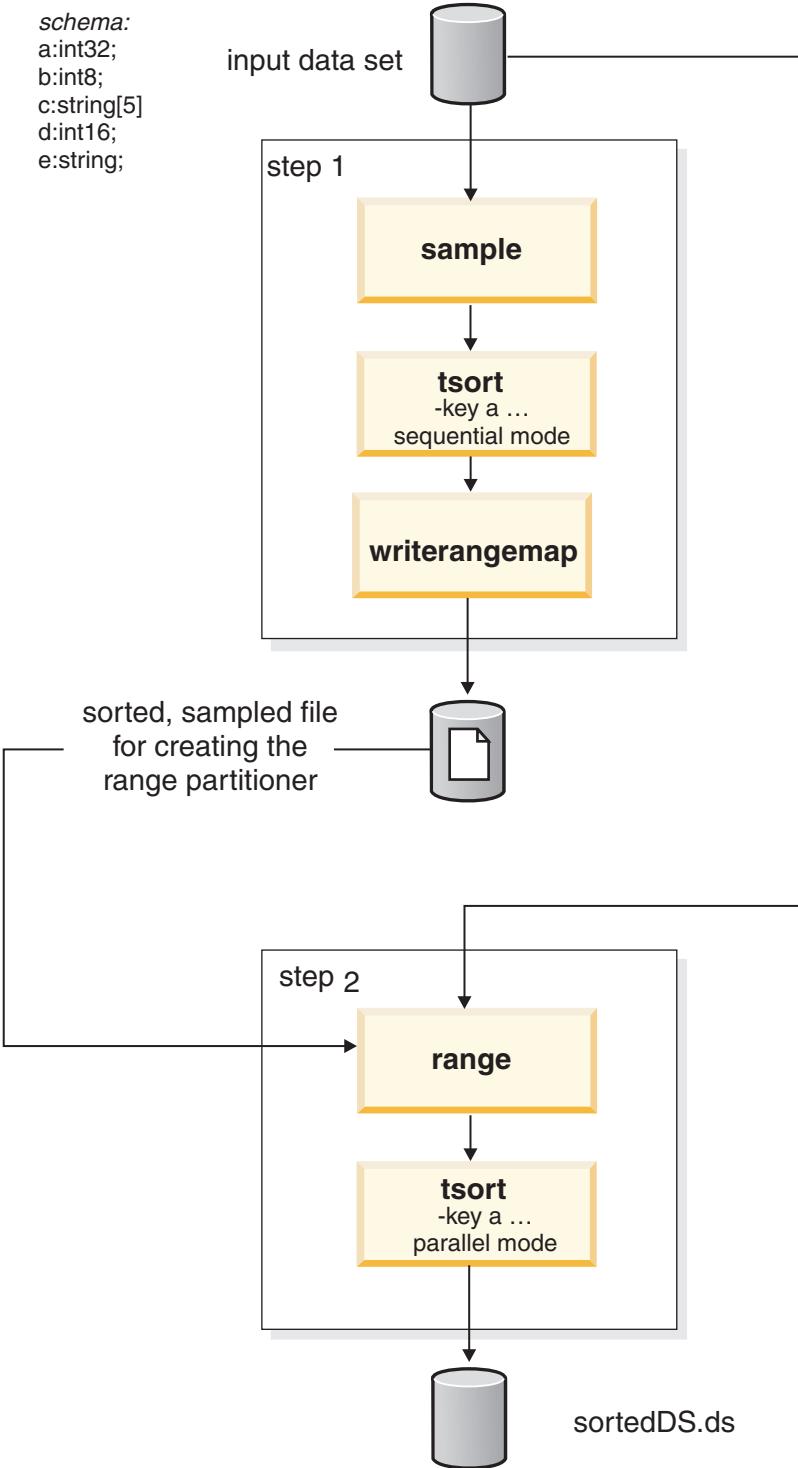
For most jobs, a partition sort is the correct sorting approach. For example, if you are sorting a data set as a preprocessing step to remove duplicate records, you use a hash partitioner with a partition sort to partition the data set by the key fields; a total sort is unnecessary. A hash partitioner assigns records with the same key fields to the same partition, then sorts the partition. A second operator

can compare adjacent records in a partition to determine if they have the same key fields to remove any duplicates.

Example: performing a total sort

This section contains an example that uses the range partitioner and the tsort operator to perform a total sort on a parallel data set. The following page shows the data flow for this example.

The figure shows the data flow for this example:



As the figure shows, this example uses two steps. The first step creates the sorted sample of the input data set used to create the range map required by the range partitioner. The second step performs the parallel sort using the range partitioner.

WebSphere DataStage supplies the UNIX command line utility, makerangemap, that performs the first of these two steps for you to create the sorted, sampled file, or range map. See "The range Partitioner" for more information on makerangemap.

Shown below are the commands for the two steps shown above using makerangemap:

```
$ makerangemap -rangemap sampledData -key a -key e inDS.ds  
$ osh "range -sample sampledData -key a -key c < inDS.ds |  
      tsort -key a -key e > sortDS.ds"
```

The psort operator

Many data mining applications require the records of a data set to be sorted. The sorting operation is usually based on one or more fields of the records in the data set. For example, you might want to sort a data set by a zip code field, then by last name within the zip code.

The partition sort operator, psort, sorts the records of a data set. The execution mode of the operator, parallel or sequential, determines how the operator sorts the data set. The first section of this topic describes these different execution modes.

Fields used to sort records are referred to as key fields of the sort. In the example mentioned above, the zip code field of each record would be the primary sorting key, and the last name field would be the secondary sorting key. See "Specifying Sorting Keys" for more information on sorting keys.

The partition sort operator uses the UNIX sort command to perform the actual sorting operation. See "Configuring the Partition Sort Operator" for information on configuring the operator.

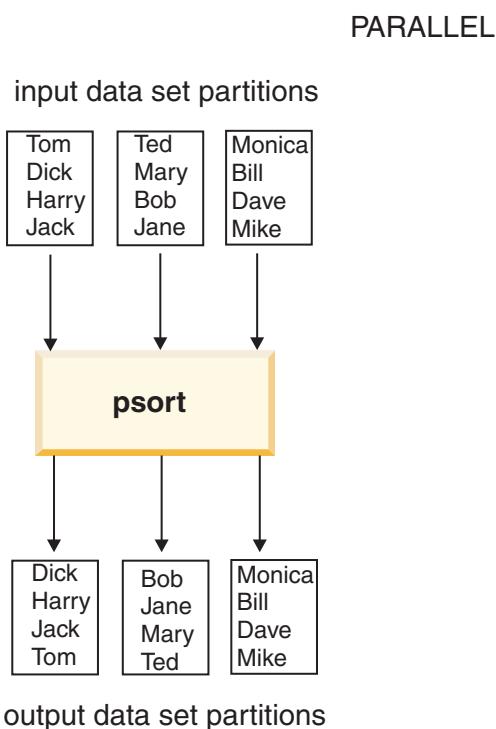
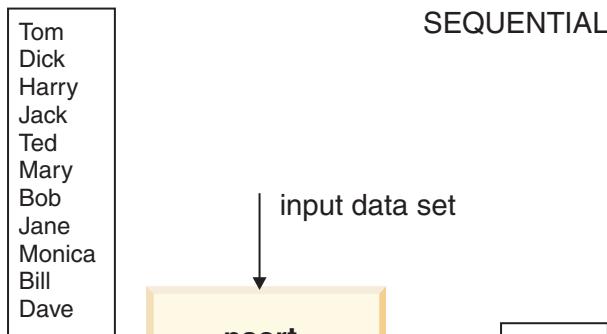
WebSphere DataStage also provides the sorting operator tsort. It is described in "The tsort Operator" .

Performing a partition sort

WebSphere DataStage includes the partition sort operator, psort, which you can use to sort the records of a data set. The psort operator can execute as either a sequential or a parallel operator. The execution mode of the psort operator determines its action:

- Sequential mode: The psort operator executes on a single processing node to sort an entire data set. On completion, the records of the data set are sorted completely.
- Parallel mode: The psort operator executes on multiple processing nodes in your system. On completion, the records within each partition of the data set are sorted.

The following figure shows the difference between these two operating modes:



The psort operator on the left side of this figure runs sequentially to completely sort the records of a data set. Typically, you use the psort operator sequentially when you require a total sort of a data set, or when you create a sequential job.

The psort operator on the right side of this figure runs in parallel and sorts records within a partition. Remember that a parallel operator executes on multiple nodes in a system, where each node receives a partition of a data set. A parallel psort operator outputs partitions, where the records are sorted within the partition.

Typically you use a parallel psort operator as part of a series of operators that requires sorted partitions. For example, you can combine a sort operator with an operator that removes duplicate records from a data set. After the partitions of a data set are sorted, duplicate records are adjacent in a partition.

In order to perform a parallel sort, you specify a partitioning method to the operator, enabling you to control how the records of a data set are partitioned before the sort. For example, you could partition records by a name field, so that all records whose name field begins with the same one-, two-, or three-letter sequence are assigned to the same partition. See "Example: Using a Parallel Partition Sort Operator" for more information.

If you combine a parallel sort with a range partitioner, you can perform a total sort. A totally sorted data set output is completely ordered. See "Performing a Total Sort" for more information.

The example shown above for a parallel psort operator illustrates an input data set and an output data set containing the same number of partitions. However, the number of partitions in the output data set is determined by the number of processing nodes in your system configured to run the psort operator. The psort operator repartitions the input data set such that the output data set has the same number of partitions as the number of processing nodes. See the next section for information on configuring the psort operator.

Configuring the partition sort operator

Several UNIX sorting commands, including the UNIX sort command, exist for single-processor workstations. The psort operator APT_PartitionSortOperator can use the UNIX sort command to sort the records of a data set. The psort operator APT_PartitionSortOperator can sort either in a parallel or a sequential mode.

Records processed by the psort operator APT_PartitionSortOperator must be less than 32 KB in length. If the psort operator APT_PartitionSortOperator encounters a record longer than 32 KB, the operator aborts execution. The step containing the operator fails, and an error message is written to the error log.

Though it is not required for normal execution of the psort operator APT_PartitionSortOperator, you might want to create node pools and resources for use by the psort operator APT_PartitionSortOperator.

Using a sorted data set

You perform a partition sort for several reasons. For example, you might want to sort a data set by a zip code field, then by last name within the zip code. Once you have sorted the data set, you can filter the data set by comparing adjacent records and removing any duplicates.

However, you must be careful when processing a sorted data set: many types of processing, such as repartitioning, can destroy the sorting order of the data. For example, assume you sort a data set on a system with four processing nodes and store the results to a persistent data set. The data set will therefore have four partitions. You then use that data set as input to an operator executing on a different number of nodes. This might be because your system does not have a SyncSort license for every node, or because of a node failure.

WebSphere DataStage automatically repartitions a data set to spread out the data set to all nodes in the system, possibly destroying the sort order of the data. To prevent an accidental repartitioning of the sorted output data set, the psort operator sets the preserve-partitioning flag in its output data set. If set, the preserve-partitioning flag prohibits an WebSphere DataStage operator using a partitioning method of any from repartitioning the data set.

An operator that takes a sorted data set as input could also use the partitioning method same. An operator using this partitioning method does not perform any repartitioning as it reads the input data set; that is, the partitions of an input data set are unchanged by the processing nodes executing the operator.

Using a Sorted Data Set with a Sequential Operator

You must also be careful when using a sequential operator to process a sorted data set. A sequential operator executes on a single processing node to perform its action. Sequential operators always repartition a data set when the data set has more than one partition; therefore, a sequential operator might also destroy the sorting order of its input data set.

You can use a partition-sorted data set as input to a sequential operator using the collection method sortmerge with the appropriate keys to preserve the sorting order of the data set. A sequential operator using this collection method reads all records from the first partition of a data set, then all records from the second partition, and so on. .

Passing a Sorted Data Set to an RDBMS: General Information

WebSphere DataStage allows you to read RDBMS data into a data set and to write a data set to an RDBMS table. Note that an RDBMS does not guarantee deterministic ordering behavior unless the SQL operation used to retrieve the records constrains it to do so. For example, if you write a data set (using psort or tsort) to DB2, then read the data set back, the records of the data set are not guaranteed to be in the original order. Also, if you read the same DB2 table multiple times, DB2 does not guarantee to deliver the records in the same order every time.

See the topic on your particular RDBMS for more information.

Specifying sorting keys

Sorting keys specify the criteria used to perform the sort. The psort operator allows you to set a primary sorting key and multiple secondary sorting keys.

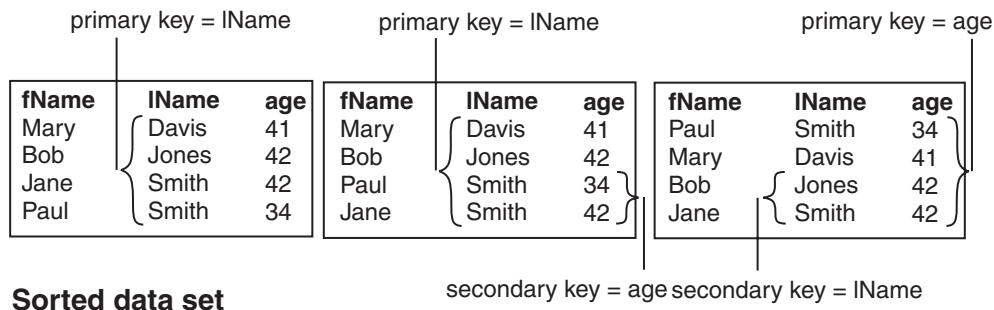
The psort operator uses the sorting keys to determine the sorting order of a data set. The sort operator first sorts the records by the primary sorting key. If multiple records have the same primary key value, the psort operator then sorts these records by any secondary keys.

You must define a single primary sorting key for the psort operator. You might optionally define as many secondary keys as required by your job. Note, however, that each record field can be used only once as a sorting key. Therefore, the total number of primary and secondary sorting keys must be less than or equal to the total number of fields in the record.

The following figure shows four records whose schema contains three fields:

Unsorted data set

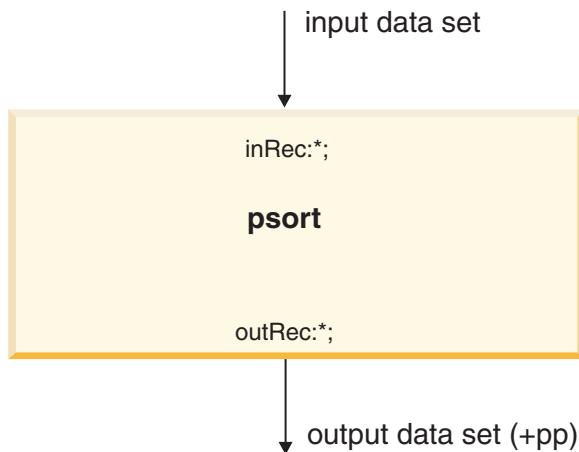
	fName	IName	age
Jane		Smith	42
Paul		Smith	34
Mary		Davis	41
Bob		Jones	42



This figure also shows the results of three sorts using different combinations of sorting keys. In this figure, the lName field represents a string field and the age field represents an integer. By default, the psort operator uses a case-sensitive algorithm for sorting. This means that uppercase strings appear before lowercase strings in a sorted data set. You can use an option to the psort operator to select case-insensitive sorting. You can use the member function APT_PartitionSortOperator::setKey() to override this default, to perform case-insensitive sorting on string fields.

By default, the psort operator APT_PartitionSortOperator uses ascending sort order, so that smaller values appear before larger values in the sorted data set. You can use an option to the psort operator to select descending sorting.

Data Flow Diagram



The psort operator:

- Takes a single data set as input.
- Writes its results to a single output data set and sets the preserve-partitioning flag in the output data set.
- Has a dynamic input interface schema that allows you to specify as many input key fields as are required by your sort operation.
- Has an input interface schema containing the schema variable inRec and an output interface schema consisting of a single schema variable outRec.
- Does not add any new fields to the sorted records.

psort: properties

Table 120. psort Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	inRec:*. The dynamic input interface schema lets you specify as many input key fields as are required by your sort operation.
Output interface schema	outRec:*
Transfer behavior	inRec -> outRec without record modification
Execution mode	sequential or parallel

Table 120. *psort Operator Properties (continued)*

Property	Value
Partitioning method	parallel mode: any
Collection method	sequential mode: any
Preserve-partitioning flag in output data set	set
Composite operator	no

Psort: syntax and options

The syntax for the psort operator in an osh command is shown below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose the value in single or double quotes.

```
psort
  [-key field_name [-ci | -cs] [-asc | -desc] [-ebcdic]
   [-key field_name [-ci | -cs] [-asc | -desc] [-ebcdic] ...]
   [-extraOpts syncsort_options ]
   [-memory num_megabytes ]
   [-sorter unix | syncsort]
   [-stable]
   [-stats]
   [-unique]
   [-workspace workspace]
```

You must use -key to specify at least one sorting key to the operator. You use the -part option to configure the operator to run in parallel, and the -seq option to specify sequential operation.

If you include the -ebcdic option, you must also include the -sorter option with a value of syncsort. When you do not include the -sorter option, the default sort is unix which is incompatible with the EBCDIC collation sequence.

Example usage:

```
psort -sorter syncsort -key a -ebcdic
```

Table 121. *psort Operator Options*

Option	Use
<p>-key</p>	<p>-key <i>field_name</i> [-ci -cs] [-asc -desc] [-ebcdic]</p> <p>If the -ebcdic suboption is specified, you must also include the -sorter option with a value of syncsort.</p> <p>The -key option specifies a key field for the sort. The first -key option defines the primary key field for the sort; lower-priority key fields are supplied on subsequent -key specifications.</p> <p>You must specify this option to psort at least once.</p> <p>-key requires that <i>field_name</i> be a field of the input data set. The data type of the field must be one of the following data types:</p> <ul style="list-style-type: none"> int8 , int16 , int32 , int64, uint8, uint16, uint32 , uint64 sfloat, dfloat <p>string[<i>n</i>] , where <i>n</i> is an integer literal specifying the string length</p> <p>-ci or -cs are optional arguments for specifying case-sensitive or case-insensitive sorting. By default, the operator uses a case-sensitive algorithm for sorting. This means that uppercase strings appear before lowercase strings in a sorted data set. You can override this default to perform case-insensitive sorting on string fields.</p> <p>-asc or -desc specify optional arguments for specifying ascending or descending sorting. By default, the operator uses ascending sort order, so that smaller values appear before larger values in the sorted data set. You can use descending sorting order as well, so that larger values appear before smaller values in the sorted data set.</p> <p>-ebcdic (string fields only) specifies to use EBCDIC collating sequence for string fields. Note that WebSphere DataStage stores strings as ASCII text; this property only controls the collating sequence of the string.</p> <p>If you include the -ebcdic option, you must also include the -sorter option with a value of syncsort. When you do not include the -sorter, the default sort is unix which is incompatible with the EBCDIC collation sequence.</p> <p>When you use the EBCDIC collating sequence, lowercase letters sort before upper-case letters (unless you specify the -ci option to select case-insensitive sorting). Also, the digits 0-9 sort after alphabetic characters. In the default ASCII collating sequence used by the operator, numbers come first, followed by uppercase, then lowercase letters.</p>
<p>-extraOpts</p>	<p>-extraOpts <i>syncsort_options</i></p> <p>Specifies command-line options passed directly to SyncSort. <i>syncsort_options</i> contains a list of SyncSort options just as you would normally type them on the SyncSort command line.</p>

Table 121. *psort Operator Options (continued)*

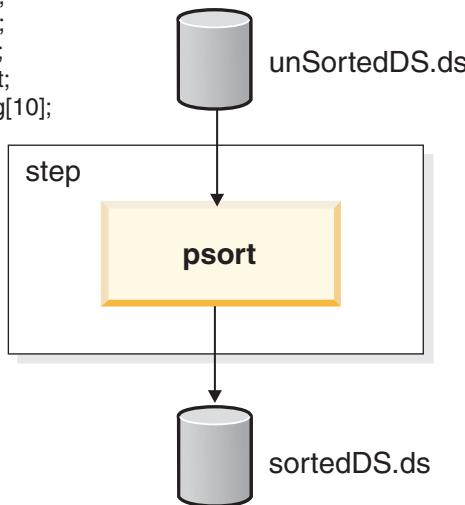
Option	Use
-memory	<p>-memory <i>num_megabytes</i></p> <p>Causes the operator to restrict itself to <i>num_megabytes</i> megabytes of virtual memory on a processing node.</p> <p>-memory requires that $1 < \text{num_megabytes} <$ the amount of virtual memory available on any processing node. We recommend that <i>num_megabytes</i> be smaller than the amount of physical memory on a processing node.</p>
-part	<p>-part <i>partitioner</i></p> <p>This is a deprecated option. It is included for backward compatibility.</p>
-seq	<p>-seq</p> <p>This is a deprecated option. It is included for backward compatibility.</p>
-sorter	<p>-sorter unix syncsort</p> <p>Specifies the sorting utility used by the operator. The default is unix, corresponding to the UNIX sort utility.</p>
-stable	<p>-stable</p> <p>Specifies that this sort is stable. A stable sort guarantees not to rearrange records that are already sorted properly in a data set.</p> <p>The default sorting method is unstable. In an unstable sort, no prior ordering of records is guaranteed to be preserved by the sorting operation, but might processing might be slightly faster.</p>
-stats	<p>-stats</p> <p>Configures psort to generate output statistics about the sorting operation and to print them to the screen.</p>
-unique	<p>-unique</p> <p>Specifies that if multiple records have identical sorting key values, only one record is retained. If stable is set, then the first record is retained.</p>
-workspace	<p>-workspace <i>workspace</i></p> <p>Optionally supply a string indicating the workspace directory to be used by the sorter</p>

Example: using a sequential partition sort operator

This section contains an example using a sequential psort operator, as shown in the following figure:

input data set

schema:
a:int32;
b:int32;
c:int16;
d:sfloat;
e:string[10];



This step uses a sequential psort operator to completely sort the records of an input data set. The primary key field is a; the secondary sorting key field is e.

By default, the psort operator executes sequentially; you do not have to perform any configuration actions.

Note that case-sensitive and ascending sort are the defaults.

Shown below is the osh command line for this step:

```
$ osh "psort -key a -key e -stable [seq] < unSortedDS.ds > sortedDS.ds "
```

Example: using a parallel partition sort operator

A parallel psort operator executes on multiple processing nodes in your system to sort the records within each partition of a data set. To use the psort operator to execute the sort in parallel, you must specify a partitioning method for the operator. It is by specifying the partitioning method that you configure the operator to run in parallel.

Choose a partitioning method that is correct for the sorting operation. For example, assume that you are sorting records in a data set based on the last-name field of each record. If you randomly allocate records into any partition, records with similar last names are not guaranteed to be in the same partition and are not, therefore, processed by the same node. Similar records can be sorted by an operator only if they are in the same partition of the data set.

A better method of partitioning data in this case would be to hash the records by the first five or six characters of the last name. All records containing similar names would be in the same partition and, therefore, would be processed by the same node. The psort operator could then compare the entire last names, first names, addresses, or other information in the records, to determine the sorting order of the records.

For example:

```
record ( fname:string[30]; lname:string[30]; )
... | modify -spec "lname_hash:string[6] = substring[0,6](lname)"
    | hash -key lname_hash
    | tsort -key lname | ...
```

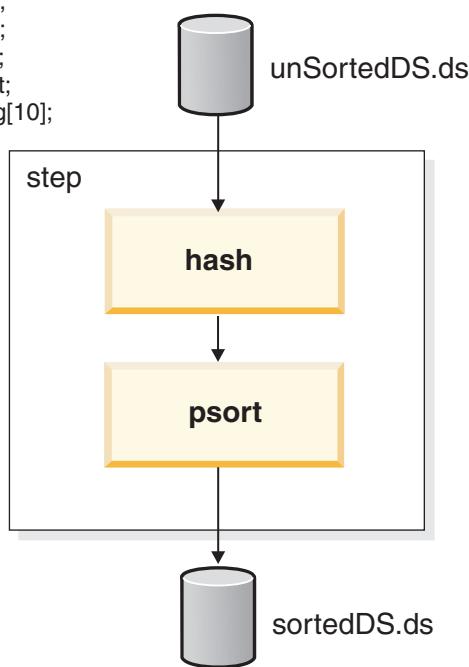
WebSphere DataStage supplies a hash partitioner operator that allows you to hash records by one or more fields. See "The hash Partitioner" for more information on the hash operator. You can also use any one of the supplied WebSphere DataStage partitioning methods.

The following example is a modification of the previous example, "Example: Using a Sequential Partition Sort Operator", to execute the psort operator in parallel using a hash partitioner operator. In this example, the hash operator partitions records using the integer field a, the primary sorting key. Therefore, all records containing the same value for field a are assigned to the same partition. The figure below shows this example:

input data set

schema:

```
a:int32;
b:int32;
c:int16;
d:sfloat;
e:string[10];
```



To configure the psort operator in osh to execute in parallel, use the [par] annotation. Shown below is the osh command line for this step:

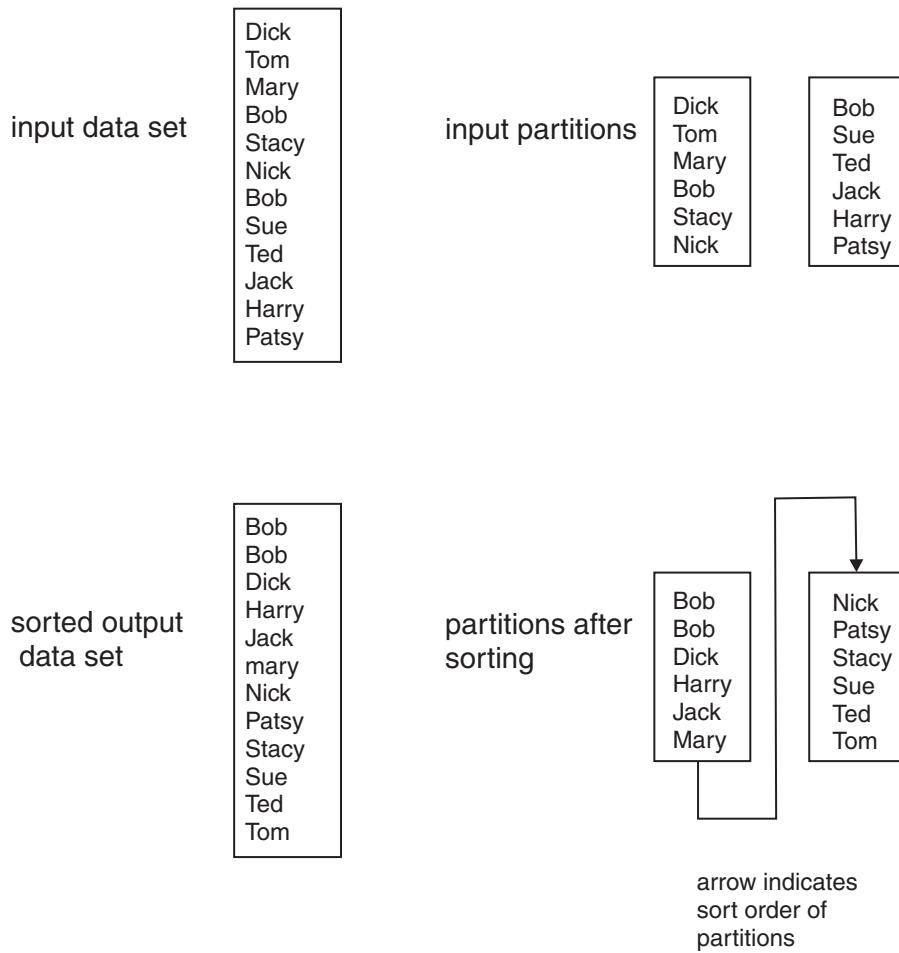
```
$ osh " hash -key a -key e < unSortedDS.ds |
      psort -key a -key e [par] > sortedDS.ds"
```

Performing a total sort

The previous section showed an example of a parallel sorter using the hash partitioner to sort the partitions of a data set. When you use a hash partitioner, all records containing the same hash key values are in the same partition and are therefore processed and sorted by the same node.

In contrast to a partition sort, you can also use the psort operator to perform a total sort.

The following figure shows a total sort performed on a data set with two partitions:



In this example, the partitions of the output data set contain sorted records, and the partitions themselves are sorted. As you can see in this example, a total sort requires that all similar and duplicate records are located in the same partition of the data set. Similarity is based on the key fields in a record.

Because each partition of a data set is sorted by a single processing node, another requirement must also be met before a total sort can occur. Not only must similar records be in the same partition, but the partitions themselves should be approximately equal in size so that no one node becomes a processing bottleneck. To meet these two requirements, you use the range partitioner on the input data set before performing the actual sort.

The range partitioner guarantees that all records with the same key fields are in the same partition, but it does more. The range partitioner also calculates partition boundaries, based on the sorting keys, in order to evenly distribute records to the partitions. All records with sorting keys values between the partition boundaries are assigned to the same partition so that the partitions are ordered and that the partitions are approximately the same size. See the next section for more information on using the range partitioning operator.

You need to perform a total sort only when your job requires a completely ordered data set, either as output or as a preprocessing step. For example, you might want to perform a total sort on a mailing list so that the list is sorted by zip code, then by last name within zip code.

For most jobs, a partition sort is the correct sorting component. For example, if you are sorting a data set as a preprocessing step to remove duplicate records, you use a hash partitioner with a partition sort to partition the data set by the sorting key fields; a total sort is unnecessary. A hash partitioner assigns

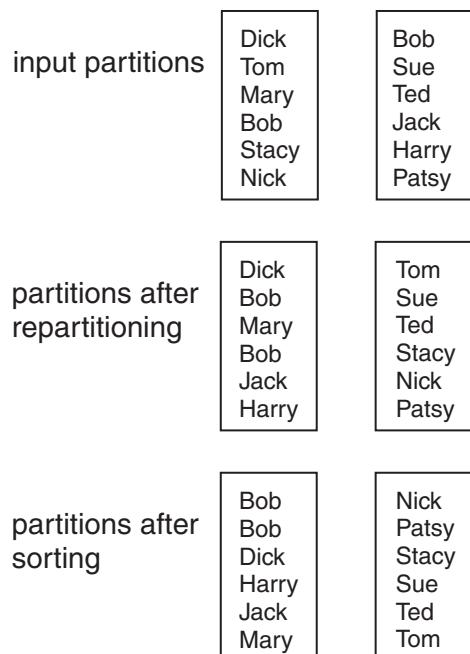
records with the same sorting key fields to the same partition, then sorts the partition. A second operator can compare adjacent records in a partition to determine if they have the same key fields to remove any duplicates.

Range partitioning

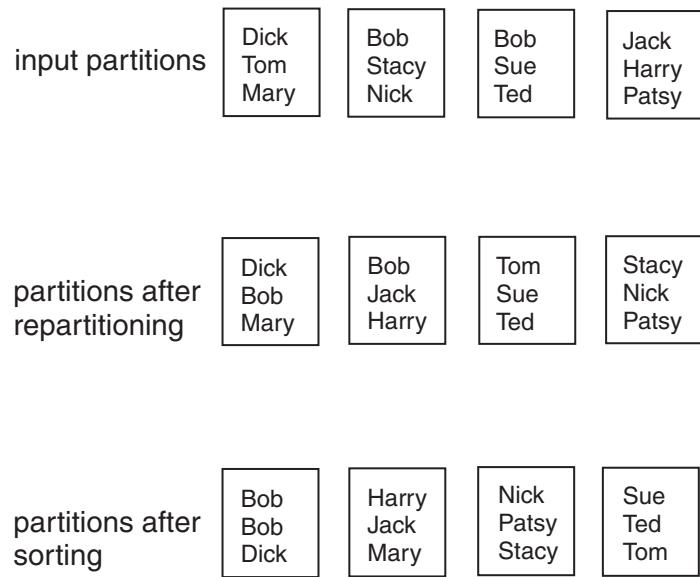
The range partitioner guarantees that all records with the same key field values are in the same partition and it creates partitions that are approximately equal in size so that all processing nodes perform an equal amount of work when performing the sort. In order to do so, the range partitioner must determine distinct partition boundaries and assign records to the correct partition.

To use a range partitioner, you first sample the input data set to determine the distribution of records based on the sorting keys. From this sample, the range partitioner determines the partition boundaries of the data set. The range partitioner then repartitions the entire input data set into approximately equal-sized partitions in which similar records are in the same partition.

The following example shows a data set with two partitions:



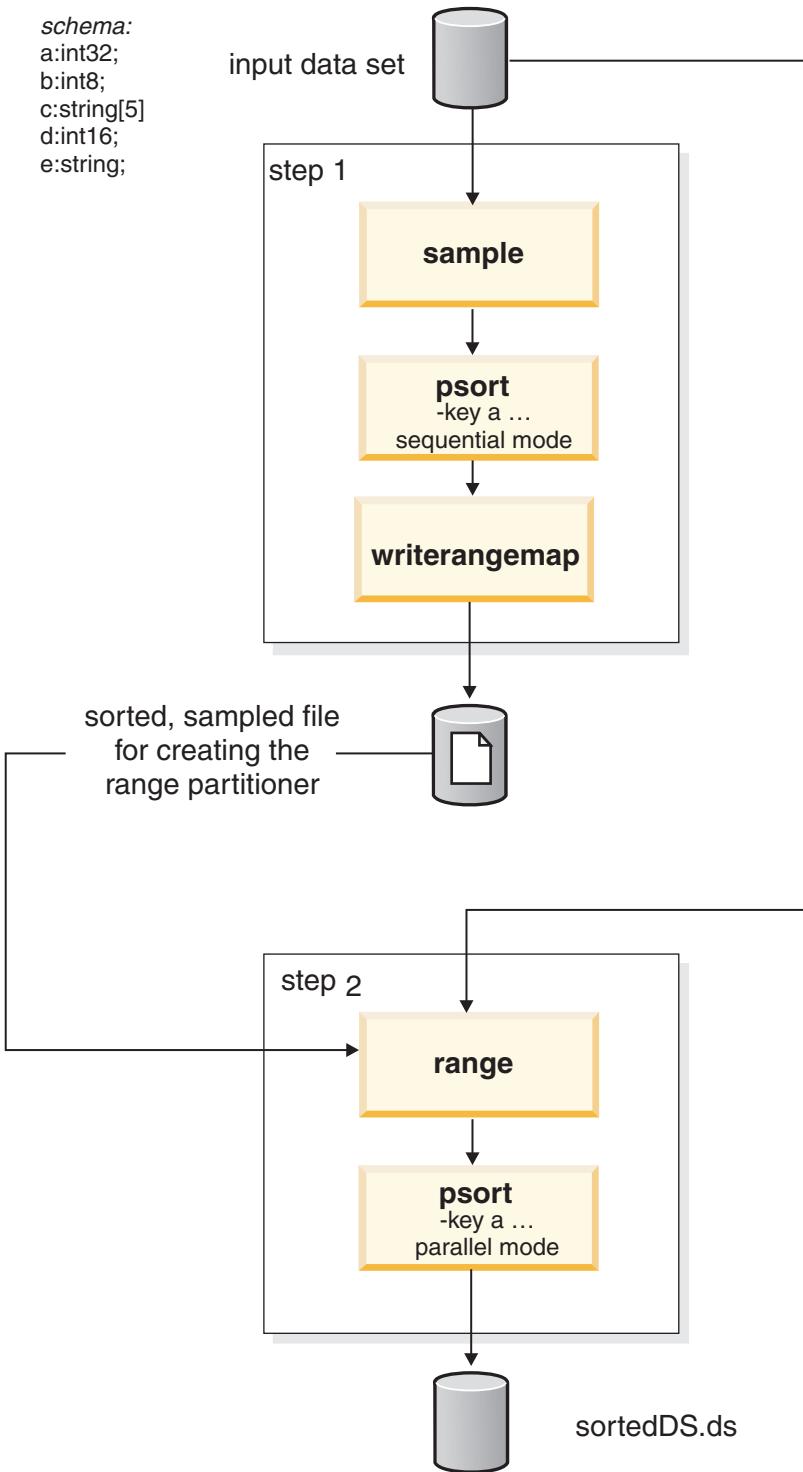
This figure shows range partitioning for an input data set with four partitions and an output data set with four:



In this case, the range partitioner calculated the correct partition size and boundaries, then assigned each record to the correct partition based on the sorting key fields of each record. See "The range Partitioner" for more information on range partitioning.

Example: Performing a Total Sort

This section contains an example that uses the range partitioner and the psort operator to perform a total sort on a data set. The following figure shows the data-flow model for this example:



As you can see in this figure, this example uses two steps. The first step creates the sorted sample of the input data set used to create the range map required by the range partitioner. The second step performs the parallel sort using the range partitioner and the psort operator.

WebSphere DataStage supplies the UNIX command line utility, makerangemap, that performs the first of these two steps for you to create the sorted, sampled file, or range map. See "The range Partitioner" for more information on makerangemap.

Shown below are the commands for the two steps shown above using makerangemap:

```
$ makerangemap -rangemap sampledData -key a -key e inDS.ds  
$ osh "range -sample sampledData -key a -key c < inDS.ds |  
      psort -key a -key e > srtDS.ds"
```

Chapter 13. The join library

The join library contains four operators. The operators let you join data rows together in different ways.

The innerjoin, leftouterjoin, and rightouterjoin operators accept two or more input data sets and perform cascading joins on them. The fullouterjoin operator accepts exactly two data sets. All four operators output a single data set.

In this topic, the first input data set and a data set resulting from an intermediate join are called the left data set and the data set joining them is called the right data set.

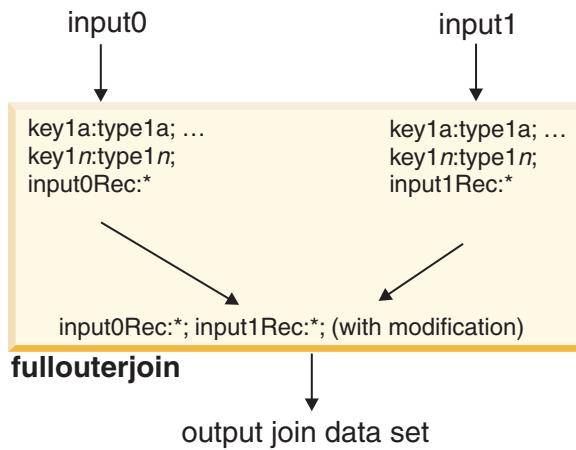
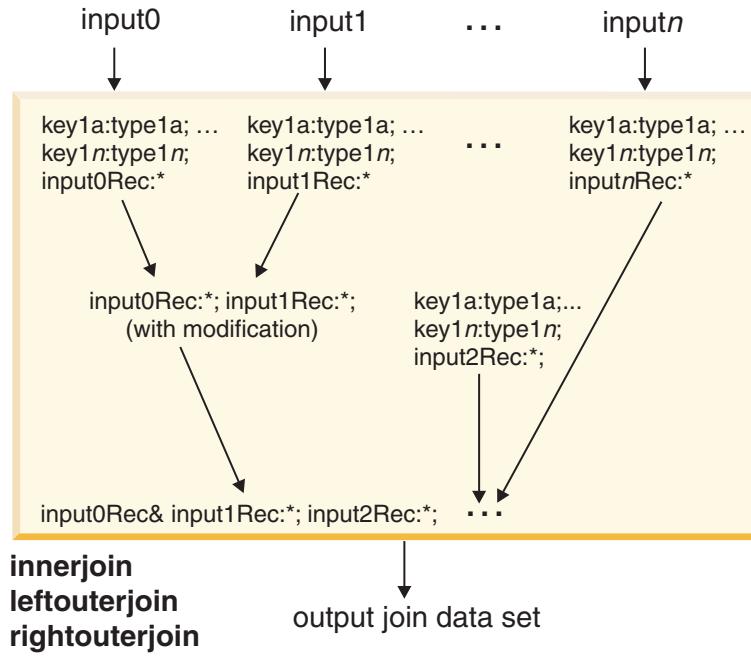
Here is a brief description of each join operator:

- The innerjoin operator outputs the records from two or more input data sets whose key fields contain equal values. Records whose key fields do not contain equal values are dropped. .
- The leftouterjoin operator outputs all values from the left data set and outputs values from the right data set only where key fields match. The operator drops the key field from the right data set. Otherwise, the operator writes default values. .
- The rightouterjoin operator outputs all values from the right data set and outputs values from the left data set only where key fields match. The operator drops the key field from the left data set. Otherwise, the operator writes default values. .
- The fullouterjoin operator transfers records in which the contents of the key fields are equal from both input data sets to the output data set. It also transfers records whose key fields contain unequal values from both input data sets to the output data set.

Data flow diagrams

The innerjoin, leftouterjoin, and rightouterjoin operators accept two or more input data sets and perform cascading joins on them. The data set that results from joining the first and second data sets is joined with the third data set, and the result of that join is joined with the fourth data set, and so on, until the last data set is joined.

The fullouterjoin operator transfers records whose key fields are equal in both input data sets to the output data set. It also transfers records whose key fields contain unequal values from both input data sets to the output data set.



Join: properties

Table 122. Join Operators Properties

Property	Value
Number of input data sets	2 or more for the innerjoin, leftouterjoin, rightouterjoin operators and exactly 2 for the fullouterjoin operator
Number of output data sets	1
Input interface schema:	key1a:type1a; ...; key1n:type1n; input0Rec:*
	key1a:type1a; ...; key1n:type1n; input1Rec:*
Output interface schema	leftRec:*, rightRec:*
Transfer behavior from source to output	leftRec; -> leftRec; rightRec -> rightRec; with modifications
Composite operator	yes
Input partitioning style	keys in same partition

Transfer behavior

The join operators:

- Transfer the schemas of the input data sets to the output and catenates them to produce a "join" data set.
- Transfer the schema variables of the input data sets to the corresponding output schema variables. The effect is a catenation of the fields of the schema variables of the input data sets. For the fullouterjoin operator, duplicate field names are copied from the left and right data sets as leftRec_field and rightRec_field, where field is the name of the duplicated key field.
- Transfer duplicate records to the output, when appropriate.
- The innerjoin and leftouterjoin operators drop the key fields from the right data set.
- The rightouterjoin operator drops the key fields from the left data set.

Input data set requirements

These are as follows:

- The innerjoin, leftouterjoin, and rightouterjoin operators accept two or more input data sets, perform cascading joins on them, and produce one output data set. The fullouterjoin operator acts on exactly two input data sets to produce one output data set
- Each record of the left data set and the right data set must have one or more key fields with the same names and compatible data types. If they do not, you can change unmatched names and incompatible data types by means of the modify operator. See "Modify Operator" for a description of the modify operator.
- Compatible data types are those that WebSphere DataStage converts by default, translating a value in a source field to the data type of a destination field.
- Key fields can contain nulls. The join operators treat nulls as distinct values for the purposes of key comparison. Null handling is automatic.

Memory use

For the right data set, for each value of the key group, the collection of records with that key value must fit comfortably in memory to prevent paging from slowing performance.

Job monitor reporting

When the Job Monitor reports the number of records processed by a join operator, the count represents the number of records actually read by the operator. Depending on the type of join, there are some records that are not read. Examples are the records in a data set which do not contain the key values present in the other data set.

Comparison with other operators

The join operators are similar in some ways to the lookup and merge operators in that each are used to combine data from two or more sources. Each operator takes a primary input on input port 0 and one or more secondary inputs on input ports 1 and possibly 2, 3, 4, and so on and produces an output on output port 0 and one or more reject outputs on output ports 1 and possibly 2, 3, 4, and so on. The differences are summarized in the following table.

Table 123. Comparison of Joins, Lookup, and Merge

	Joins	Lookup	Merge
Description	RDBMS-style relational tables	Source and lookup table in RAM	Master table and one or more update tables
Memory usage	Light	Heavy	Light

Table 123. Comparison of Joins, Lookup, and Merge (continued)

	Joins	Lookup	Merge
Number and names of inputs	2 or more inputs	1 source and N lookup tables	1 master table and N update tables
Handling of duplicates in primary input	OK, produces a cross-product	OK	Warning given. Duplicate will be an unmatched primary.
Handling of duplicates in secondary input	OK, produces a cross-product	Warning given. The second lookup table entry is ignored.	OK only when $N = 1$
Options on unmatched primary	NONE	Fail, continue, drop, or reject. Fail is the default.	Keep or drop. Keep is the default.
Options on unmatched secondary	NONE	NONE	Capture in reject sets
On match, secondary entries are	reusable	reusable	reusable
Number of outputs	1	1 output and optionally 1 reject	1 output and 1 reject for each update table
Captured in reject sets	Does not apply	Unmatched primary entries	Unmatched secondary entries

Input data used in the examples

The next sections discuss the four join operators individually and give example joins for each. To simplify the examples, only two input data sets are used. However, the innerjoin, leftouterjoin, and rightouterjoin operators accept two or more input data sets and perform cascading joins on them, but the process is similar for each join.

Each operator section shows the following input data sets as acted upon by the operator in question. The data sets, which are presented in tabular form, track real estate purchases. Equal values in the price field of each data set are shown with double underscores.

left data set			right data set	
status field	price field		price field	id field
Sold	125		113	NI6325
Sold	213		125	BR9658
Offered	378		285	CZ2538
Pending	575		628	RU5713
Pending	649		668	SA5680
Offered	777		777	JA1081
Offered	908		908	DE1911
Pending	908		908	FR2081

innerjoin operator

The innerjoin operator transfers records from both input data sets whose key fields contain equal values to the output data set. Records whose key fields do not contain equal values are dropped.

Innerjoin: syntax and options

The syntax for the innerjoin operator is:

```
innerjoin  
-key field_name [-cs | ci] [-param params]  
[-key field_name [-cs | ci] [-param params] ...]  
[-collation_sequence locale | collation_file_pathname | OFF]
```

There is one required option, -key. You can specify it multiple times. Only top-level, non-vector fields can be keys.

Table 124. innerjoin Operator option

Option	Use
-key	<p>-key <i>field_name</i> [-cs ci] [-param <i>params</i>]</p> <p>Specify the name of the key field or fields.</p> <p>You can specify multiple keys. For each one, specify the -key option and supply the key's name.</p> <p>By default, WebSphere DataStage interprets the value of key fields in a case-sensitive manner. Specify -ci to override this default. Do so for each key you choose, for example:</p> <p style="padding-left: 2em;">-key A -ci -key B -ci</p> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property = value</i> pairs separated by commas.</p>
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none">• Specify a predefined IBM ICU locale• Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i>• Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site:</p> <p style="padding-left: 2em;">http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>

Example

In this example, the innerjoin operation is performed on two input data sets using the price field as the key field. Equal values in the price field of the left and right data sets are shown with double underscores.

left data set		right data set	
status field	price field	price Field	id Field
Sold	125	113	NI6325
Sold	213	125	BR9658
Offered	378	285	CZ2538
Pending	575	628	RU5713
Pending	649	668	SA5680
Offered	777	777	JA1081
Offered	908	908	DE1911
Pending	908	908	FR2081

Here are the results of the innerjoin operation:

status Field	price Field	id field
Sold	125	BR9658
Offered	777	JA1081
Offered	908	DE1911
Offered	908	FR2081
Pending	908	DE1911
Pending	908	FR2081

The osh syntax for the above example is:

```
$ osh "... innerjoin -key price ..."
```

leftouterjoin operator

The leftouterjoin operator transfers all values from the left data set and transfers values from the right data set only where key fields match. The operator drops the key field from the right data set. Otherwise, the operator writes default values.

Leftouterjoin: syntax and options

The syntax for the leftouterjoin operator is:

```
leftouterjoin
  -key field_name [-cs | -ci] [-param params]
  [-key field_name [-cs | -ci] [-param params] ...]
  [-collation_sequence locale | collation_file_pathname | OFF]
```

There is one required option, -key. You can specify it multiple times. Only top-level, non-vector fields can be keys.

Table 125. *leftouterjoin* Operator Option

Option	Use
-key	<p>-key <i>field_name</i> [-ci or -cs] [-param <i>params</i>]</p> <p>Specify the name of the key field or fields.</p> <p>You can specify multiple keys. For each one, specify the -key option and supply the key's name.</p> <p>By default, WebSphere DataStage interprets the value of key fields in a case-sensitive manner. Specify -ci to override this default. Do so for each key you choose, for example:</p> <pre>-key A -ci -key B -ci</pre> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property = value</i> pairs separated by commas.</p>
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> • Specify a predefined IBM ICU locale • Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> • Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, see reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>

Example

In this example, the *leftouterjoin* operation is performed on two data sets using the price field as the key field. Equal values in the price field of each data set are shown with double underscores.

Here are the input data sets:

left data set		right data set	
status Field	price Field	price Field	id Field
Sold	125	113	NI6325
Sold	213	125	BR9658
Offered	378	285	CZ2538
Pending	575	628	RU5713
Pending	649	668	SA5680
Offered	777	777	JA1081

left data set		right data set	
status Field	price Field	price Field	id Field
Offered	908	<u>908</u>	DE1911
Pending	908	<u>908</u>	FR2081

Here are the results of the leftouterjoin operation on the left and right data sets.

status Field	price Field	id Field
Sold	125	BR9658
Sold	213	
Offered	378	
Pending	575	
Pending	649	
Offered	777	JA1081
Offered	908	DE1911
Offered	908	FR2081
Pending	908	DE1911
Pending	908	FR2081

Here is the syntax for the example shown above in an osh command:

```
$ osh "... leftouterjoin -key price ..."
```

rightouterjoin operator

The rightouterjoin operator transfers all values from the right data set and transfers values from the left data set only where key fields match. The operator drops the key field from the left data set. Otherwise, the operator writes default values.

Rightouterjoin: syntax and options

The syntax for the rightouterjoin operator is shown below:

```
rightouterjoin
-key field_name [-cs | -ci] [-param params]
[-key field_name [-cs | -ci] [-param params] ...]
[-collation_sequence locale | collation_file_pathname | OFF]
```

There is one required option, -key. You can specify it multiple times. Only top-level, non-vector fields can be keys.

Table 126. rightouterjoin Operator Options

Option	Use
-key	<p>-key <i>field_name</i> [-ci or -cs] [-param <i>params</i>]</p> <p>Specify the name of the key field or fields.</p> <p>You can specify multiple keys. For each one, specify the -key option and supply the key's name.</p> <p>By default, WebSphere DataStage interprets the value of key fields in a case-sensitive manner. Specify -ci to override this default. Do so for each key you choose, for example:</p> <pre>-key A -ci -key B -ci</pre> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property = value</i> pairs separated by commas.</p>
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> • Specify a predefined IBM ICU locale • Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> • Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>

Example

In this example, the rightouterjoin operation is performed on two data sets using the price field as the key field. Equal values in the price field of each data set are shown with double underscores.

Here are the input data sets:

left data set		right data set	
status Field	price Field	price Field	id Field
Sold	125	113	NI6325
Sold	213	125	BR9658
Offered	378	285	CZ2538
Pending	575	628	RU5713
Pending	649	668	SA5680
Offered	777	777	JA1081

left data set		right data set	
status Field	price Field	price Field	id Field
Offered	908	<u>908</u>	DE1911
Pending	908	<u>908</u>	FR2081

Here are the results of the rightouterjoin operation on the left and right data sets.

status Field	Field	id Field
	113	NI6325
Sold	125	BR9658
	285	CZ2538
	628	RU5713
	668	SA5680
Offered	777	JA1081
Offered	908	DE1911
Offered	908	FR2081
Pending	908	DE1911
Pending	908	FR2081

Here is the syntax for the example shown above in an osh command:

```
$ osh "... rightouterjoin -key price ..."
```

fullouterjoin operator

The fullouterjoin operator transfers records whose key fields are equal in both input data sets to the output data set. It also transfers records whose key fields contain unequal values from both input data sets to the output data set.

The output data set:

- Contains all input records, except where records match; in this case it contains the cross-product of each set of records with an equal key
- Contains all input fields
- Renames identical field names of the input data sets as follows: `leftRec_field` (left data set) and `rightRec_field` (right data set), where `field` is the field name
- Supplies default values to the output data set, where values are not equal

Fullouterjoin: syntax and options

The syntax for the fullouterjoin operator is shown below:

```
fullouterjoin
  -key field_name [-cs | -ci] [-param params]
  [-key field_name [-cs | -ci] [-param params] ...]
  [-collation_sequence locale | collation_file_pathname | OFF]
```

There is one required option, `-key`. You can specify it multiple times. Only top-level, non-vector fields can be keys.

Table 127. fullouterjoin Operator Option

Option	Use
-key	<p>-key <i>field_name</i> [-ci or -cs] [-param <i>params</i>]</p> <p>Specify the name of the key field or fields.</p> <p>You can specify multiple keys. For each one, specify the -key option and supply the key's name.</p> <p>By default, WebSphere DataStage interprets the value of key fields in a case-sensitive manner. Specify -ci to override this default. Do so for each key you choose, for example:</p> <pre>-key A -ci -key B -ci</pre> <p>The -param suboption allows you to specify extra parameters for a field. Specify parameters using <i>property = value</i> pairs separated by commas.</p>
-collation_sequence	<p>-collation_sequence <i>locale</i> <i>collation_file_pathname</i> OFF</p> <p>This option determines how your string data is sorted. You can:</p> <ul style="list-style-type: none"> • Specify a predefined IBM ICU locale • Write your own collation sequence using ICU syntax, and supply its <i>collation_file_pathname</i> • Specify OFF so that string comparisons are made using Unicode code-point value order, independent of any locale or custom sequence. <p>By default, WebSphere DataStage sorts strings using byte-wise comparisons.</p> <p>For more information, reference this IBM ICU site: http://oss.software.ibm.com/icu/userguide/Collate_Intro.html</p>

Example

In this example, the rightouterjoin operation is performed using the price field as the key field. Equal values in the price field of each data set are shown with double underscores.

Here are the input data sets:

left data set		right data set	
status Field	price Field	price Field	id Field
Sold	125	113	NI6325
Sold	213	125	BR9658
Offered	378	285	CZ2538
Pending	575	628	RU5713
Pending	649	668	SA5680
Offered	<u>777</u>	<u>777</u>	JA1081

left data set		right data set	
status Field	price Field	price Field	id Field
Offered	<u>908</u>	<u>908</u>	DE1911
Pending	<u>908</u>	<u>908</u>	FR2081

Here are the results of the fullouterjoin operation on the left and right data sets.

status Field	leftRec_Price Field	rightRec_price Field	id Field
		113	NI6325
Sold	125	125	BR9658
Sold	213		
		285	CZ2538
Offered	378		
Pending	575		
		628	RU5713
Pending	649		
		668	SA5680
Offered	777	777	JA1081
Offered	908	908	DE1911
Offered	908	908	FR2081
Pending	908	908	DE1911
Pending	908	908	FR2081

The syntax for the example shown above in an osh command is:

```
$ osh "... fullouterjoin -key price ..."
```

Chapter 14. The ODBC interface library

The ODBC operators connect to various databases using the ODBC interface.

The library contains the following operators:

- `odbcread`. Reads records from an external datasource table and places them in a WebSphere DataStage data set. To do this, the Stage uses the `odbcread` operator.
- `dbcwrite`. Sets up a connection to an external datasource and inserts records into a table. The operator takes a single input data set. The write mode determines how the records of a data set are inserted into the table.
- `dbcupsert`. Inserts data into an external datasource table or updates an external datasource table with data contained in a WebSphere DataStage data set. You can match records based on field names and then update or insert those records.
- `dbclookup`. Performs a join operation between an external datasource table and a WebSphere DataStage data set, with the resulting data output as a WebSphere DataStage data set.

Accessing ODBC from WebSphere DataStage

This section assumes that WebSphere DataStage users have been configured to access ODBC.

This section assumes that WebSphere DataStage client machines have been configured to access external datasource(s) using the external datasource configuration process.

To access an external datasource from WebSphere DataStage:

When ODBC Enterprise stage is on a distributed WebSphere DataStage Server and the parallel engine shares the same ODBC settings:

1. DataDirect ODBC drivers are installed in the directory `$dshome/.../branded_odbc`. The shared library path is modified to include `$dshome/.../branded_odbc/lib`. The ODBCINI environment variable will be set to `$dshome/.odbc.ini`.
2. Start the external datasource.
3. Add `$APT_ORCHHOME/branded_odbc` to your PATH and `$APT_ORCHHOME/branded_odbc/lib` to your LIBPATH, LD_LIBRARY_PATH, or SHLIB_PATH. The ODBCINI environment variable must be set to the full path of the odbc.ini file.
4. Access the external datasource using a valid user name and password.

National Language Support

WebSphere DataStage's National Language Support (NLS) makes it possible for you to process data in international languages using Unicode character sets. WebSphere DataStage uses International Components for Unicode (ICU) libraries to support NLS functionality. For information on National Language Support, see *WebSphere DataStage NLS Guide* and access the ICU home page:

<http://oss.software.ibm.com/developerworksopensource/icu/project>

The ODBC operators support Unicode character data in:

- schema, table, and index names.
- user names and passwords.
- column names.

- table and column aliases.
- SQL*Net service names.
- SQL statements.
- file-name and directory paths.

ICU character set options

The operators have two options which optionally control character mapping:

- `-db_cs icu_character_set`

Specifies an ICU character set to map between ODBC char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to ODBC.

- `-nchar_cs icu_character_set`

Specifies an ICU character set to map between ODBC nchar and nvarchar2 values and WebSphere DataStage ustring data.

Mapping between ODBC and ICU character sets

To determine what ODBC character set corresponds to your ICU character-set specifications, WebSphere DataStage uses the table in \$APT_ORCHHOME/etc/ODBC_cs.txt. The table is populated with some default values. Update this file if you are using an ICU character set that has no entry in the file; otherwise UTF8 is used. The ODBC character set is specified in the sqldr control file as the CHARACTERSET option for loading your data.

Here is a sample ODBC_cs.txt table:

Table 128. ODBC_cs.txt Table

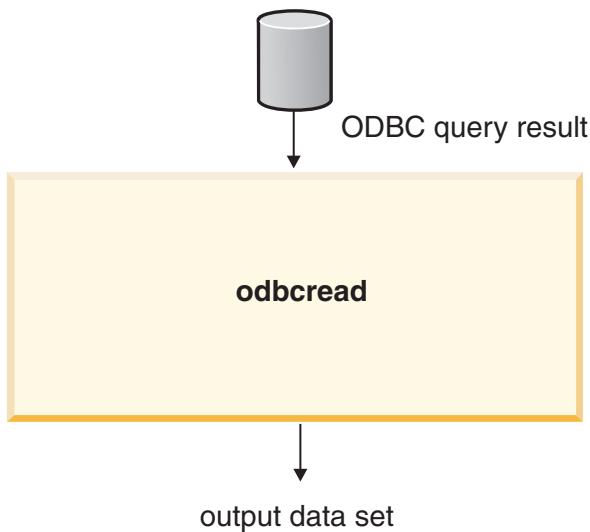
ICU Character Set	ODBC Character Set
UTF-8	UTF8
UTF-16	AL16UTF16
ASCL_ISO8859-1	WE8ISO8859P1
ISO-8859-1	WE8ISO8859P1
ASCL_MS1252	WE8ISO8859P1
EUC-JP	JA16EUC
ASCL-JPN-EUC	JA16EUC
ASCL_JPN-SJIS	JA16SJIS
Shift_JIS	JA16SJIS
US-ASCII	US7ASCII
ASCL-ASCII	US7ASCII

The odbcread operator

The odbcread operator reads records from an ODBC table, using an SQL query to request the records from the table. It then places the records in a WebSphere DataStage output data set.

The operator can perform both parallel and sequential database reads. The default execution mode is sequential. If you wish to use the parallel mode, then you must specify the `-partitioncol` property. For details about how to specify this property, see the odbcread Operator Options table below.

Data flow diagram



odbcread: properties

Table 129. odbcread Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1
Input interface schema	None
Output interface schema	Determined by the SQL query
Transfer behavior	None
Execution mode	Parallel and sequential
Partitioning method	MOD based
Collection method	Not applicable
Preserve-partitioning flag in the output data set	Clear
Composite Stage	No

Odbclookup: syntax and options

The syntax for the odbcread operator follows. The optional values that you provide are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
odbcread
  -query sql_query |
  -tablename table_name
    [-filter where_predicate]
    [-list select_predicate]
  -datasourcename data_source_name [-username user_name]
  [-password password]
  [-close close_command]
  [-db_cs icu_code_page [-use_strings]]
  [-open open_command]
  [-partitionCol PartitionColumnName]
  [-arraysize size]
  [-isolation_level read_committed | read_uncommitted | repeatable_read | serializable]
```

You must specify either the -query or -tablename option. You must also specify the -datasourcename, -username, and -password options.

Table 130. odbcread Operator Options

Option	Use
-query	<p>-query <i>sql_query</i></p> <p>Specify an SQL query to read from one or more tables.</p> <p>Note: The -query option is mutually exclusive with the -table option.</p>
-table	<p>-tablename <i>table_name</i></p> <p>Specify the table to be read from. It might be fully qualified. This option has two suboptions:</p> <ul style="list-style-type: none"> • -filter <i>where_predicate</i>: Optionally specify the rows of the table to exclude from the read operation. This predicate is appended to the where clause of the SQL statement to be executed. • -list <i>select_predicate</i>: Optionally specify the list of column names that appear in the select clause of the SQL statement to be executed. <p>Note: This option is mutually exclusive with the -query option.</p>
-datasource	<p>-datasourcename <i>data_source_name</i></p> <p>Specify the data source to be used for all database connections. This option is required.</p>
-user	<p>-username <i>user_name</i></p> <p>Specify the user name used to connect to the data source. This option might or might not be required depending on the data source.</p>
-password	<p>-password <i>password</i></p> <p>Specify the password used to connect to the data source. This option might or might not be required depending on the data source.</p>
-open	<p>-opencommand <i>open_command</i></p> <p>Optionally specify an SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.</p>
-close	<p>-closecommand <i>close_command</i></p> <p>Optionally specify an SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statements are executed only once on the conductor node.</p>
-arraysize	<p>-arraysize <i>n</i></p> <p>Specify the number of rows to retrieve during each fetch operation. The default number of rows is 1.</p>
-isolation_level	<p>--isolation_level</p> <p>read_uncommitted read_committed repeatable_read serializable</p> <p>Optionally specify the transaction level for accessing data. The default transaction level is decided by the database or possibly specified in the data source.</p>
-db_cs	<p>-db_cs <i>character_set</i></p> <p>Optionally specify the ICU code page which represents the database character set in use. The default is ISO-8859-1.</p>
-use_strings	<p>-use_strings</p> <p>If the -use_strings option is set, strings instead of ustrings are generated in the WebSphere DataStage schema.</p>

Table 130. *odbcread Operator Options (continued)*

Option	Use
-partitioncol	<p>-partition_column</p> <p>Use this option to run the ODBC read operator in parallel mode. The default execution mode is sequential. Specify the key column; the data type of this column must be integer. The column should preferably have the Monoatomically Increasing Order property.</p> <p>If you use the -partitioncol option, then you must follow the sample OSH scripts below to specify your -query and -table options.</p> <ul style="list-style-type: none"> • Sample OSH for -query option: <pre>odbcread -data_source SQLServer -user sa -password asas -query 'select * from SampleTable where Col1 =2 and %orchmodColumn% ' -partitionCol Col1 > OutDataSet.ds</pre> <ul style="list-style-type: none"> • Sample OSH for -table option: <pre>odbcread -data_source SQLServer -user sa -password asas -table SampleTable -partitioncol Col1 > OutDataset.ds</pre>

Operator action

Here are the chief characteristics of the odbcread operator:

- It runs both in parallel and sequential mode.
- It translates the query's result set (a two-dimensional array) row by row to a WebSphere DataStage data set.
- Its output is a WebSphere DataStage data set you can use as input to a subsequent WebSphere DataStage operator.
- Its translation includes the conversion of external datasource data types to WebSphere DataStage data types.
- The size of external datasource rows can be greater than that of WebSphere DataStage records.
- It specifies either an external datasource table to read or directs WebSphere DataStage to perform an SQL query.
- It optionally specifies commands to be run before the read operation is performed and after it has completed.
- It can perform a join operation between a WebSphere DataStage data set and an external data source. There might be one or more tables.

Column name conversion

An external datasource result set is defined by a collection of rows and columns. The odbcread operator translates the query's result set, a two-dimensional array, to a WebSphere DataStage data set. The external datasource query result set is converted to a WebSphere DataStage data set in the following way:

- The schema of an external datasource result set should match the schema of a WebSphere DataStage data set.
- The columns of an external datasource row should correspond to the fields of a WebSphere DataStage record. The name and data type of an external datasource column should correspond to the name and data type of a WebSphere DataStage field.
- The column names are used except when the external datasource column name contains a character that WebSphere DataStage does not support. In that case, two underscore characters replace the unsupported character.

- Both external datasource columns and WebSphere DataStage fields support nulls. A null contained in an external datasource column is stored as the keyword NULL in the corresponding WebSphere DataStage field.

Data type conversion

The odbcread operator converts external data source data types to osh data types, as shown in the following table:

Table 131. Mapping of ODBC Data Types to osh Data Types

ODBC Data Type	DataStage Data Type
SQL_CHAR	string[n]
SQL_VARCHAR	string[max=n]
SQL_WCHAR	wstring(n)
SQL_WVARCHAR	wstring(max=n)
SQL_DECIMAL	decimal(p,s)
SQL_NUMERIC	decimal(p,s)
SQL_SMALLINT	int16
SQL_INTEGER	int32
SQL_REAL	decimal(p,s)
SQL_FLOAT	decimal(p,s)
SQL_DOUBLE	decimal(p,s)
SQL_BIT	int8 (0 or 1)
SQL_TINYINT	int8
SQL_BIGINT	int64
SQL_BINARY	raw(n)
SQL_VARBINARY	raw(max=n)
SQL_TYPE_DATEP[6]P	date
SQL_TYPE_TIMEP[6]P	time[p]
SQL_TYPE_TIMESTAMPP[6]P	timestamp[p]
SQL_GUID	string[36]

Note: Data types that are not listed in the table above generate an error.

External data source record size

External datasource records can be larger than 32 KB, however, the size of a WebSphere DataStage record is limited to 32 KB. If you attempt to read a record larger than 32 KB, WebSphere DataStage returns an error and terminates the application.

Reading external data source tables

To read an external datasource table use one of the following methods:

- Specify the table name. This allows WebSphere DataStage to generate a default query that reads all records in the table.
- Explicitly specify the query.

The read methods are described in the following two sections.

Specifying the external data source table name

When you use the -table option, WebSphere DataStage issues the following SQL SELECT statement to read the table:

```
select [list]
from table_name and (filter);
```

You can optionally specify these options to narrow the read operation:

- -list specifies the columns of the table to be read. By default, WebSphere DataStage reads all columns.
- -filter specifies the rows of the table to exclude from the read operation. By default, WebSphere DataStage reads all rows.

You can optionally specify the -opencommand and -closecommand options. The open commands are executed by ODBC on the external data source before the table is opened; and the close commands are executed by ODBC on the external data source after it is closed.

Specifying an SQL SELECT statement

When you use the -query option, an SQL query is passed to the odbcread operator. The query does a select on the table as it is read into WebSphere DataStage. The SQL statement can contain joins, views, database links, synonyms, and so on. However, the following restrictions apply to this option:

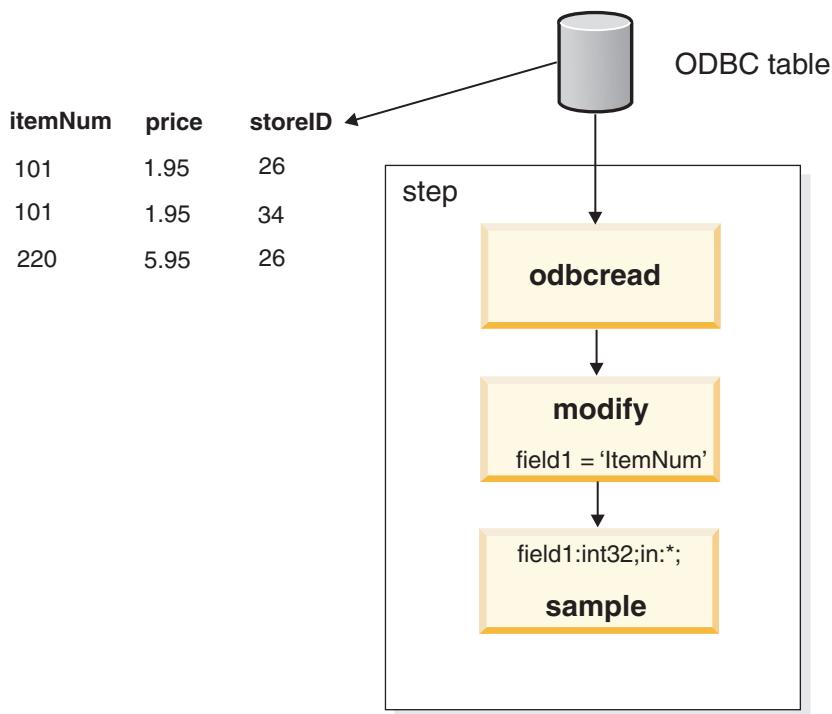
- It cannot contain bind variables.
- If you want to include -filter or -list options, you must specify them as part of the query.
- The query runs sequentially.
- You can specify optional open and close commands. These commands can be run just before reading data from a table and after reading data from a table.

Join operations

You can perform a join operation between WebSphere DataStage data sets and external data. First use the odbcread operator, and then the lookup operator or the join operator.

Odbcread example 1: reading an external data source table and modifying a field name

The following figure shows an external data source table used as input to a WebSphere DataStage operator:



The external datasource table contains three columns. The operator converts the data types of these columns as follows:

- itemNum of type NUMBER[3,0] is converted to int32.
- price of type NUMBER[6,2] is converted to decimal[6,2].
- storeID of type NUMBER[2,0] is converted to int32.

The schema of the WebSphere DataStage data set created from the table is also shown in this figure. Note that the WebSphere DataStage field names are the same as the column names of the external datasource table.

However, the operator to which the data set is passed has an input interface schema containing the 32-bit integer field field1, while the data set created from the external datasource table does not contain a field with the same name. Therefore, the modify operator must be placed between the odbcread operator and the sample operator to translate the name of the itemNum field to field1.

Here is the osh syntax for this example:

```
$ osh "odbcread -tablename 'table_name'
      -datasource data_source_name
      -user user_name
      -password password
      | modify '$modifySpec' | ...
      $modifySpec="field1 = itemNum;""
      modify
      ('field1 = itemNum,;')
```

The odbcwrite operator

The odbcwrite operator sets up a connection to an external data source and inserts records into a table. The operator takes a single input data set. The write mode of the operator determines how the records of a data set are inserted into the table.

Writing to a multibyte database

Specifying chars and varchars

Specify chars and varchars in bytes, with two bytes for each character. The following example specifies 10 characters:

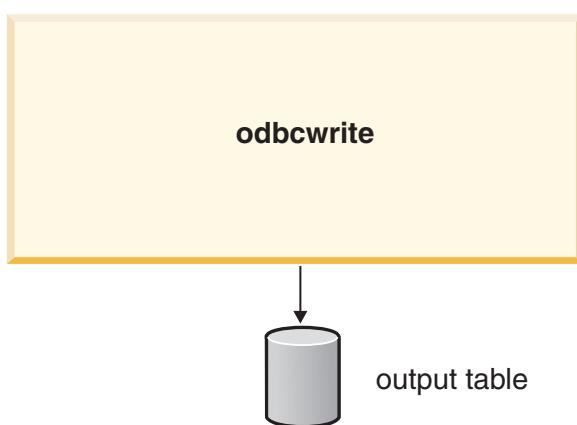
```
create table orch_data(col_a varchar(20));
```

Specifying nchar and nvarchar2 column size

Specify nchar and nvarchar2 columns in characters. The example below specifies 10 characters:

```
create table orch_data(col_a nvarchar2(10));
```

Data flow diagram



odbcwrite: properties

Table 132. odbcwrite Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	0
Input interface schema	Derived from the input data set
Output interface schema	None
Transfer behavior	None
Execution mode	Sequential by default or parallel
Partitioning method	Not applicable
Collection method	Any
Preserve-partitioning flag	The default is clear
Composite Stage	No

Operator action

Below are the chief characteristics of the odbcwrite operator:

- Translation includes the conversion of WebSphere DataStage data types to external datasource data types.

- The operator appends records to an existing table, unless you specify the create, replace, or truncate write mode.
- When you write to an existing table, the input data set schema must be compatible with the table schema.
- Each instance of a parallel write operator running on a processing node writes its partition of the data set to the external datasource table. You can optionally specify external datasource commands to be parsed and executed on all processing nodes before the write operation runs or after it is completed.

Where the odbcwrite operator runs

The default execution mode of the odbcwrite operator is parallel. By default the number of processing nodes is based on the configuration file. However, when the environment variable APT _CONFIG_FILE is set, the number of players is set to the number of nodes.

To run sequentially, specify the [seq] argument. You can optionally set the resource pool or a single node on which the operator runs.

Data conventions on write operations to external data sources

External datasource column names are identical to WebSphere DataStage field names, with the following restrictions:

- External datasource column names are limited to 30 characters. If a WebSphere DataStage field name is longer, you can do one of the following:
 - Choose the -truncate or -truncateLength options to configure the odbcwrite to truncate WebSphere DataStage field names to the maximum length of the datasource column name. If you choose -truncateLength, you can specify the number of characters to be truncated; the number should be less than the maximum length the data source supports.
 - Use the modify operator to modify the field name.
- A data set written to an external data source might not contain fields of certain types. If it does, an error occurs and the corresponding step terminates. However, WebSphere DataStage automatically modifies certain data types to those accepted by the external data source, as shown in the following table.

Table 133. Mapping of osh Data Types to ODBC Data Types

WebSphere DataStage Datatype	ODBC Datatype
string[n]	SQL_CHAR
string[max=n]	SQL_VARCHAR
wstring(n)	SQL_WCHAR
wstring(max=n)	SQL_WVARCHAR
decimal(p,s)	SQL_DECIMAL
decimal(p,s)	SQL_NUMERIC
int16	SQL_SMALLINT
int32	SQL_INTEGER
decimal(p,s)	SQL_REAL
decimal(p,s)	SQL_FLOAT
decimal(p,s)	SQL_DOUBLE
int8 (0 or 1)	SQL_BIT
int8	SQL_TINYINT
int64	SQL_BIGINT

Table 133. Mapping of osh Data Types to ODBC Data Types (continued)

WebSphere DataStage Datatype	ODBC Datatype
raw(<i>n</i>)	SQL_BINARY
raw(max= <i>n</i>)	SQL_VARBINARY
date	SQL_TYPE_DATEP[6]
time[<i>p</i>]	SQL_TYPE_TIMEP[6]P
timestamp[<i>p</i>]	SQL_TYPE_TIMESTAMPPI[6]P
string[36]	SQL_GUID

Write modes

The write mode of the operator determines how the records of the data set are inserted into the destination table. The four write modes are:

- append: This is the default mode. The table must exist and the record schema of the data set must be compatible with the table. The odbcwrite operator appends new rows to the table. The schema of the existing table determines the input interface of the operator.
- create: The odbcwrite operator creates a new table. If a table exists with the same name as the one being created, the step that contains the operator terminates with an error. The schema of the WebSphere DataStage data set determines the schema of the new table. The table is created with simple default properties. To create a table that is partitioned, indexed, in a non-default table space, or in some other non-standard way, you can use the -createstmt option with your own create table statement.
- replace: The operator drops the existing table and creates a new one in its place. If a table exists with the same name as the one you want to create, the existing table is overwritten. The schema of the WebSphere DataStage data set determines the schema of the new table.
- truncate: The operator retains the table attributes but discards existing records and appends new ones. The schema of the existing table determines the input interface of the operator. Each mode requires the specific user privileges shown in the table below.

Note: If a previous write operation failed, you can try again. Specify the replace write mode to delete any information in the output table that might have been written by the previous attempt to run your program.

Table 134. Required External Data Source Privileges for External Data Source Writes

Write Mode	Required Privileges
Append	INSERT on existing table
Create	TABLE CREATE
Replace	INSERT and TABLE CREATE on existing table
Truncate	INSERT on existing table

Matched and unmatched fields

The schema of the external datasource table determines the interface schema for the operator. Once the operator determines the schema, it applies the following rules to determine which data set fields are written to the table:

- Fields of the input data set are matched by name with fields in the input interface schema. WebSphere DataStage performs default data type conversions to match the input data set fields with the input interface schema.
- You can also use the modify operator to perform explicit data type conversions.

3. If the input data set contains fields that do not have matching components in the table, the operator generates an error and terminates the step.
4. WebSphere DataStage does not add new columns to an existing table if the data set contains fields that are not defined in the table. Note that you can use the odbcwrite -drop option to drop extra fields from the data set. Columns in the external datasource table that do not have corresponding fields in the input data set are set to their default value, if one is specified in the external datasource table. If no default value is defined for the external datasource column and it supports nulls, it is set to null. Otherwise, WebSphere DataStage issues an error and terminates the step.
5. WebSphere DataStage data sets support nullable fields. If you write a data set to an existing table and a field contains a null, the external data- source column must also support nulls. If not, WebSphere DataStage issues an error message and terminates the step. However, you can use the modify operator to convert a null in an input field to another value.

Odbcwrite: syntax and options

Syntax for the odbcwrite operator is given below. Optional values that you provide are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes. Exactly one occurrence of the -tablename option is required.

```
odbcwrite
  -tablename table_name  -datasourcename data_source_name  [-username user_name]
  [-password password]
  [-closecommand close_command]
  [-statement statement]
  [-drop]
  [-db_cs icu_code_page]
  [-arraysize n]
  [-transactionLevels read_uncommitted | read_committed | repeatable read | serializable]
  [-mode create | replace | append | truncate]
  [-opencommand open_command]
  [-rowCommitInterval n]
  [-truncate]
  [-truncateLength n]
  [-useNchar]
```

Table 135. odbcwrite Operator Options and Values

Option	Use
-datasourcename	<p>-datasourcename <i>data_source_name</i></p> <p>Specify the data source to be used for all database connections. This option is required.</p>
-username	<p>-username <i>user_name</i></p> <p>Specify the user name used to connect to the data source. This option might or might not be required depending on the data source.</p>
-password	<p>-password <i>password</i></p> <p>Specify the password used to connect to the data source. This option might or might not be required depending on the data source.</p>
-tablename	<p>-tablename <i>table_name</i></p> <p>Specify the table to write to. The table name might be fully qualified.</p>

Table 135. *odbcwrite* Operator Options and Values (continued)

Option	Use
-mode	<p>-mode append create replace truncate</p> <p>Specify the write mode as one of these modes:</p> <p>append: This operator appends new records into an existing table.</p> <p>create: This operator creates a new table. If a table exists with the same name as the one you want to create, the step that contains the operator terminates with an error. The schema of the new table is determined by the schema of the WebSphere DataStage data set. The table is created with simple default properties. To create a table that is partitioned, indexed, in a non-default table space, or in some other non-standard way, you can use the -createtestmt option with your own create table statement.</p> <p>replace: This operator drops the existing table and creates a new one in its place. The schema of the WebSphere DataStage data set determines the schema of the new table.</p> <p>truncate: This operator deletes all records from an existing table before loading new records.</p>
-statement	<p>-statement <i>create_statement</i></p> <p>Optionally specify the create statement to be used for creating the table when -mode create is specified.</p>
-drop	<p>-drop</p> <p>If this option is set, unmatched fields in the WebSphere DataStage data set are dropped. An unmatched field is a field for which there is no identically named field in the datasource table.</p>
-truncate	<p>-truncate</p> <p>When this option is set, column names are truncated to the maximum size allowed by the ODBC driver.</p>
-truncateLength	<p>-truncateLength <i>n</i></p> <p>Specify the length to truncate column names.</p>
-opencommand	<p>-opencommand <i>open_command</i></p> <p>Optionally specify an SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.</p>
-closecommand	<p>-closecommand <i>close_command</i></p> <p>Optionally specify an SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statements are executed only once on the conductor node.</p>
-arraysize	<p>-arraysize <i>n</i></p> <p>Optionally specify the size of the insert array. The default size is 2000 records.</p>

Table 135. odbcwrite Operator Options and Values (continued)

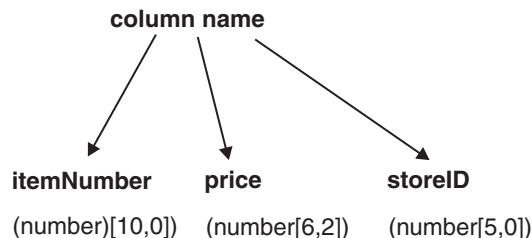
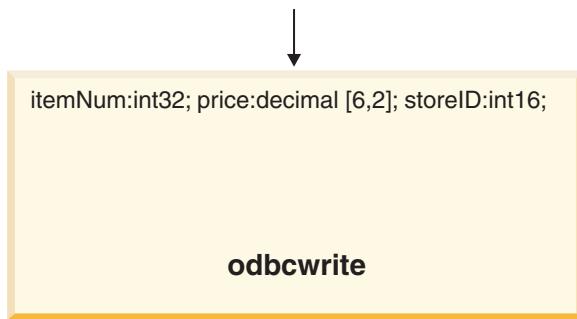
Option	Use
-rowCommitInterval	<p>-rowCommitInterval <i>n</i></p> <p>Optionally specify the number of records to be committed before starting a new transaction. This option can only be specified if arraysize = 1. Otherwise rowCommitInterval = arraysize. This is because of the rollback retry logic that occurs when an array execute fails.</p>
-transactionLevels	<p>-transactionLevels read_uncommitted read_committed repeatable_read serializable</p> <p>Optionally specify the transaction level for accessing data. The default transaction level is decided by the database or possibly specified in the data source.</p>
-db_cs	<p>-db_cs <i>code_page_name</i></p> <p>Optionally specify the ICU code page which represents the database character set in use. The default is ISO-8859-1.</p>
-useNchar	<p>-useNchar</p> <p>Read all nchars/nvarchar from the database.</p>

Example 1: writing to an existing external data source table

When an existing external datasource table is written to:

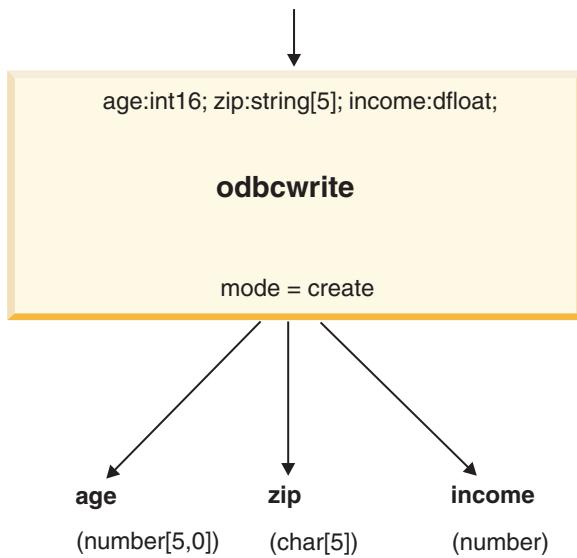
- The column names and data types of the external datasource table determine the input interface schema of the odbcwrite operator.
- This input interface schema then determines the fields of the input data set written to the table.

For example, the following figure shows the odbcwrite operator writing to an existing table:



Example 2: creating an external datasource table

To create a table, specify either create or replace write mode. The next figure is a conceptual diagram of the create operation:



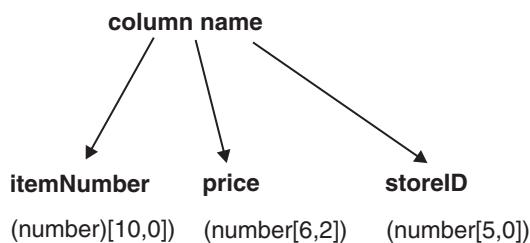
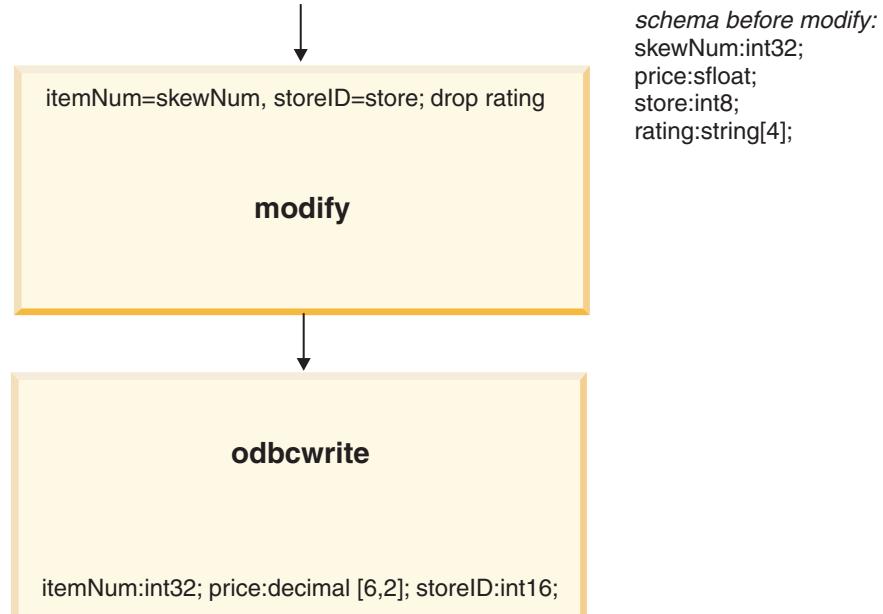
Here is the osh syntax for this example:

```
$ osh "... odbcwrite -table table_2 -mode create -dboptions {'user = user101, password = userPword'} ..."
```

To create the table, the `odbcwrite` operator names the external data-source columns the same as the fields of the input WebSphere DataStage data set and converts the WebSphere DataStage data types to external datasource data types.

Example 3: writing to an external data source table using the modify operator

The modify operator allows you to drop unwanted fields from the write operation and translate the name or data type of a field in the input data set to match the input interface schema of the operator.



In this example, the modify operator is used to:

- Translate the field names of the input data set to the names of the corresponding fields of the operator's input interface schema, that is, skewNum to itemNum and store to storeID.
- Drop the unmatched rating field so that no error occurs.

Note: WebSphere DataStage performs an automatic type conversion of store, promoting its int8 data type in the input data set to int16 in the odbcwrite input interface.

Here is the osh syntax for this example:

```
$ modifySpec="itemNum = skewNum, storeID = store;  
drop rating"  
$ osh "... op1 | modify '$modifySpec' | odbcwrite -table table_2  
-dboptions {'user = user101, password = userPword'}"
```

Other features

Quoted identifiers

All operators that accept SQL statements as arguments support quoted identifiers in those arguments. The quotes should be escaped with the `\' character.

Case sensitive column identifiers

Support for case-sensitive column identifiers will be added in a future release.

Largeobjects (BLOB, CLOB, and so on)

Large objects are not supported in this release.

Stored procedures

The ODBC operator does not support stored procedures.

Transaction rollback

Because it is not possible for native transactions to span multiple processes, rollback is not supported in this release.

Unit of work

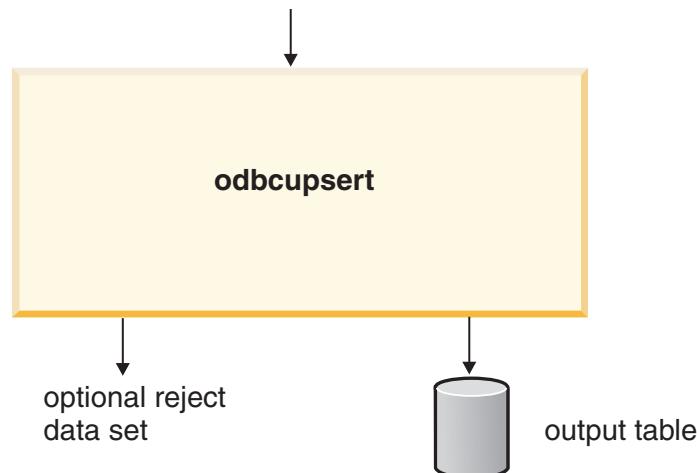
The unit-of-work operator has not been modified to support ODBC in this release.

The `odbcupsert` operator

The `odbcupsert` operator inserts and updates external datasource table records with data contained in a WebSphere DataStage data set. You provide the insert and update SQL statements using the `-insert` and `-update` options. By default, `odbcupsert` uses external datasource host-array processing to optimize the performance of inserting records.

This operator receives a single data set as input and writes its output to an external datasource table. You can request an optional output data set that contains the records which fail to be inserted or updated.

Data flow diagram



odbcupsert: properties

Table 136. odbcupsert Properties and Values

Property	Value
Number of input data sets	1
Number of output data sets by default	None; 1 when you select the -reject option
Input interface schema	Derived from your insert and update statements
Transfer behavior	Rejected update records are transferred to an output data set when you select the -reject option
Execution mode	Parallel by default, or sequential
Partitioning method	Same You can override this partitioning method. However, a partitioning method of entire cannot be used.
Collection method	Any
Combinable stage	Yes

Operator action

Here are the main characteristics of odbcupsert:

- If an -insert statement is included, the insert is executed first. Any records that fail to be inserted because of a unique-constraint violation are then used in the execution of the update statement.
- By default, WebSphere DataStage uses host-array processing to enhance the performance of insert array processing. Each insert array is executed with a single SQL statement. Updated records are processed individually.
- Use the -insertArraySize option to specify the size of the insert array. For example:
`-insertArraySize 250`
- The default length of the insert array is 500. To direct WebSphere DataStage to process your insert statements individually, set the insert array size to 1:
`-insertArraySize 1`
- The record fields can be of variable-length strings. You can either specify a maximum length or use the default maximum length of 80 characters.

The example below specifies a maximum length of 50 characters:

```
record(field1:string[max=50])
```

The maximum length in the example below is by default 80 characters:

```
record(field1:string)
```

- When an insert statement is included and host array processing is specified, a WebSphere DataStage update field must also be a WebSphere DataStage insert field.
- The odbcupsert operator converts all values to strings before passing them to the external data source. The following osh data types are supported:
 - int8, uint8, int16, uint16, int32, uint32, int64, and uint64
 - dfloat and sfloat
 - decimal
 - strings of fixed and variable length
 - timestamp
 - date
- By default, odbcupsert does not produce an output data set. Using the -reject option, you can specify an optional output data set containing the records that fail to be inserted or updated. Its syntax is:

`-reject filename`

Odbcupsert: syntax and options

The syntax for odbcupsert is shown below. Optional values are shown in italics. When your value contains a space or a tab character, you must enclose it in single quotes.

`odbcupsert`

```
-datasourcename data_source_name
-updateStatement update_statement |
-insertStatement insert_statement |
-deleteStatement
[-username user_name]
[-password password]
[-insertArraySize n]
[-reject]
[-opencommand open_command]
[-closecommand close_command]
[-db_cs icu_code_page]
[-rowCommitInterval n]
```

Exactly one occurrence of of -update option is required. All others are optional.

Specify an ICU character set to map between external datasource char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to an external data source. The default character set is UTF-8, which is compatible with the osh jobs that contain 7-bit US-ASCII data.

Table 137. odbcupsert Operator Options

Option	Use
<code>-datasourcename</code>	<code>-datasourcename <i>data_source_name</i></code> Specify the data source to be used for all database connections. This option is required.
<code>-username</code>	<code>-username <i>user_name</i></code> Specify the user name used to connect to the data source. This option might or might not be required depending on the data source.
<code>-password</code>	<code>-password <i>password</i></code> Specify the password used to connect to the data source. This option might or might not be required depending on the data source.
statement options	You must specify at least one of the following options but not more than two. An error is generated if you do not specify any statement option or specify more than two. <code>-updateStatement <i>update_statement</i></code> Optionally specify the update or delete statement to be executed. <code>-insertStatement <i>insert_statement</i></code> Optionally specify the insert statement to be executed. <code>-deleteStatement <i>delete_statement</i></code> Optionally specify the delete statement to be executed.

Table 137. *odbcupsert* Operator Options (continued)

Option	Use
-mode	<p>-mode insert_update update_insert delete_insert</p> <p>Specify the upsert mode to be used when two statement options are specified. If only one statement option is specified, the upsert mode is ignored.</p> <p>insert_update: The insert statement is executed first. If the insert fails due to a duplicate key violation (if the record being inserted exists), the update statement is executed. This is the default upsert mode.</p> <p>update_insert: The update statement is executed first. If the update fails because the record does not exist, the insert statement is executed.</p> <p>delete_insert: The delete statement is executed first; then the insert statement is executed.</p>
-reject	<p>-reject</p> <p>If this option is set, records that fail to be updated or inserted are written to a reject data set. You must designate an output data set for this purpose. If this option is not specified, an error is generated if records fail to update or insert.</p>
-open	<p>-open <i>open_command</i></p> <p>Optionally specify an SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.</p>
-close	<p>-close <i>close_command</i></p> <p>Optionally specify an SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statements are executed only once on the conductor node.</p>
-insertarraysize	<p>-insertarraysize <i>n</i></p> <p>Optionally specify the size of the insert/update array. The default size is 2000 records.</p>
-rowCommitInterval	<p>-rowCommitInterval <i>n</i></p> <p>Optionally specify the number of records to be committed before starting a new transaction. This option can only be specified if arraysizes = 1. Otherwise rowCommitInterval = arraysizes. This is because of the rollback logic/retry logic that occurs when an array execution fails.</p>

Example

This example shows the updating of an ODBC table that has two columns named acct_id and acct_balance, where acct_id is the primary key. Two of the records cannot be inserted because of unique key constraints. Instead, they are used to update existing records. One record is transferred to the reject data set because its acct_id generates an error.

Summarized below are the state of the ODBC table before the data flow is run, the contents of the input file, and the action that WebSphere DataStage performs for each record in the input file.

Table 138. Example Data

Table before dataflow	Input file contents	WebSphere DataStage Action		
acct_id	acct_balance	acct_id	acct_balance	
073587	45.64	873092	67.23	Update
873092	2001.89	865544	8569.23	Insert
675066	3523.62	566678	2008.56	Update
566678	89.72	678888	7888.23	Insert
		073587	82.56	Update
		995666	75.72	Insert

Here is the osh syntax:

```
$ osh "import
-schema record(acct_id:string[6] acct_balance:dfloat;)
-file input.txt |
hash -key acct_id |
tsort -key acct_id |
odbcupsert -data_source dsn -user apt -password test
-insert 'insert into accounts
values(ORCHESTRATE.acct_id, ORCHESTRATE.acct_balance)'
-update 'update accounts
set acct_balance = ORCHESTRATE.acct_balance
where acct_id = ORCHESTRATE.acct_id'
-reject '/user/home/reject/reject.ds'"
```

Table 139. Table after Upsert

acct_id	acct_balance
073587	82.56
873092	67.23
675066	3523.62
566678	2008.56
865544	8569.23
678888	7888.23
995666	75.72

The odbclookup operator

Using the odbclookup operator, you can perform a join between one or more external datasource tables and a WebSphere DataStage data set. The resulting output data is a WebSphere DataStage data set containing both WebSphere DataStage and external datasource data.

This join is performed by specifying an SQL SELECT statement, or one or more external datasource tables and one or more key fields on which to perform the lookup.

This operator is particularly useful for sparse lookups, where the WebSphere DataStage data set you are trying to match is much smaller than the external datasource table. If you expect to match 90% of your data, use the odbcread and odbclookup operators.

Because odbclookup can perform lookups against more than one external datasource table, it is useful for joining multiple external datasource tables in one query.

The -statement option command corresponds to an SQL statement of this form:

```
select a,b,c from data.testtbl  
where  
Orchestrate.b = data.testtbl.c  
and  
Orchestrate.name = "Smith"
```

The odbclookup operator replaces each WebSphere DataStage field name with a field value; submits the statement containing the value to the external data source; and outputs a combination of external data source and WebSphere DataStage data.

Alternatively, you can use the -key/-table options interface to specify one or more key fields and one or more external data source tables. The following osh options specify two keys and a single table:

```
-key a -key b -table data.testtbl
```

You get the same result as you would by specifying:

```
select * from data.testtbl  
where  
Orchestrate.a = data.testtbl.a  
and  
Orchestrate.b = data.testtbl.b
```

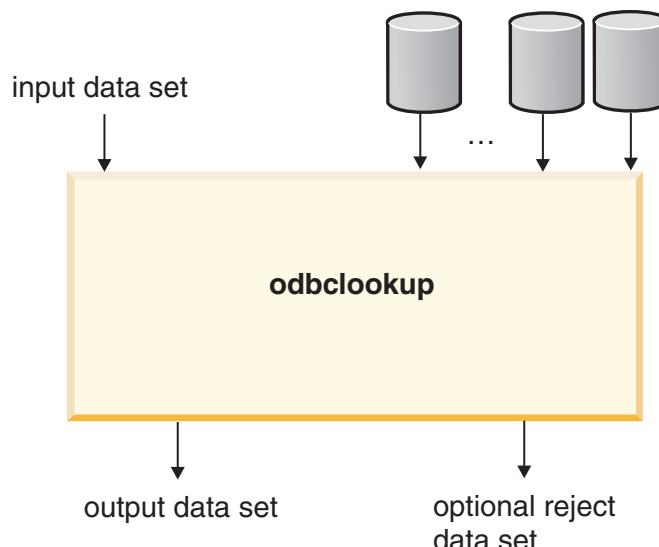
The resulting WebSphere DataStage output data set includes the WebSphere DataStage records and the corresponding rows from each referenced external data source table. When an external data source table has a column name that is the same as a WebSphere DataStage data set field name, the external datasource column is renamed using the following syntax:

APT_integer_fieldname

An example is APT_0_lname. The integer component is incremented when duplicate names are encountered in additional tables.

Note: If the external datasource table is not indexed on the lookup keys, the performance of this operator is likely to be poor.

Data flow diagram



odbclookup: properties

Table 140. odbclookup Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1; 2 if you include the -ifNotFound reject option
Input interface schema	Determined by the query
Output interface schema	Determined by the SQL query
Transfer behavior	Transfers all fields from input to output
Execution mode	Sequential or parallel (default)
Partitioning method	Not applicable
Collection method	Not applicable
Preserve-partitioning flag in the output data set	Clear
Composite stage	No

Odbclookup: syntax and options

The syntax for the odbclookup operator is given below. Optional values are shown in italics. When your value contains a space or a tab character, you must enclose it in single quotes.

```
odbclookup
  -data_source data_source_name
  -user username
  -password passwd
  -tablename table_name -key field [-key field ...]
  [-tablename table_name -key field [-key field ...]] |
  -query sql_query
  -ifNotFound fail | drop | reject | continue
  -fetcharraysize n
  [-open open_command]
  [-close close_command]
```

You must specify either the -query option or one or more -table options with one or more -key fields.

The odbclookup operator is parallel by default. The options are as follows:

Table 141. odbclookup Operator Options and Values

Option	Value
-datasourcename	-datasourcename <i>data_source_name</i> Specify the data source to be used for all database connections. This option is required.
-user	-user <i>user_name</i> Specify the user name used to connect to the data source. This option might or might not be required depending on the data source.
-password	-password <i>password</i> Specify the password used to connect to the data source. This option might or might not be required depending on the data source.

Table 141. *odbclookup Operator Options and Values (continued)*

Option	Value
-tablename	<p>-tablename <i>table_name</i></p> <p>Specify a table and key fields to be used to generate a lookup query. This option is mutually exclusive with the -query option. The -table option has three suboptions:</p> <ul style="list-style-type: none"> -filter <i>where_predicate</i>: Specify the rows of the table to exclude from the read operation. This predicate is appended to the where clause of the SQL statement to be executed. -selectlist <i>select_predicate</i>: Specify the list of column names that will appear in the select clause of the SQL statement to be executed. -key <i>field</i>: Specify a lookup key. A lookup key is a field in the table that is used to join with a field of the same name in the WebSphere DataStage data set. The -key option can be used more than once to specify more than one key field.
-ifNotFound	<p>-ifNotFound fail drop reject continue</p> <p>Specify an action to be taken when a lookup fails. Choose any of the following actions:</p> <ul style="list-style-type: none"> fail: Stop job execution. drop: Drop the failed record from the output data set. reject: Put records that are not found into a reject data set. You must designate an output data set for this option. continue: Leave all records in the output data set (outer join).
-query	<p>-query <i>sql_query</i></p> <p>Specify a lookup query to be executed. This option is mutually exclusive with the -table option.</p>
-open	<p>-open <i>open_command</i></p> <p>Optionally specify a SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.</p>
-close	<p>-close <i>close_command</i></p> <p>Optionally specify an SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statement is executed only once on the conductor node.</p>
-fetcharraysize	<p>-fetcarrraysize <i>n</i></p> <p>Specify the number of rows to retrieve during each fetch operation. The default number is 1.</p>

Table 141. *odbclookup Operator Options and Values (continued)*

Option	Value
-db_cs	<p>-db_cs <i>code_page_name</i></p> <p> Optionally specify the ICU code page which represents the database character set in use. The default is ISO-8859-1. This option has this suboption:</p> <p>-use_strings</p> <p>If this suboption is not set, strings, not ustrings, are generated in the WebSphere DataStage schema.</p>

Example

Suppose you want to connect to the APT81 server as user user101, with the password test. You want to perform a lookup between a WebSphere DataStage data set and a table called target, on the key fields lname, fname, and DOB. You can configure odbclookup in either of two ways to accomplish this.

Here is the osh command using the -table and -key options:

```
$ osh " odbclookup -key lname -key fname -key DOB
      < data1.ds > data2.ds "
```

Here is the equivalent OSH command using the -query option:

```
$ osh " odbclookup
      -query 'select * from target
              where lname = Orchestrate.lname
                  and fname = Orchestrate.fname
                  and DOB = Orchestrate.DOB'
      < data1.ds > data2.ds"
```

WebSphere DataStage prints the lname, fname and DOB column names and values from the WebSphere DataStage input data set, and the lname, fname, and DOB column names and values from the external datasource table.

If a column name in the external datasource table has the same name as a WebSphere DataStage output data set schema fieldname, the printed output shows the column in the external datasource table renamed using this format:

APT_integer_fieldname

For example, lname might be renamed to APT_0_lname.

Chapter 15. The SAS interface library

The SAS operators make it possible to execute SAS applications using the computing power and data handling capabilities of WebSphere DataStage's parallel processing system.

You need to make only minor modifications to your existing SAS applications for execution within WebSphere DataStage. The topic "Parallelizing SAS Steps" describes these modifications.

The WebSphere DataStage SAS Interface Library contains four operators:

- sasin. This operator converts a standard WebSphere DataStage data set into a SAS data set capable of being processed in parallel by the sas and sasout operators. The sasin operator options include an option for defining the input data-set schema.
- sas. This operator allows you to execute part or all of a SAS application in parallel. It takes SAS code in the form of DATA and PROC steps as its argument.
- sasout. This operator converts the SAS data set output by the sas operator to the standard data set format. The operator has a number of options including the -schema and -schemaFile options. You use either of these options to specify the schema of the output WebSphere DataStage data set.
- sascontents. This operator generates a report about a self-describing input data set. The report is similar to the report generated by the SAS procedure PROC CONTENTS.

Using WebSphere DataStage to run SAS code

WebSphere DataStage lets SAS users optimally exploit the performance potential of parallel relational database management systems (RDBMS) running on scalable hardware platforms. WebSphere DataStage extends SAS by coupling the parallel data transport facilities of WebSphere DataStage with the rich data access, manipulation, and analysis functions of SAS.

While WebSphere DataStage allows you to execute your SAS code in parallel, sequential SAS code can also take advantage of WebSphere DataStage to increase system performance by making multiple connections to a parallel database for data reads and writes, as well as through pipeline parallelism.

The SAS system consists of a powerful programming language and a collection of ready-to-use programs called procedures (PROCS). This section introduces SAS application development and explains the differences in execution and performance of sequential and parallel SAS programs.

Writing SAS programs

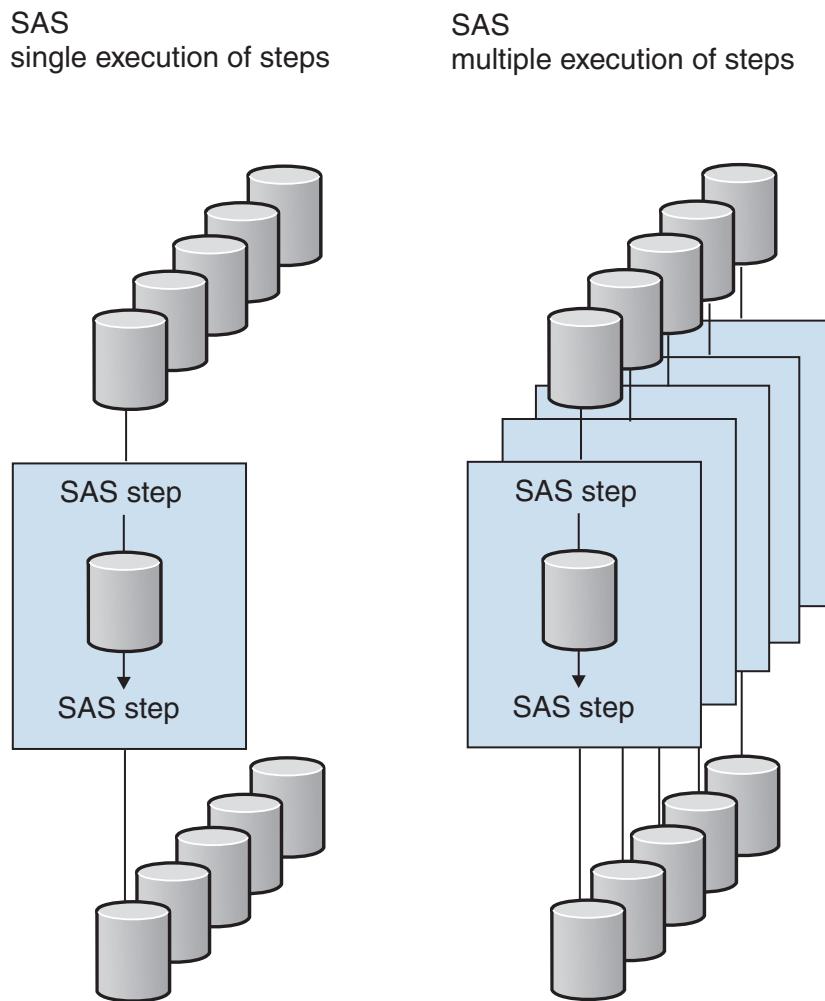
You develop SAS applications by writing SAS programs. In the SAS programming language, a group of statements is referred to as a SAS step. SAS steps fall into one of two categories: DATA steps and PROC steps.

SAS DATA steps usually process data one row at a time and do not look at the preceding or following records in the data stream. This makes it possible for the sas operator to process data steps in parallel. SAS PROC steps, however, are precompiled routines which cannot always be parallelized.

Using SAS on sequential and parallel systems

The SAS system is available for use on a variety of computing platforms, including single-processor UNIX workstations. Because SAS is written as a sequential application, WebSphere DataStage can provide a way to exploit the full power of scalable computing technology within the traditional SAS programming environment.

For example, the left side of the next figure shows a sequential SAS application that reads its input from an RDBMS and writes its output to the same database. This application contains a number of sequential bottlenecks that can negatively impact its performance.



In a typical client/server environment, a sequential application such as SAS establishes a single connection to a parallel RDBMS in order to access the database. Therefore, while your database might be explicitly designed to support parallel access, SAS requires that all input be combined into a single input stream.

One effect of single input is that often the SAS program can process data much faster than it can access the data over the single connection, therefore the performance of a SAS program might be bound by the rate at which it can access data. In addition, while a parallel system, either MPP or SMP, contains multiple CPUs, a single SAS job in SAS itself executes on only a single CPU; therefore, much of the processing power of your system is unused by the SAS application.

In contrast, WebSphere DataStage's parallel processing model, shown above on the right, allows the database reads and writes, as well as the SAS program itself, to run simultaneously, reducing or eliminating the performance bottlenecks that might otherwise occur when SAS is run on a parallel computer.

WebSphere DataStage enables SAS users to:

- Access, for reading or writing, large volumes of data in parallel from parallel relational databases, with much higher throughput than is possible using PROC SQL.

- Process parallel streams of data with parallel instances of SAS DATA and PROC steps, enabling scoring or other data transformations to be done in parallel with minimal changes to existing SAS code.
- Store large data sets in parallel, circumventing restrictions on data-set size imposed by your file system or physical disk-size limitations. Parallel data sets are accessed from SAS programs in the same way as conventional SAS data sets, but at much higher data I/O rates.
- Realize the benefits of pipeline parallelism, in which some number of WebSphere DataStage sas operators run at the same time, each receiving data from the previous process as it becomes available.

Pipeline parallelism and SAS

WebSphere DataStage applications containing multiple sas operators take advantage of pipeline parallelism. With pipeline parallelism, all WebSphere DataStage sas operators in your application run at the same time. A sas operator might be active, meaning it has input data available to be processed, or it might be blocked as it waits to receive input data from a previous sas operator or other WebSphere DataStage operator.

Steps that are processing on a per-record or per-group basis, such as a DATA step or a PROC MEANS with a BY clause, can feed individual records or small groups of records into downstream sas operators for immediate consumption, bypassing the usual write to disk.

Steps that process an entire data set at once, such as a PROC SORT, can provide no output to downstream steps until the sort is complete. The pipeline parallelism enabled by WebSphere DataStage allows individual records to feed the sort as they become available. The sort is therefore occurring at the same time as the upstream steps. However, pipeline parallelism is broken between the sort and the downstream steps, since no records are output by the PROC SORT until all records have been processed. A new pipeline is initiated following the sort.

Configuring your system to use the SAS interface operators

To enable use of the SAS interface operators, you or your WebSphere DataStage administrator should specify several SAS-related resources in your configuration file. You need to specify the following information:

- The location of the SAS executable;
- Optionally, a SAS work disk, which is defined as the path of the SAS work directory;
- Optionally, a disk pool specifically for parallel SAS data sets, called sasdataset.

An example of each of these declarations is below. The resource sas declaration must be included; its pathname can be empty. If the -sas_cs option is specified, the resource sasint line must be included.

```
resource sas " [/usr/sas612/] " { }
resource sasint " [/usr/sas8.2int/] " { }
resource sasworkdisk "/usr/sas/work/" { }
resource disk "/data/sas/" { pool "" "sasdataset" }
```

In a configuration file with multiple nodes, the path for each resource disk must be unique in order to obtain the SAS parallel data sets. The following configuration file for an MPP system differentiates the resource disk paths by naming the final directory with a node designation; however, you can use any directory paths as long as they are unique across all nodes. Bold type is used for illustration purposes only to highlight the resource directory paths.

```
{
  node "elwood1"
  {
    fastname "elwood"
    pools ""
    resource sas "/usr/local/sas/sas8.2" { }
    resource disk "/opt/IBMm/InformationServer/Server/Datasets/node1"
      {pools "" "sasdataset" }
```

```

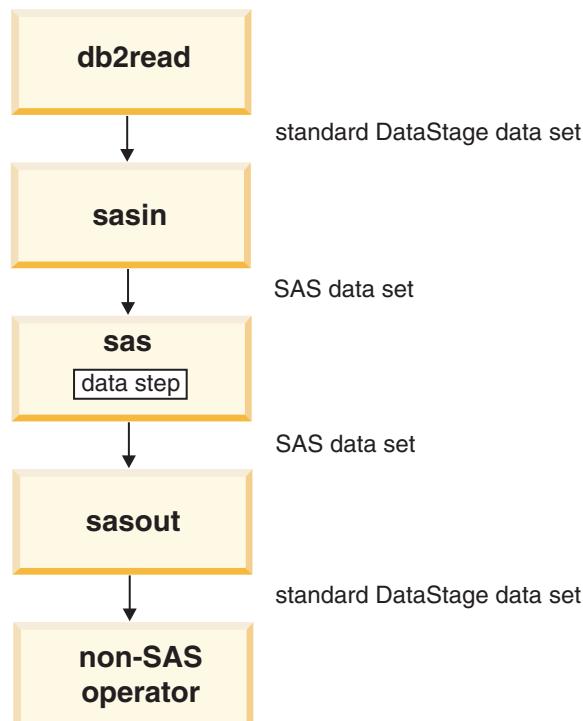
resource scratchdisk "/opt/IBMr/InformationServer/Server/Scratch"
  {pools ""}
}
node "solpxqa01-1"
{
  fastname "solpxqa01"
  pools ""
  resource sas "/usr/local/sas/sas8.2" { }
  resource disk "/opt/IBMr/InformationServer/Server/Datasets/node2"
    {pools "" "sasdataset" }
  resource scratchdisk "/opt/IBMr/InformationServer/Server/Scratch"
  {pools ""}
}
{
}

```

The resource names sas and sasworkdisk and the disk pool name sasdata set are reserved words.

An example data flow

This example shows a SAS application reading data in parallel from DB2.



The WebSphere DataStage db2read operator streams data from DB2 in parallel and converts it automatically into the standard WebSphere DataStage data set format. The sasin operator then converts it, by default, into an internal SAS data set usable by the sas operator. Finally, the sasout operator inputs the SAS data set and outputs it as a standard WebSphere DataStage data set.

The SAS interface operators can also input and output data sets in Parallel SAS data set format. Descriptions of the data set formats are in the next section.

Representing SAS and non-SAS Data in DataStage

WebSphere DataStage data set format

This is data in the normal WebSphere DataStage data set format. WebSphere DataStage operators, such as the database read and write operators, process data sets in this format.

Sequential SAS data set format

This is SAS data in its original SAS sequential format. You include statements in the SAS code of the sas operator to read in a SAS data set for processing. A SAS data set cannot be an input to a WebSphere DataStage operator except through a SAS read statement.

Today, reading of a SAS psds does not convert ANY SAS char fields to ustrings. User must put in a modify operator. Request that on creation, WebSphere DataStage schema fields of type ustring have names stored in psds description file (along with icu_map used). Then on reading, those fields can be converted to ustring.

Put ustring field names in psds description file

Parallel SAS data set format

This is a WebSphere DataStage-provided data set format consisting of one or more sequential SAS data sets. Each data set pointed to by the file corresponds to a single partition of data flowing through WebSphere DataStage. The file name for this data set is *.psds. It is analogous to a persistent *.ds data set.

The header is an ASCII text file composed of a list of sequential SAS data set files. By default, the header file lists the SAS char fields that are to be converted to ustring fields in WebSphere DataStage, making it unnecessary to use the modify operator to convert the data type of these fields. The format is:

```
DataStage SAS data set
UstringFields[ icu_mapname ]; field_name_1; field_name_2 ... field_name_n;
LFile:
platform_1 : filename_1
LFile: platform_2 : filename_2 ... LFile: platform_n : filename_n
```

For example:

```
DataStage SAS data set
UstringFields[ISO-2022-JP];B_FIELD;C_FIELD;
LFile:
eartha-gbit 0:
/apt/eartha1sand0/jgeyer701/orch_master/apt/pds_files/node0/
test_saswu.psds.26521.135470472.1065459831/test_saswu.sas7bdat
```

Set the APT_SAS_NO_PSDS_USTRING environment variable to output a header file that does not list ustring fields. This format is consistent with previous versions of WebSphere DataStage:

```
DataStage SAS data set
Lfile:
HOSTNAME:DIRECTORY/FILENAME
Lfile:
HOSTNAME:DIRECTORY/FILENAME
Lfile:
HOSTNAME:DIRECTORY/FILENAME
...
...
```

Note: The UstringFields line does not appear in the header file when any one of these conditions is met:

- You are using a version of WebSphere DataStage that is using a parallel engine prior to Orchestrate 7.1.
- Your data has no ustring schema values.

You have set the APT_SAS_NO_PSDS_USTRING environment variable. If one of these conditions is met, but your data contains multi-byte Unicode data, you must use the modify operator to convert the data schema type to ustring. This process is described in "Handling SAS char Fields Containing Multi-Byte Unicode Data".

To save your data as a parallel SAS data set, either add the framework argument [psds] to the output redirection or add the .psds extension to the outfile name on your osh command line. These two commands are equivalent, except that the resulting data sets have different file names. The first example outputs a file without the .psds extension and second example outputs a file with the .psds extension.

```
$ osh ". . . sas options > [psds] outfile "
$ osh ". . . sas options > outfile .psds "
```

You can likewise use a parallel SAS data set as input to the sas operator, either by specifying [psds] or by having the data set in a file with the .psds extension:

```
$ osh ". . . sas options < [psds] infile "
$ osh ". . . sas options < infile .psds "
```

A persistent parallel SAS data set is automatically converted to a WebSphere DataStage data set if the next operator in the flow is not a SAS interface operator.

SAS data set format

When data is being moved into or out of a parallel SAS interface operator, the data must be in a format that allows WebSphere DataStage to partition it to multiple processing nodes. For this reason, WebSphere DataStage provides an internal data set representation called an SAS data set, which is specifically intended for capturing the record structure of a SAS data set for use by the WebSphere DataStage SAS engine.

This is a non-persistent WebSphere DataStage version of the SAS data set format that is the default output from SAS interface operators. If the input data to be processed is in a SAS data set or a WebSphere DataStage data set, the conversion to an SAS data set is done automatically by the SAS interface operator.

In this format, each SAS record is represented by a single WebSphere DataStage raw field named SASdata:

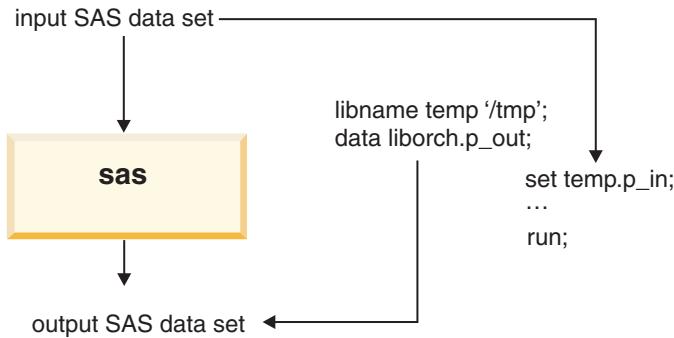
```
record ( SASdata:raw; )
```

Getting input from a SAS data set

Using a SAS data set in a sas operator entails reading it in from the library in which it is stored using a SAS step, typically, a DATA step. The data is normally then output to the liborch library in SAS data set format.

liborch is the SAS engine provided by WebSphere DataStage that you use to reference SAS data sets as inputs to and outputs from your SAS code. The sas operator converts a SAS data set or a standard WebSphere DataStage data set into the SAS data set format.

Here is a schematic of this process:



In this example

- libname temp '/tmp' specifies the library name of the SAS input
- data liborch.p_out specifies the output SAS data set
- temp.p_in is the input SAS data set.

The output SAS data set is named using the following syntax:

`liborch.osds_name`

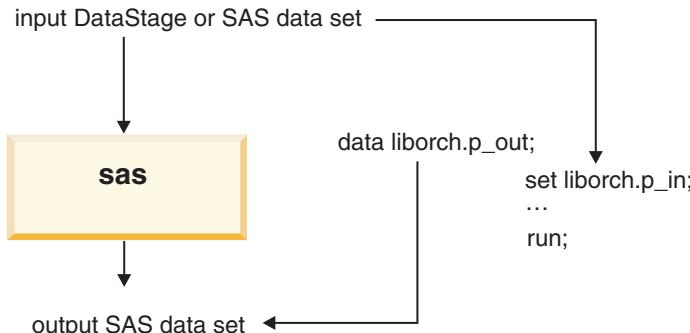
As part of writing the SAS code executed by the operator, you must associate each input and output data set with an input and output of the SAS code. In the figure above, the sas operator executes a DATA step with a single input and a single output. Other SAS code to process the data would normally be included.

In this example, the output data set is prefixed by liborch, a SAS library name corresponding to the WebSphere DataStage SAS engine, while the input data set comes from another specified library; in this example the data set file would normally be named /tmp/p_in.ssd01.

Getting input from a WebSphere DataStage data set or a SAS data set

You might have data already formatted as an WebSphere DataStage data set that you want to process with SAS code, especially if there are other WebSphere DataStage operators earlier in the data flow. In that case, the WebSphere DataStage SAS interface operators automatically convert a data set in the normal WebSphere DataStage data set format into an SAS data set. However, you still need to provide the SAS step that connects the data to the inputs and outputs of the sas operator.

For WebSphere DataStage data sets or SAS data sets the liborch library is referenced. The operator creates an SAS data set as output. Shown below is a sas operator with one input and one output data set:



As part of writing the SAS code executed by the operator, you must associate each input and output data set with an input and output of the SAS code. In the figure above, the sas operator executes a DATA step with a single input and a single output. Other SAS code to process the data would normally be included.

To define the input data set connected to the DATA step input, you use the -input option to the sas operator.

This option has the following osh format:

```
-input in_port_# ds_name
```

where:

- *ds_name* is the member name of a standard WebSphere DataStage data set or an SAS data set used as an input by the SAS code executed by the operator. You need only to specify the member name here; do not include any SAS library name prefix. You always include this member name in your sas operator SAS code, prefixed with liborch.
- *in_port_#* is an input data set number. Input data sets are numbered starting from 0.

You use the corresponding -output option to connect an output SAS data set to an output of the DATA step.

The osh syntax of -output is:

```
-output out_port_# ods_name
```

For the example above, the settings for -input and -output are:

```
$ osh "... sas -input Op_in  
-output Op_out other_options "
```

Converting between data set types

You might have an existing WebSphere DataStage data set, which is the output of an upstream WebSphere DataStage operator or is the result of reading data from a database such as INFORMIX or Oracle using a WebSphere DataStage database read operator. Also, the WebSphere DataStage database write operators, like most WebSphere DataStage operators, take as input standard WebSphere DataStage data sets. Therefore, the SAS interface operators must convert WebSphere DataStage data to SAS data and also convert SAS data to WebSphere DataStage data. These two kinds of data-set conversion are covered in the following sections.

Converting WebSphere DataStage data to SAS data

Once you have provided the liborch statements to tell the sas operator where to find the input data set, as described in "Getting Input from a WebSphere DataStage Data Set or an SAS Data Set", the operator automatically converts the WebSphere DataStage data set into an SAS data set before executing your SAS code. In order to do so, a SAS interface operator must convert both the field names and the field data types in the input WebSphere DataStage data set.

WebSphere DataStage supports 32-character field and data set names for SAS Version 8. However, SAS Version 6 accepts names of only up to eight characters. Therefore, you must specify whether you are using SAS Version 6 or SAS Version 8 when you install WebSphere DataStage. When you specify SAS Version 6, WebSphere DataStage automatically truncates names to eight characters when converting a standard WebSphere DataStage data set to an SAS data set.

The SAS interface operators convert WebSphere DataStage data types to corresponding SAS data types. All WebSphere DataStage numeric data types, including integer, float, decimal, date, time, and timestamp, are converted to the SAS numeric data type. WebSphere DataStage raw and string fields are converted to SAS string fields. WebSphere DataStage raw and string fields longer than 200 bytes are truncated to 200 bytes, which is the maximum length of a SAS string.

The following table summarizes these data type conversions:

Table 142. Conversions to SAS Data Types

WebSphere DataStage Data Type	SAS Data Type
date	SAS numeric date value
decimal[<i>p,s</i>] (<i>p</i> is precision and <i>s</i> is scale)	SAS numeric
int64 and uint64	not supported
int8, int16, int32, uint8, uint16, uint32,	SAS numeric
sfloat, dfloat	SAS numeric
fixed-length raw, in the form: raw[<i>n</i>]	SAS string with length <i>n</i> The maximum string length is 200 bytes; strings longer than 200 bytes are truncated to 200 bytes.
variable-length raw, in the form: raw[max= <i>n</i>]	SAS string with a length of the actual string length The maximum string length is 200 bytes; strings longer than 200 bytes are truncated to 200 bytes.
variable-length raw in the form: raw	not supported
fixed-length string or ustring, in the form: string[<i>n</i>] or ustring[<i>n</i>]	SAS string with length <i>n</i> The maximum string length is 200 bytes; strings longer than 200 bytes are truncated to 200 bytes.
variable-length string or ustring, in the form: string[max= <i>n</i>] or ustring[max= <i>n</i>]	SAS string with a length of the actual string length The maximum string length is 200 bytes; strings longer than 200 bytes are truncated to 200 bytes.
variable-length string or ustring in the form: string or ustring	not supported
time	SAS numeric time value
timestamp	SAS numeric datetime value

Converting SAS data to WebSphere DataStage data

After your SAS code has run, it might be necessary for the SAS interface operator to convert an output SAS data set to the standard WebSphere DataStage data-set format. This is required if you want to further process the data using the standard WebSphere DataStage operators, including writing to a database using a WebSphere DataStage database write operator.

In order to do this, the operator needs a record schema for the output data set. You can provide a schema for the WebSphere DataStage data set in one of two ways:

- Using the `-schema` option or the `-schemaFile` option of the `sasout` operator, you can define a schema definition for the output data set.
- A persistent parallel SAS data set is automatically converted to a WebSphere DataStage data set if the next operator in the flow is not a SAS interface operator.

When converting a SAS numeric field to a WebSphere DataStage numeric, you can get a numeric overflow or underflow if the destination data type is too small to hold the value of the SAS field. You can avoid the error this causes by specifying all fields as nullable so that the destination field is simply set to null and processing continues.

a WebSphere DataStage example

A Specialty Freight Carrier charges its customers based on distance, equipment, packing, and license requirements. They need a report of distance traveled and charges for the month of July grouped by license type.

Here is the osh syntax for this example:

```
osh "sas -output 0 in_data
    -source 'libname shipments \"/july/distances/input\";
        data liborch.in_data;
        set shipments.ship_data;
        if substr(SHIPDATE,1,3)=\"Jul\";
        run;'

[seq]

hash key LICENSE

sas -input 0 in_data
    -source "proc sort data=liborch.in_data
        out=liborch.out_sort;
        by LICENSE;
        run;"

    -output 0 out_sort

sas -input 0 out_sort
    -source 'proc means data=liborch.out_sort
        N MEAN SUM;
        by LICENSE;
        var DISTANCE CHARGE;
        run;'

    -options nocenter"
```

The table below shows some representative input data:

Ship Date	District	Distance	Equipment	Packing	License	Charge
...						
Jun 2 2000	1	1540	D	M	BUN	1300
Jul 12 2000	1	1320	D	C	SUM	4800
Aug 2 2000	1	1760	D	C	CUM	1300
Jun 22 2000	2	1540	D	C	CUN	13500
Jul 30 2000	2	1320	D	M	SUM	6000
...						

Here is the SAS output:

```
The SAS System
17:39, October 26, 2002
...
LICENSE=SUM
Variable   Label      N      Mean       Sum
-----
DISTANCE   DISTANCE    720    1563.93    1126030.00
CHARGE     CHARGE     720    28371.39   20427400.00
...
Step execution finished with status = OK.
```

Parallelizing SAS steps

This section describes how to write SAS applications for parallel execution. Parallel execution allows your SAS application and data to be distributed to the multiple processing nodes within a scalable system.

The first section describes how to execute SAS DATA steps in parallel; the second section describes how to execute SAS PROC steps in parallel. A third section provides rules of thumb that might be helpful for parallelizing SAS code.

Executing DATA steps in parallel

This section describes how to convert a sequential SAS DATA step into a parallel DATA step.

Characteristics of DATA steps that are candidates for parallelization using the round robin partitioning method include:

- Perform the same action for every record
- No RETAIN, LAGN, SUM, or + statements
- Order does not need to be maintained.

Characteristics of DATA steps that are candidates for parallelization using hash partitioning include:

- BY clause
- Summarization or accumulation in retained variables
- Sorting or ordering of inputs.

Often, steps with these characteristics require that you group records on a processing node or sort your data as part of a preprocessing operation.

Some DATA steps should not be parallelized. These include DATA steps that:

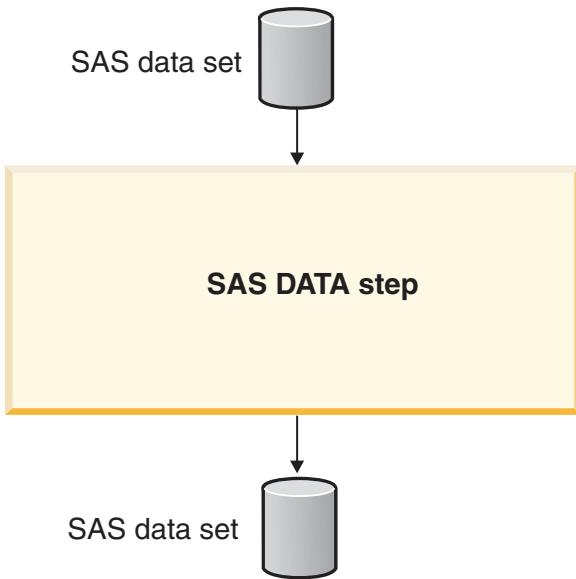
- Process small amounts of data (startup overhead outweighs parallel speed improvement)
- Perform sequential operations that cannot be parallelized (such as a SAS import from a single file)
- Need to see every record to produce a result

Parallelizing these kinds of steps would offer little or no advantage to your code. Steps of these kinds should be executed within a sequential WebSphere DataStage sas operator.

Example applications

Example 1: parallelizing a SAS data step

This section contains an example that executes a SAS DATA step in parallel. Here is a figure describing this step:



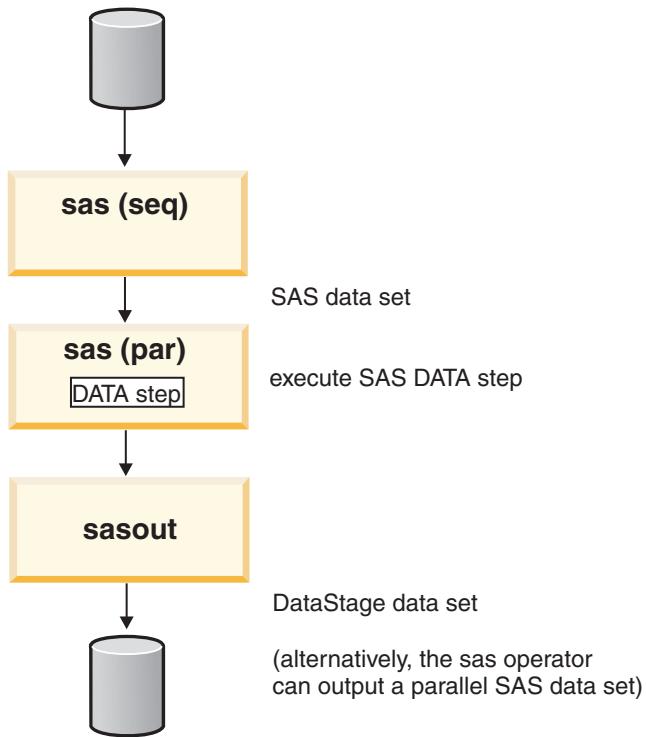
The step takes a single SAS data set as input and writes its results to a single SAS data set as output. The DATA step recodes the salary field of the input data to replace a dollar amount with a salary-scale value. Here is the original SAS code:

```

libname prod "/prod";
data prod.sal_scl;
  set prod.sal;
  if (salary < 10000)
    then salary = 1;
  else if (salary < 25000)
    then salary = 2;
  else if (salary < 50000)
    then salary = 3;
  else if (salary < 100000)
    then salary = 4;
  else salary = 5;
run;
  
```

This DATA step requires little effort to parallelize because it processes records without regard to record order or relationship to any other record in the input. Also, the step performs the same operation on every input record and contains no BY clauses or RETAIN statements.

The following figure shows the WebSphere DataStage data flow diagram for executing this DATA step in parallel:



In this example, you:

- Get the input from a SAS data set using a sequential sas operator;
- Execute the DATA step in a parallel sas operator;
- Output the results as a standard WebSphere DataStage data set (you must provide a schema for this) or as a parallel SAS data set. You might also pass the output to another sas operator for further processing. The schema required might be generated by first outputting the data to a Parallel SAS data set, then referencing that data set. WebSphere DataStage automatically generates the schema.

The first sequential sas operator executes the following SAS code as defined by the -source option to the operator:

```
libname prod "/prod";
data liborch.out;
  set prod.sal;
run;
```

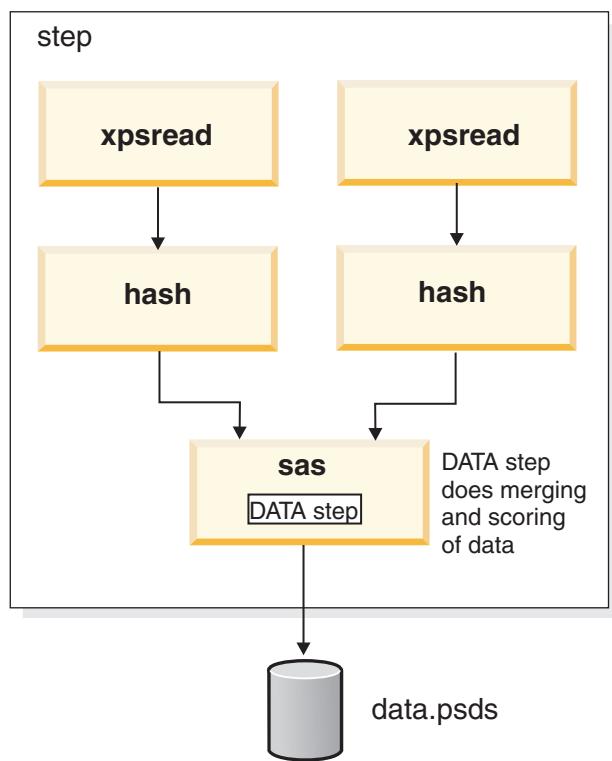
This parallel sas operator executes the following SAS code:

```
libname prod "/prod";
data liborch.p_sal;
  set liborch.sal;
  . . . (salary field code from previous page)
run;
```

The sas operator can then do one of three things: use the sasout operator with its -schema option to output the results as a standard WebSphere DataStage data set, output the results as a Parallel SAS data set, or pass the output directly to another sas operator as an SAS data set. The default output format is SAS data set. When the output is to a Parallel SAS data set or to another sas operator, for example, as a standard WebSphere DataStage data set or a Parallel SAS data set, the liborch statement must be used. Conversion of the output to a standard WebSphere DataStage data set or a Parallel SAS data set is discussed in "SAS Data Set Format" and "Parallel SAS Data Set Format".

Example 2: using the hash partitioner with a SAS DATA step

This example reads two INFORMIX tables as input, hash partitions on the workloc field, then uses a SAS DATA step to merge the data and score it before writing it out to a parallel SAS data set.



The sas operator in this example runs the following DATA step to perform the merge and score:

```
data liborch.emptabd;
  merge liborch.wltab liborch.emptab;
  by workloc;
  a_13 = (f1-f3)/2;
  a_15 = (f1-f5)/2;
  .
  .
  .
run;
```

Records are hashed based on the workloc field. In order for the merge to work correctly, all records with the same value for workloc must be sent to the same processing node and the records must be ordered. The merge is followed by a parallel WebSphere DataStage sas operator that scores the data, then writes it out to a parallel SAS data set.

Example 3: using a SAS SUM statement

This section contains an example using the SUM clause with a DATA step. In this example, the DATA step outputs a SAS data set where each record contains two fields: an account number and a total transaction amount. The transaction amount in the output is calculated by summing all the deposits and withdrawals for the account in the input data where each input record contains one transaction.

Here is the SAS code for this example:

```
libname prod "/prod";
proc sort data=prod.trans;
  out=prod.s_trans
```

```

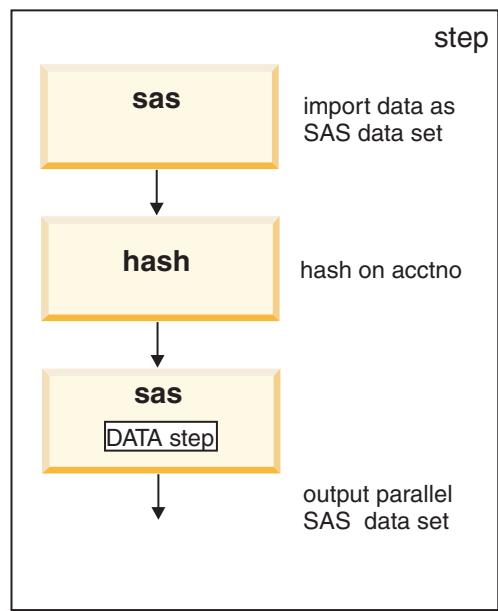
by acctno;
data prod.up_trans (keep = acctno sum);
set prod.s_trans;
by acctno;
if first.acctno then sum=0;
if type = "D"
  then sum + amt;
if type = "W"
  then sum - amt;
if last.acctno then output;
run;

```

The SUM variable is reset at the beginning of each group of records where the record groups are determined by the account number field. Because this DATA step uses the BY clause, you use the WebSphere DataStage hash partitioning operator with this step to make sure all records from the same group are assigned to the same node.

Note that DATA steps using SUM without the BY clause view their input as one large group. Therefore, if the step used SUM but not BY, you would execute the step sequentially so that all records would be processed on a single node.

Shown below is the data flow diagram for this example:



The SAS DATA step executed by the second sas operator is:

```

data liborch.nw_trans (keep = acctno sum);
  set liborch.p_trans;
  by acctno;
  if first.acctno then sum=0;

```

```

if type = "D"
  then sum + amt;
if type = "W"
  then sum - amt;
if last.acctno then output;
run;

```

Executing PROC steps in parallel

This section describes how to parallelize SAS PROC steps using WebSphere DataStage. Before you parallelize a PROC step, you should first determine whether the step is a candidate for parallelization.

When deciding whether to parallelize a SAS PROC step, you should look for those steps that take a long time to execute relative to the other PROC steps in your application. By parallelizing only those PROC steps that take the majority of your application's execution time, you can achieve significant performance improvements without having to parallelize the entire application. Parallelizing steps with short execution times might yield only marginal performance improvements.

Many of the characteristics that mark a SAS DATA step as a good candidate for parallelization are also true for SAS PROC steps. Thus PROC steps are likely candidates if they:

- do not require sorted inputs
- perform the same operation on all records.

PROC steps that generate human-readable reports might also not be candidates for parallel execution unless they have a BY clause. You might want to test this type of PROC step to measure the performance improvement with parallel execution.

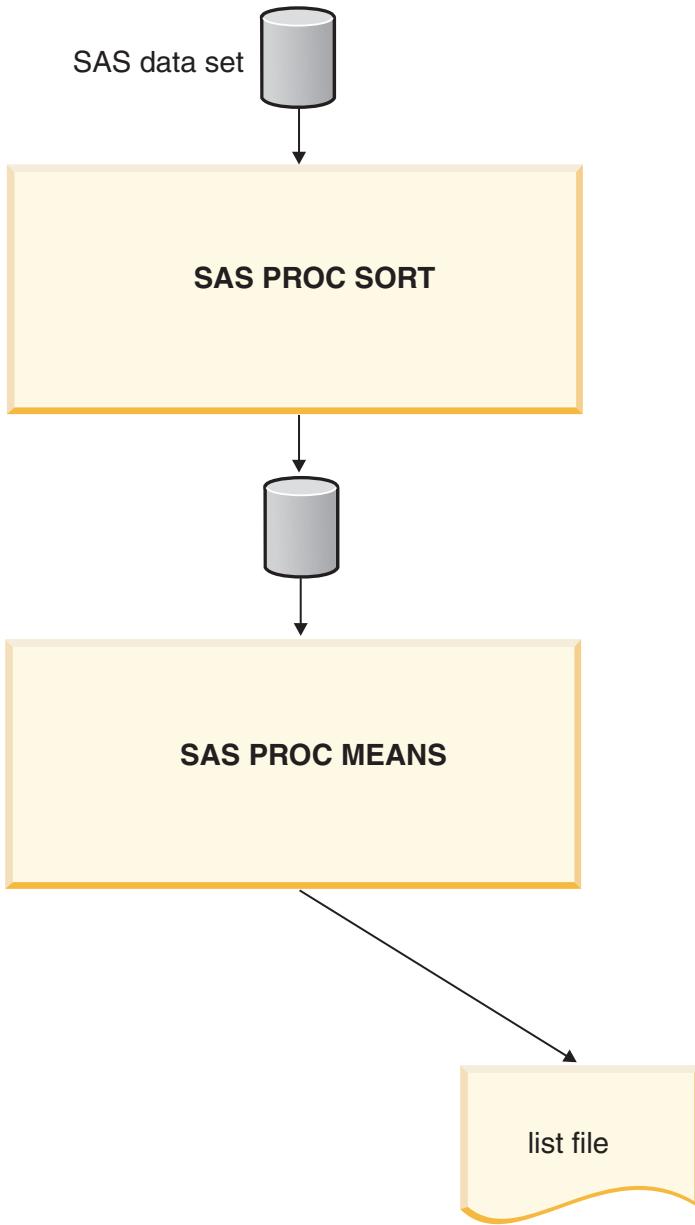
The following section contains two examples of running SAS PROC steps in parallel.

Example applications

Example 1: parallelize PROC steps using the BY keyword

This example parallelizes a SAS application using PROC SORT and PROC MEANS. In this example, you first sort the input to PROC MEANS, then calculate the mean of all records with the same value for the acctno field.

The following figure illustrates this SAS application:



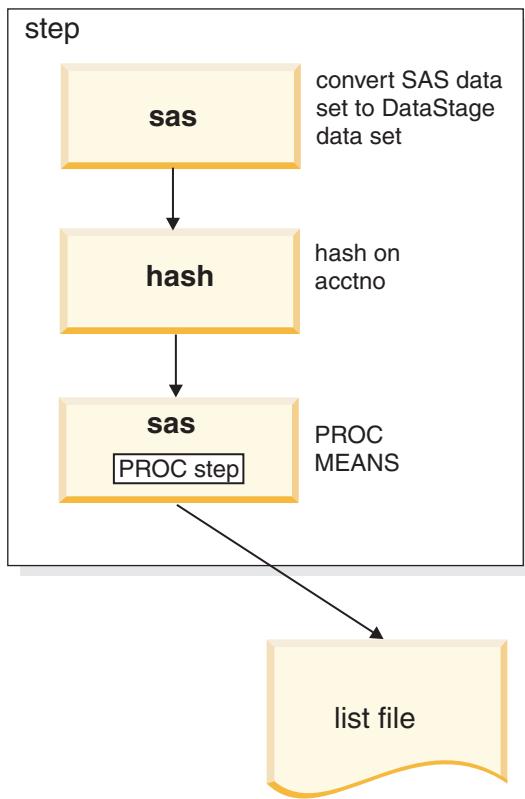
Shown below is the original SAS code:

```

libname prod "/prod";
proc means data=prod.dhist;
  BY acctno;
run;
  
```

The BY clause in a SAS step signals that you want to hash partition the input to the step. Hash partitioning guarantees that all records with the same value for acctno are sent to the same processing node. The SAS PROC step executing on each node is thus able to calculate the mean for all records with the same value for acctno.

Shown below is the WebSphere DataStage implementation of this example:



PROC MEANS pipes its results to standard output, and the sas operator sends the results from each partition to standard output as well. Thus the list file created by the sas operator, which you specify using the -listds option, contains the results of the PROC MEANS sorted by processing node.

Shown below is the SAS PROC step for this application:

```
proc means data=liborch.p_dhist;
  by acctno;
run;
```

Example 2: parallelizing PROC steps using the CLASS keyword

One type of SAS BY GROUP processing uses the SAS keyword CLASS. CLASS allows a PROC step to perform BY GROUP processing without your having to first sort the input data to the step. Note, however, that the grouping technique used by the SAS CLASS option requires that all your input data fit in memory on the processing node.

Note also that as your data size increases, you might want to replace CLASS and NWAY with SORT and BY.

Whether you parallelize steps using CLASS depends on the following:

- If the step also uses the NWAY keyword, parallelize it.
- When the step specifies both CLASS and NWAY, you parallelize it just like a step using the BY keyword, except the step input doesn't have to be sorted. This means you hash partition the input data based on the fields specified to the CLASS option. See the previous section for an example using the hash partitioning method.
- If the CLASS clause does not use NWAY, execute it sequentially.
- If the PROC STEP generates a report, execute it sequentially, unless it has a BY clause.

For example, the following SAS code uses PROC SUMMARY with both the CLASS and NWAY keywords:

```

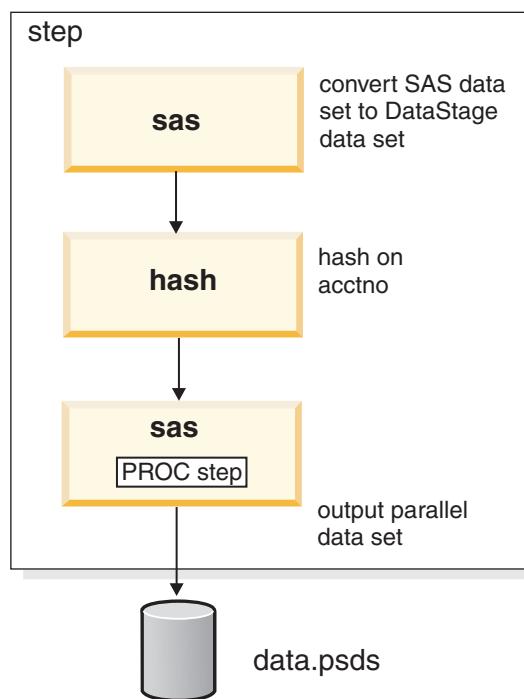
libname prod "/prod";
proc summary data=prod.txdlst
  missing NWAY;
  CLASS acctno lstrxd fpzd;
  var xdamt xdcnt;
  output out=prod.xnlstr(drop=_type_ _freq_) sum=;
run;

```

In order to parallelize this example, you hash partition the data based on the fields specified in the CLASS option. Note that you do not have to partition the data on all of the fields, only to specify enough fields that your data is correctly distributed to the processing nodes.

For example, you can hash partition on acctno if it ensures that your records are properly grouped. Or you can partition on two of the fields, or on all three. An important consideration with hash partitioning is that you should specify as few fields as necessary to the partitioner because every additional field requires additional overhead to perform partitioning.

The following figure shows the WebSphere DataStage application data flow for this example:



The SAS code (DATA step) for the first sas operator is:

```

libname prod "/prod";
data liborch.p_txdlst
  set prod.txdlst;
run;

```

The SAS code for the second sas operator is:

```

proc summary data=liborch.p_txdlst
  missing NWAY;
  CLASS acctno lstrxd fpzd;
  var xdamt xdcnt;
  output out=liborch.p_xnlstr(drop=_type_ _freq_) sum=;
run;

```

Some points to consider in parallelizing SAS code

Which SAS programs benefit from parallelization?

There are four basic points you should consider when parallelizing your SAS code. If your SAS application satisfies any of the following criteria it is probably a good candidate for parallelization, provided the application is run against large volumes of data. SAS applications that run quickly, against small volumes of data, do not benefit.

1. Does your SAS application extract a large number of records from a parallel relational database? A few million records or more is usually sufficiently large to offset the cost of initializing parallelization.
2. Is your SAS program CPU intensive? CPU-intensive applications typically perform multiple CPU-demanding operations on each record. Operations that are CPU-demanding include arithmetic operations, conditional statements, creation of new field values for each record, and so on. For SMP users, WebSphere DataStage provides you with the biggest performance gains if your code is CPU-intensive.
3. Does your SAS program pass large records of lengths greater than 100 bytes? WebSphere DataStage introduces a small record-size independent CPU overhead when passing records into and out of the sas operator. You might notice this overhead if you are passing small records that also perform little or no CPU-intensive operations.
4. Does your SAS program perform sorts? Sorting is a memory-intensive procedure. By performing the sort in parallel, you can reduce the overall amount of required memory. This is always a win on an MPP platform and frequently a win on an SMP.

Rules of thumb

There are rules of thumb you can use to specify how a SAS program runs in parallel. Once you have identified a program as a potential candidate for use in WebSphere DataStage, you need to determine how to divide the SAS code itself into WebSphere DataStage steps.

The sas operator can be run either parallel or sequentially. Any converted SAS program that satisfies one of the four criteria outlined above will contain at least one parallel segment. How much of the program should be contained in this segment? Are there portions of the program that need to be implemented in sequential segments? When does a SAS program require multiple parallel segments? Here are some guidelines you can use to answer such questions.

1. Identify the slow portions of the sequential SAS program by inspecting the CPU and real-time values for each of the PROC and DATA steps in your application. Typically, these are steps that manipulate records (CPU-intensive) or that sort or merge (memory-intensive). You should be looking for times that are a relatively large fraction of the total run time of the application and that are measured in units of many minutes to hours, not seconds to minutes. You might need to set the SAS fullstimer option on in your config.sas612 or in your SAS program itself to generate a log of these sequential run times.
2. Start by parallelizing only those slow portions of the application that you have identified in Step 1 above. As you include more code within the parallel segment, remember that each parallel copy of your code (referred to as a partition) sees only a fraction of the data. This fraction is determined by the partitioning method you specify on the input or inpipe lines of your sas operator source code.
3. Any two sas operators should only be connected by one pipe. This ensures that all pipes in the WebSphere DataStage program are simultaneously flowing for the duration of the execution. If two segments are connected by multiple pipes, each pipe must drain entirely before the next one can start.
4. Keep the number of sas operators to a minimum. There is a performance cost associated with each operator that is included in the data flow. Rule 3 takes precedence over this rule. That is, when reducing the number of operators means connecting any two operators with more than one pipe, don't do it.
5. If you are reading or writing a sequential file such as a flat ASCII text file or a SAS data set, you should include the SAS code that does this in a sequential sas operator. Use one sequential operator

- for each such file. You will see better performance inputting one sequential file per operator than if you lump many inputs into one segment followed by multiple pipes to the next segment, in line with Rule 2 above.
6. When you choose a partition key or combination of keys for a parallel operator, you should keep in mind that the best overall performance of the parallel application occurs if each of the partitions is given approximately equal quantities of data. For instance, if you are hash partitioning by the key field year (which has five values in your data) into five parallel segments, you will end up with poor performance if there are big differences in the quantities of data for each of the five years. The application is held up by the partition that has the most data to process. If there is no data at all for one of the years, the application will fail because the SAS process that gets no data will issue an error statement. Furthermore, the failure will occur only after the slowest partition has finished processing, which might be well into your application. The solution might be to partition by multiple keys, for example, year and storeNumber, to use roundrobin partitioning where possible, to use a partitioning key that has many more values than there are partitions in your WebSphere DataStage application, or to keep the same key field but reduce the number of partitions. All of these methods should serve to balance the data distribution over the partitions.
 7. Multiple parallel segments in your WebSphere DataStage application are required when you need to parallelize portions of code that are sorted by different key fields. For instance, if one portion of the application performs a merge of two data sets using the patientID field as the BY key, this PROC MERGE will need to be in a parallel segment that is hash partitioned on the key field patientID. If another portion of the application performs a PROC MEANS of a data set using the procedureCode field as the CLASS key, this PROC MEANS will have to be in a parallel sas operator that is hash partitioned on the procedureCode key field.
 8. If you are running a query that includes an ORDER BY clause against a relational database, you should remove it and do the sorting in parallel, either using SAS PROC SORT or a WebSphere DataStage input line order statement. Performing the sort in parallel outside of the database removes the sequential bottleneck of sorting within the RDBMS.
 9. A sort that has been performed in a parallel operator will order the data only within that operator. If the data is then streamed into a sequential operator, the sort order will be lost. You will need to re-sort within the sequential operator to guarantee order.
 10. Within a parallel sas operator you might only use SAS work directories for intermediate writes to disk. SAS generates unique names for the work directories of each of the parallel operators. In an SMP environment this is necessary because it prevents the multiple CPUs from writing to the same work file. Do not use a custom-specified SAS library within a parallel operator.
 11. Do not use a liborch directory within a parallel segment unless it is connected to an inpipe or an outpipe. A liborch directory might not be both written and read within the same parallel operator.
 12. A liborch directory can be used only once for an input, inpipe, output or outpipe. If you need to read or write the contents of a liborch directory more than once, you should write the contents to disk via a SAS work directory and copy this as needed.
 13. Remember that all WebSphere DataStage operators in a step run simultaneously. This means that you cannot write to a custom-specified SAS library as output from one WebSphere DataStage operator and simultaneously read from it in a subsequent operator. Connections between operators must be via WebSphere DataStage pipes which are virtual data sets normally in SAS data set format.

Using SAS with European languages

Use basic SAS to perform European-language data transformations, not international SAS which is designed for languages that require double-byte character transformations.

The basic SAS executable sold in Europe might require you to create workarounds for handling some European characters. WebSphere DataStage has not made any modifications to SAS that eliminate the need for this special character handling.

To use WebSphere DataStage with European languages:

1. Specify the location of your basic SAS executable using one of the methods below. The SAS international syntax in these methods directs WebSphere DataStage to perform European-language transformations.

- Set the APT_SASINT_COMMAND environment variable to the absolute path name of your basic SAS executable. For example:

```
APT_SASINT_COMMAND /usr/local/sas/sas8.2/sas
```

- Include a resource sas entry in your configuration file. For example:

```
resource sasint "[/usr/sas612/]" { }
```

2. Using the -sas_cs option of the SAS-interface operators, specify an ICU character set to be used by WebSphere DataStage to map between your ustring values and the char data stored in SAS files. If your data flow also includes the sasin or the sasout operator, specify the same character set you specified to the sas operator. The syntax is:

```
-sas_cs icu_character_set
```

For example:

```
-sas_cs fr_CA-iso-8859
```

3. In the table of your sascstxt file, enter the designations of the ICU character sets you use in the right columns of the table. There can be multiple entries. Use the left column of the table to enter a symbol that describes the character sets. The symbol cannot contain space characters, and both columns must be filled in. The platform-specific sascstxt files are in:

```
$APT_ORCHHOME/apt/etc/ platform /
```

The platform directory names are: sun, aix, hpx , and lunix. For example:

```
$APT_ORCHHOME/etc/aix/sascstxt
```

Example table entries:

CANADIAN_FRENCH	
	fr_CA-iso-8859

ENGLISH	
	ISO-8859-5

Using SAS to do ETL

Only a simple SAS step is required to extract data from a SAS file. *The Little SAS Book* from the SAS Institute provides a good introduction to the SAS step language.

In the following example, SAS is directed to read the SAS file cl_ship, and to deliver that data as a WebSphere DataStage data stream to the next WebSphere DataStage step. In this example, the next step consists of the peek operator.

See "Representing SAS and Non-SAS Data in WebSphere DataStage" for information on data-set formats, and "Getting Input from a SAS Data Set" for a description of liborch.

The -schemaFile option instructs WebSphere DataStage to generate the schema that defines the WebSphere DataStage virtual data stream from the meta data in the SAS file cl_ship.

```
osh sas -source 'libname curr_dir \'.';  
      DATA liborch.out_data;  
      SET curr_dir.cl_ship;  
      RUN;  
      -output 0 out_data -schemaFile 'cl_ship'  
      -workingdirectory '.'  
      [seq] 0> Link2.v;  
      peek -all [ seq] 0< Link2.v;
```

If you know the fields contained in the SAS file, the step can be written as:

```

osh sas -source 'libname curr_dir \'.\'';
      DATA liborch.out_data;
      SET curr_dir.cl_ship;
      RUN;
      -output 0 out_data
      -schema record(SHIP_DATE:nullable string[50];
                      DISTRICT:nullable string[10];
                      DISTANCE:nullable sfloat;
                      EQUIPMENT:nullable string[10];
                      PACKAGING:nullable string[10];
                      LICENSE:nullable string[10];
                      CHARGE:nullable sfloat;
      [seq] 0> DSLink2.v;
      peek -all [seq] 0< DSLink2.v;

```

It is also easy to write a SAS file from a WebSphere DataStage virtual datastream. The following osh commands generated the SAS file described above.

```

osh import
-schema record {final_delim=end, delim=',', quote=double}
  (Ship_Date:string[max=50];
   District:string[max=10];
   Distance:int32;
   Equipment:string[max=10];
   Packaging:string[max=10];
   License:string[max=10];
   Charge:int32)
-file '/home/cleonard/sas_in.txt'
-rejects continue
-reportProgress yes
0> DSLink2a.v;

osh sas -source 'libname curr_dir \'.\'';
      DATA curr_dir.cl_ship;
      SET liborch.in_data;
      RUN;
      -input 0 in_data  [seq] 0< DSLink2a.v;

```

The SAS interface operators

The sasin and sasout operators provide some options that the sas operator does not provide. However, if your application does not require that schema fields be dropped or modified, it is not necessary to explicitly include the sasin or sasout operators in your data flow. You can simply specify the -input or -output options to the sas operator, and WebSphere DataStage automatically propagates the appropriate option values and commands of the sasin and sasout operators.

However, the -sas_cs option is not propagated when you explicitly include the sasin or sasout operators. If your WebSphere DataStage data includes ustring values, see the next section, "Specifying a Character Set and SAS Mode" below for information on handling ustring values consistently.

Specifying a character set and SAS mode

Using -sas_cs to specify a character set

If your WebSphere DataStage data includes ustring values, you can specify what character set WebSphere DataStage uses to map between your ustring values and the char data stored in SAS files. You use the -sas_cs option of the sas operator to indicate your character-set choice. Setting this option also causes SAS to be invoked in international mode unless you have configured your system for European languages as described in "Using SAS with European Languages".

See the next section, "Using -sas_cs to Determine SAS mode" for information on modes.

Note: If your data flow also includes the sasin or the sasout operator, use the -sas_cs option of these operators to specify the same character set you specified to the sas operator.

The sasin and sas operators use the -sas_cs option to determine what character-set encoding should be used when exporting ustring (UTF-16) values to SAS. The sas and sasout operators use this option to determine what character set WebSphere DataStage uses to convert SAS char data to WebSphere DataStage ustring values.

The syntax for the -sas_cs option is:

`-sas_cs icu_character_set | DBCSLANG`

In \$APT_ORCHHOME/apt/etc/platform/ there are platform-specific sascstxt files that show the default ICU character set for each DBCSLANG setting. The platform directory names are: sun, aix, hpx , and lunix. For example:

\$APT_ORCHHOME/etc/aix/sascstxt

When you specify a character setting, the sascstxt file must be located in the platform-specific directory for your operating system. WebSphere DataStage accesses the setting that is equivalent to your -sas_cs specification for your operating system. ICU settings can differ between platforms.

For the DBCSLANG settings you use, enter your ICU equivalents.

DBCSLANG Setting

ICU Character Set

JAPANESE

icu_character_set

KATAKANA

icu_character_set

KOREAN

icu_character_set

HANGLE

icu_character_set

CHINESE

icu_character_set

TAIWANESE

icu_character_set

Note: If WebSphere DataStage encounters a ustring value but you have not specified the -sas_cs option, SAS remains in standard mode and WebSphere DataStage then determines what character setting to use when it converts your ustring values to Latin1 characters.

WebSphere DataStage first references your APT_SAS_CHARSET environment variable. If it is not set and your APT_SAS_CHARSET_ABORT environment variable is also not set, it uses the value of your -impexp_charset option or the value of your APT_IMPEXP_CHARSET environment variable.

Using -sas_cs to determine SAS mode

Specifying the -sas_cs option also tells WebSphere DataStage to invoke SAS in international mode. In this mode, the SAS DBCS, DBCSLANG, and DBCSTYPE environment variables are set, and SAS processes your ustring data and step source code using multi-byte characters in your chosen character set. Multi-byte Unicode character sets are supported in SAS char fields, but not in SAS field names or data set file names.

Your SAS system is capable of running in DBCS mode if your SAS log output has this type of header:

NOTE: SAS (r) Proprietary Software Release 8.2 (TS2MO DBCS2944)

If you do not specify the -sas_cs option for the sas operator, SAS is invoked in standard mode. In standard mode, SAS processes your string fields and step source code using single-byte LATIN1 characters. SAS standard mode, is also called "the basic US SAS product". When you have invoked SAS in standard mode, your SAS log output has this type of header:

NOTE: SAS (r) Proprietary Software Release 8.2 (TS2MO)

Parallel SAS data sets and SAS International

Automatic WebSphere DataStage step insertion

When you save a .ds data set as a parallel SAS data set by using the [psds] directive or adding a .psds suffix to the output file, WebSphere DataStage automatically inserts a step in your data flow which contains the sasin, sas, and sasout operators. Using the -source option, the sas operator specifies specialized SAS code to be executed by SAS.

The SAS executable used is the one on your \$PATH unless its path is overwritten by the APT_SAS_COMMAND or APT_SASINT_COMMAND environment variable. WebSphere DataStage SAS-specific variables are described in "Environment Variables" . The DBCS, DBCSLANG, and DBCSTYLE environment variables are not set for the step.

If the WebSphere DataStage-inserted step fails, you can see the reason for its failure by rerunning the job with the APT_SAS_SHOW_INFO environment variable set.

For more information on Parallel SAS Data Sets, see "Parallel SAS Data Set Format" .

Handling SAS char fields containing multi-byte Unicode data

When WebSphere DataStage writes a .psds data set, it must reference your ICU character setting when it encounters a ustring field. WebSphere DataStage determines your character setting by first referencing your APT_SAS_CHARSET environment variable. If this variable is not set and your APT_SAS_CHARSET_ABORT environment variable is also not set, it uses the value of your -impexp_charset option or the value of the APT_IMPEXP_CHARSET environment variable.

When you set the APT_SAS_NO_PSDS_USTRING environment variable, WebSphere DataStage imports all SAS char fields as WebSphere DataStage string fields when reading a .psds data set. Use the modify operator to import those char fields that contain multi-byte Unicode characters as ustrings, and specify a character set to be used for the ustring value.

The syntax for this modify operator conversion specification is:

```
destField [: data type ]=cstring_from_string[ charset ]
( sourceField )
```

Here is the conversion specification used in an example osh command:

```
osh "modify 'b_field:cstring[10]= cstring_from_string[ISO-2022-JP](b_field)' < [psds] test_sasw.psds
| peek -name"
```

When the APT_SAS_NO_PSDS_USTRING environment variable is not set, the .psds header file lists the SAS char fields that are to be converted to ustring fields in WebSphere DataStage, making it unnecessary to use the modify operator to convert the data type of these fields. For more information, see "Parallel SAS Data Set Format" .

Specifying an output schema

You must specify an output schema to the sas and sasout operators when the downstream operator is a standard WebSphere DataStage operator such as peek or copy. WebSphere DataStage uses the schema to convert the SAS data used by sas or sasout to a WebSphere DataStage data set suitable for processing by a standard WebSphere DataStage operator.

There are two ways to specify the schema:

- Use the -schema option to supply a WebSphere DataStage schema. By supplying an explicit schema, you obtain better job performance and gain control over which SAS char data is to be output as ustring values and which SAS char data is to be output as string values.
- Use the -schemaFile option to specify a SAS file that has the same meta data description as the SAS output stream. WebSphere DataStage generates the schema from that file.

Use this syntax:

```
-schemaFile schema_file_name -debug yes
```

Specifying -debug yes displays the generated schema. For example:

```
..<APT_SAS_OutputOperator> output schema from
SAS file's MetaData is;
record(
  S: nullable string[8];
  B: nullable string[8];
)
```

You can use the APT_SAS_SCHEMASOURCE_DUMP environment variable to see the SAS CONTENTS report used by the -schemaFile option. The output also displays the command line given to SAS to produce the report and the pathname of the report. The input file and output file created by SAS is not deleted when this variable is set.

You can also obtain a WebSphere DataStage schema by using the -schema option of the sascontents operator. Use the peek operator in sequential mode to print the schema to the screen. Example output is:

```
...<peek,0>Suggested schema for DataStage SAS data set 'cow'
...<peek,0>record (a : nullable dfloat;
...<peek,0>           b : nullable string[10])
```

You can then fine tune the schema and specify it to the -schema option to take advantage of performance improvements.

Note: If both the -schemaFile option and the -sas_cs option are set, all of your SAS char fields are converted to WebSphere DataStage ustring values. If the -sas_cs option is not set, all of your SAS char values are converted to WebSphere DataStage string values.

To obtain a mixture of string and ustring values use the -schema option.

Controlling ustring truncation

Your ustring values might be truncated before the space pad characters and \0 because a ustring value of n characters does not fit into n bytes of a SAS char value. You can use the APT_SAS_TRUNCATION environment variable to specify how the truncation is done. This variable is described in "Environment Variables".

If the last character in a truncated value is a multi-byte character, the SAS char field is padded with C null (zero) characters. For all other values, including non-truncated values, spaces are used for padding.

You can avoid truncation of your ustring values by specifying them in your schema as variable-length strings with an upper bound. The syntax is:

```
ustring[max= n_codepoint_units ]
```

Specify a value for n_codepoint_units that is 2.5 or 3 times larger than the number of characters in the ustring. This forces the SAS char fixed-length field size to be the value of n_codepoint_units. The maximum value for n_codepoint_units is 200.

Generating a proc contents report

To obtain a report similar to the report that is generated by the SAS procedure PROC CONTENTS for a self-describing data set, use one of these methods:

- Specify the -report option of the sasin or sasout operator.
- Specify -debug yes to the sasin, sas, or sasout operator.
- Insert the sascontents operator in your data flow.

Use the peek operator in sequential mode to print the report to the screen.

Here is an example report:

```
...2:11:50(000) <APT_SAS_Driver in sas,0> input dataset 0, SAS library member moo  
APT_SAS_Driver in sas (APT_SAS_Driver):  
Field Count: 12 Record Length: 96 Created: 2003-05-13 16:11:47 Version: 4
```

Number	Name	Type	Len	Offset	Format	Label
1	AGE	Num	8	0	4.	age
2	CAR	Char	8	8		car
3	CHILDREN	Num	8	16	4.	children
4	CUR_ACT	Char	8	24		cur_act
5	ID	Char	8	32		id
6	INCOME	Num	8	40	4.	income
7	IRA	Char	8	48		ira
8	MARRIED	Char	8	56		married
9	MORTGAGE	Char	8	64		mortgage
10	REGION	Char	8	72		region
11	SAVE_ACT	Char	8	80		save_act
12	SEX	Char	8	88		sex

WebSphere DataStage-inserted partition and sort components

By default, WebSphere DataStage inserts partition and sort components to meet the partitioning and sorting needs of the SAS-interface operators and other operators. For a SAS-interface operator, WebSphere DataStage selects the appropriate component from WebSphere DataStage or from SAS. See the section "WebSphere DataStage-Inserted Partition and Sort Components" of this Reference for information on this facility.

Long name support

For SAS 8.2, WebSphere DataStage handles SAS file and column names up to 32 characters. However, this functionality is dependent on your installing a SAS patch. Obtain this patch from SAS:

SAS Hotfix 82BB40 for the SAS Toolkit

For SAS 6.12, WebSphere DataStage handles SAS file and column names up to 8 characters.

Environment variables

These are the SAS-specific environment variables:

- **APT_SAS_COMMAND *absolute_path***

Overrides the \$PATH directory for SAS and resource sas entry with an absolute path to the US SAS executable. An example path is: /usr/local/sas/sas8.2/sas.

- **APT_SASINT_COMMAND *absolute_path***

Overrides the \$PATH directory for SAS and resource sasint entry with an absolute path to the International SAS executable. An example path is: /usr/local/sas/sas8.2int/dbcs/sas.

- **APT_SAS_CHARSET *icu_character_set***

When the -sas_cs option is not set and a SAS-interface operator encounters a ustring, WebSphere DataStage interrogates this variable to determine what character set to use. If this variable is not set, but APT_SAS_CHARSET_ABORT is set, the operator aborts; otherwise WebSphere DataStage accesses the -impexp_charset option or the APT_IMPEXP_CHARSET environment variable.

- **APT_SAS_CHARSET_ABORT**

Causes a SAS-interface operator to abort if WebSphere DataStage encounters a ustring in the schema and neither the -sas_cs option or the APT_SAS_CHARSET environment variable is set.

- **APT_SAS_TRUNCATION ABORT | NULL | TRUNCATE**

Because a ustring of n characters does not fit into n bytes of a SAS char value, the ustring value might be truncated before the space pad characters and \0. The sasin and sas operators use this variable to determine how to truncate a ustring value to fit into a SAS char field. TRUNCATE, which is the default, causes the ustring to be truncated; ABORT causes the operator to abort; and NULL exports a null field. For NULL and TRUNCATE, the first five occurrences for each column cause an information message to be issued to the log.

- **APT_SAS_ACCEPT_ERROR**

When a SAS procedure causes SAS to exit with an error, this variable prevents the SAS-interface operator from terminating. The default behavior is for WebSphere DataStage to terminate the operator with an error.

- **APT_SAS_DEBUG_LEVEL=[0-2]**

Specifies the level of debugging messages to output from the SAS driver. The values of 0, 1, and 2 duplicate the output for the -debug option of the sas operator: no, yes, and verbose.

- **APT_SAS_DEBUG=1, APT_SAS_DEBUG_IO=1, APT_SAS_DEBUG_VERBOSE=1**

Specifies various debug messages output from the SASToolkit API.

- **APT_HASH_TO_SASHASH**

The WebSphere DataStage hash partitioner contains support for hashing SAS data. In addition, WebSphere DataStage provides the sashash partitioner which uses an alternative non-standard hashing algorithm. Setting the APT_HASH_TO_SASHASH environment variable causes all appropriate instances of hash to be replaced by sashash. If the APT_NO_SAS_TRANSFORMS environment variable is set, APT_HASH_TO_SASHASH has no affect.

- **APT_NO_SAS_TRANSFORMS**

WebSphere DataStage automatically performs certain types of SAS-specific component transformations, such as inserting a sasout operator and substituting sasRoundRobin for eRoundRobin. Setting the APT_NO_SAS_TRANSFORMS variable prevents WebSphere DataStage from making these transformations.

- **APT_NO_SASOUT_INSERT**

This variable selectively disables the sasout operator insertions. It maintains the other SAS-specific transformations.

- **APT_SAS_SHOW_INFO**

Displays the standard SAS output from an import or export transaction. The SAS output is normally deleted since a transaction is usually successful.

- APT_SAS_SCHEMASOURCE_DUMP
Displays the SAS CONTENTS report used by the -schemaFile option. The output also displays the command line given to SAS to produce the report and the pathname of the report. The input file and output file created by SAS is not deleted when this variable is set.
- APT_SAS_S_ARGUMENT *number_of_characters*
Overrides the value of the SAS -s option which specifies how many characters should be skipped from the beginning of the line when reading input SAS source records. The -s option is typically set to 0 indicating that records be read starting with the first character on the line. This environment variable allows you to change that offset.
For example, to skip the line numbers and the following space character in the SAS code below, set the value of the APT_SAS_S_ARGUMENT variable to 6.
`0001 data temp; x=1; run;
0002 proc print; run;`
- APT_SAS_NO_PSDS_USTRING
Outputs a header file that does not list ustring fields.

In addition to the SAS-specific debugging variables, you can set the APT_DEBUG_SUBPROC environment variable to display debug information about each subprocess operator.

Each release of SAS also has an associated environment variable for storing SAS system options specific to the release. The environment variable name includes the version of SAS it applies to; for example, SAS612_OPTIONS, SASV8_OPTIONS, and SASV9_OPTIONS. The usage is the same regardless of release.

Any SAS option that can be specified in the configuration file or on the command line at startup, can also be defined using the version-specific environment variable. SAS looks for the environment variable in the current shell and then applies the defined options to that session.

The environment variables are defined as any other shell environment variable. A ksh example is:

```
export SASV8_OPTIONS='-xwait -news SAS_news_file'
```

A option set using an environment variable overrides the same option set in the configuration file; and an option set in SASx_OPTIONS is overridden by its setting on the SAS startup command line or the OPTIONS statement in the SAS code (as applicable to where options can be defined).

The sasin operator

You can use the sasin operator to convert a standard WebSphere DataStage data set to an SAS data set suitable for input to the sas operator.

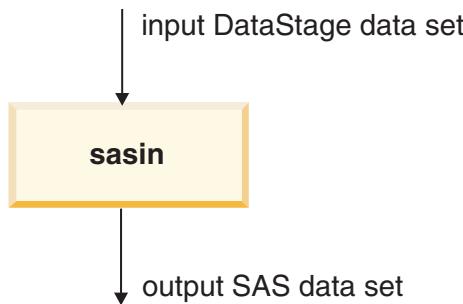
The sasin operator takes a number of optional arguments that control how the WebSphere DataStage data is converted to an SAS data set. For example, you can specify a schema for the input data set using the -schema option.

Because a ustring value of n characters does not fit into n bytes of a SAS char value, the ustring value might be truncated before the space pad characters and \0. You can use the APT_SAS_TRUNCATION environment variable to specify how the truncation is done. It is described in "Environment Variables".

If you do not need to use any of the sasin options, you can bypass the sasin operator and input your WebSphere DataStage data set directly to the sas operator

Note: When the sasin operator executes sequentially and the input WebSphere DataStage data set has multiple partitions, the sasin operator also performs the conversion sequentially.

Data flow diagram



sasin: properties

Table 143. sasin Operator Properties

Property	Value
Number of input data sets	1 standard data set
Number of output data sets	1 SAS data set
Input interface schema	from upstream operator or specified by the sasin operator -schema option
Output interface schema	If no key option is specified: record (sasData:raw;) If a key option is specified: record (sastsort:raw; sasdata:raw;)
Transfer behavior	none
Execution mode	Parallel by default or sequential
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	set

Sasin: syntax and options

The syntax for the sasin operator is shown below. All options are optional.

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
sasin
[-context prefix_string]
[-debug yes | no | verbose]
[-defaultlength length]
[-drop field0 field1 ... fieldN | -keep field0 field1 ... fieldN ]
[-key field_name [-a | -d] [-cs | -ci]]
  [-length integer field0 field1 ... fieldN]
  [-rename schema_field_name SAS_field_name ...]
[-report]
  [-sas_cs icu_character_set | DBCSLANG]
  [-schema schema]
```

Table 144. sasin Operator Options

Option	Use
<code>-context</code>	<p><code>-context <i>prefix_string</i></code></p> <p>Optionally specifies a string that prefixes any informational, warning, or error messages generated by the operator.</p>
<code>-debug</code>	<p><code>-debug yes no verbose</code></p> <p>A setting of <code>-debug yes</code> causes the operator to ignore errors in the SAS program and continue execution of the application. This allows your application to generate output even if a SAS step has an error.</p> <p>By default, the setting is <code>-debug no</code>.</p> <p>Setting <code>-debug verbose</code> is the same as <code>-debug yes</code>, but in addition it causes the operator to echo the SAS source code executed by the operator.</p>
<code>-defaultlength</code>	<p><code>-defaultlength <i>length</i></code></p> <p>Specifies the default length, in bytes, for all SAS numeric fields generated by the operator. The value must be between 3 and 8.</p> <p>This option allows you to control the length of SAS numeric fields when you know the range limits of your data. Using smaller lengths reduces the size of your data.</p> <p>You can override the default length for specific fields using the <code>-length</code> option.</p>
<code>-drop</code>	<p><code>-drop <i>field0 field1 ... fieldN</i></code></p> <p>Optionally specifies the fields of the input data set to be dropped. All fields not specified by <code>-drop</code> are written to the output data set.</p> <p><i>field</i> designates a WebSphere DataStage data set field name. It is the original name of an input field before any renaming, as performed by the <code>-rename</code> option, or name truncation for input field names longer than eight characters in SAS Version 6.</p> <p>You can also specify a range of fields to drop by specifying two field names separated by a hyphen in the form <code><i>fieldN - fieldM</i></code>. In this case, all fields between, and including <code><i>fieldN</i></code> and <code><i>fieldM</i></code> are dropped.</p> <p>This option is mutually exclusive with <code>-keep</code>.</p>

Table 144. sasin Operator Options (continued)

Option	Use
-keep	<p>-keep <i>field0 field1 ... fieldN</i></p> <p>Optionally specifies the fields of the WebSphere DataStage data set to be retained on input. All fields not specified by -keep are dropped from input.</p> <p><i>field</i> designates a WebSphere DataStage data set field name. It is the original name of an input field before any renaming, as performed by the -rename option, or name truncation for input field names longer than eight characters in SAS Version 6.</p> <p>You can also specify a range of fields to drop by specifying two field names separated by a hyphen in the form <i>fieldN - fieldM</i>. In this case, all fields between, and including <i>fieldN</i> and <i>fieldM</i> are dropped.</p> <p>This option is mutually exclusive with -drop.</p>
-key	<p>[-key <i>field_name</i> [-a -d] [-cs -ci]]</p> <p>Specifies a field of the input DataStage data set that you want to use to sort the SAS data set. The key must be the original field name before any renaming, as performed by the -rename option, or name truncation for input field names longer than eight characters for SAS Version 6.</p> <p>This option causes the sasin operator to create a new field in the output data set named <i>sastsort</i>. You specify this field as the sorting key to the tsort operator. Also, you must use a modify operator with the tsort operator to remove this field when it performs the sort.</p> <p>-a and -d specify ascending or descending sort order.</p> <p>The -ci option specifies that the comparison of value keys is case insensitive. The -cs option specifies a case-sensitive comparison, which is the default.</p>

Table 144. sasin Operator Options (continued)

Option	Use
-length	<p>-length integer field0 field1 ... fieldN</p> <p>Specifies the length in bytes, of SAS numeric fields generated by the operator. The value must be between 3 and 8.</p> <p>This option allows you to control the length of SAS numeric fields when you know the range limits of your data. Using smaller lengths reduces the size of your data.</p> <p>By default, all SAS numeric fields are 8 bytes long, or the length specified by the -defaultlength option to the operator.</p> <p>The field name is the original name of an input field before any renaming, as performed by the -rename option, or name truncation for input field names longer than eight characters for SAS 6.</p> <p>You can also specify a range of fields by specifying two field names separated by a hyphen in the form fieldN - fieldM. All fields between, and including fieldN and fieldM use the specified length.</p>
-rename	<p>-rename schema_field_name SAS_field_name</p> <p>Specifies a mapping of an input field name from the WebSphere DataStage data set to a SAS variable name.</p> <p>For multiple mappings, use multiple -rename options.</p> <p>By default, WebSphere DataStage truncates to eight characters the input fields with a name longer than eight characters in SAS Version 6. If truncation causes two fields to have the same name, the operator issues a syntax error.</p> <p>Aliases must be unique with all other alias specifications and with respect to the SAS data set field names.</p>
-report	<p>-report</p> <p>Causes the operator to output a report describing how it converts the input WebSphere DataStage data set to the SAS data set.</p>
-sas_cs	<p>-sas_cs icu_character_set DBCSLANG</p> <p>When your WebSphere DataStage data includes ustring values, you can use the -sas_cs option to specify a character set that maps between WebSphere DataStage ustrings and the char data stored in SAS files. Use the same -sas_cs character setting for all the SAS-interface operators in your data flow. See "Using -sas_cs to Specify a Character Set" for more details.</p> <p>For information on national language support, reference this IBM ICU site:</p> <p>http://oss.software.ibm.com/icu/charset</p>

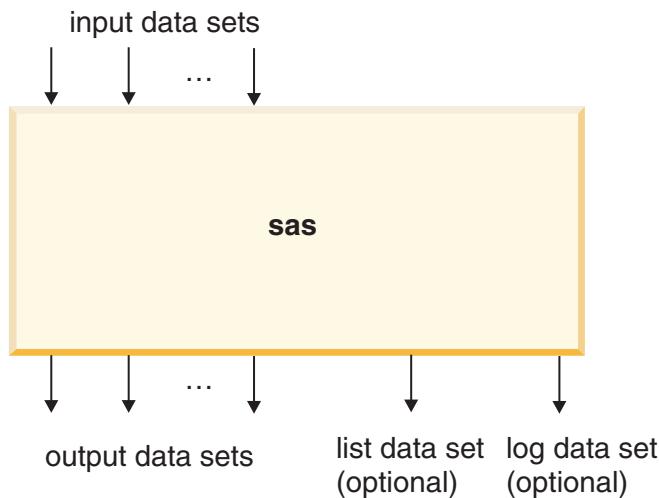
Table 144. *sasin Operator Options (continued)*

Option	Use
-schema	<p>schema <i>schema_definition</i></p> <p>Specifies the record schema of the input WebSphere DataStage data set. Only those fields specified in the record schema are written to the output data set.</p>

The sas operator

You use the sas operator to execute SAS code in parallel or sequentially as part of a WebSphere DataStage application.

Data flow diagram



sas: properties

Table 145. *sas Operator Properties*

Property	Value
Number of input data sets	<p><i>N</i> (set by user)</p> <p>Can be either WebSphere DataStage data sets or SAS data sets.</p>

Table 145. sas Operator Properties (continued)

Property	Value
Number of output data sets	<p>M (set by user)</p> <p>All output data sets can be either SAS data sets or Parallel SAS data sets. If you are passing output to another sas operator, the data should remain in SAS data set format.</p> <p>If requested, the SAS log file is written to the last output data set. For SAS 8.2, the log file contains a header and additional information at the beginning of the file.</p> <p>If requested, the SAS list file is written to the second to last output data set if you also request a log file output, and to the last data set if no log file is requested.</p>
Input interface schema	Derived from the input data set
Output interface schema	<p>For output data sets:</p> <p>As specified by the -schema option or the -schemaFile option when the sasout operator is not used.</p> <p>When the sasout operator is used and the downstream operator expects an SAS data set, the schema is:</p> <pre>record (sasData:raw;)</pre> <p>For list and log data sets:</p> <pre>record (partitionNumber:uint16; recordNumber:uint32; rec:string;)</pre>
Execution mode	Parallel (default) or sequential
Partitioning method	Any parallel mode except modulus
Collection method	Any sequential mode
Preserve-partitioning flag in output data set	Set on all output data sets; clear on log and list data sets

SAS: syntax and options

The syntax for the sas operator is shown below. You must specify either the -source or the -sourcefile option. And, if you are not using the sasout operator, you must also specify the -output option to supply an output schema. All the other options are optional.

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
sas
  -source SAS_code |
  -sourcefile SAScode_filepath
  [-sas_cs icu_character_set | DBCSLANG]
  [-options sas_options]
  [-workingdirectory directory_path | -noworkingdirectory]
  [-listds file | display | dataset | none]
  [-logds file | display | dataset | none]
  [-convertlocal]
  [-input in_port_# sas_dataset_name]
```

```

[-output out_port_# ods_name
[-schemaFile schema_file_name ( alias -schemaSource) |
 -schema schema_definition]
[-debug yes | no | verbose]

```

Table 146. sas Operator Options

Option	Use
-source	<p>-source <i>SAS_code</i></p> <p>Specifies the SAS code to be executed by SAS. The SAS code might contain both PROC steps and DATA steps.</p> <p>You must specify either the -source or -sourcefile option.</p>
-sourcefile	<p>-sourcefile <i>SAS_code_filepath</i></p> <p>Specifies the path to a file containing the SAS source code to be executed. The file path and file contents should be in the UTF-8 character set.</p> <p>You must specify either the -sourcefile or the -source option.</p>
-debug	<p>-debug yes no verbose</p> <p>A setting of -debug yes causes the sas operator to ignore errors in the SAS program and continue execution of the application. This allows your application to generate output even if a SAS step has an error.</p> <p>By default, the setting is -debug no, which causes the operator to abort when it detects an error in the SAS program.</p> <p>Setting -debug verbose is the same as -debug yes, but in addition it causes the operator to echo the SAS source code executed by the operator.</p>
-input	<p>[-input <i>in_port_# sas_ds_name</i>]</p> <p>Specifies the name of the SAS data set, <i>sas_ds_name</i>, receiving its input from the data set connected to <i>in_port_#</i>.</p> <p>The operator uses -input to connect each input data set of the operator to an input of the SAS code executed by the operator. For example, your SAS code contains a DATA step whose input you want to read from input data set 0 of the operator. The following SAS statements might be contained within your code:</p> <pre>libname temp `/tmp';data liborch.parallel_out;set temp.parallel_in;</pre> <p>In this case, you would use -input and set the <i>in_port_#</i> to 0, and the <i>sas_ds_name</i> to the member name <i>parallel_in</i>.</p> <p><i>sas_ds_name</i> is the member name of a SAS data set used as an input to the SAS code executed by the operator. You only need to specify the member name here; do not include any SAS library name prefix.</p> <p>When referencing <i>sas_ds_name</i> as part of the SAS code executed by the operator, always prefix it with liborch, the name of the WebSphere DataStage SAS engine.</p> <p><i>in_port_#</i> is the number of an input data set of the operator. Input data sets are numbered from 0, thus the first input data set is data set 0, the next is data set 1, and so on</p>

Table 146. sas Operator Options (continued)

Option	Use
-listds	<p>-listds <i>file</i> <i>dataset</i> none display</p> <p>Optionally specify that sas should generate a SAS list file.</p> <p>Specifying -listds <i>file</i> causes the sas operator to write the SAS list file generated by the executed SAS code to a plain text file in the working directory. The list is sorted before being written out. The name of the list file, which cannot be modified, is orchident.lst, where ident is the name of the operator, including an index in parentheses if there are more than one with the same name. For example, orchsas(1).lst is the list file from the second sas operator in a step.</p> <p>-listds <i>dataset</i> causes the list file to be written to the last output data set. If you also request that the SAS log file be written to a data set using -logds, the list file is written to the second to last output data set. The data set from a parallel sas operator containing the list information is not sorted.</p> <p>If you specify -listds none, the list is not generated.</p> <p>-listds display is the default. It causes the list to be written to standard error.</p>
-logds	<p>-logds <i>file</i> <i>dataset</i> none display</p> <p>Optionally specify that sas write a SAS log file.</p> <p>-logds <i>file</i> causes the operator to write the SAS log file generated by the executed SAS code to a plain text file in the working directory. The name of the log file, which cannot be modified, is orchident.log, where ident is the name of the operator, including its index in parentheses if there are more than one with the same name. For example, orchsas(1).log is the log file from the second sas operator in a step.</p> <p>For SAS 8.2, the log file contains a header and additional information at the beginning of the file.</p> <p>-logds <i>dataset</i> causes the log file to be written to the last output data set of the operator. The data set from a parallel sas operator containing the SAS log information is not sorted.</p> <p>If you specify -logds none, the log is not generated.</p> <p>-logds display is the default. It causes the log to be written to standard error.</p>
-convertlocal	<p>-convertlocal</p> <p>Optionally specify that the conversion phase of the sas operator (from the parallel data set format to Transitional SAS data set format) should run on the same nodes as the sas operator. If this option is not set, the conversion runs by default with the previous operator's degree of parallelism and, if possible, on the same nodes as the previous operator.</p>
-noworkingdirectory or -nowd	<p>-noworkingdirectory</p> <p>Disables the warning message generated by WebSphere DataStage when you omit the -workingdirectory option.</p> <p>If you omit the -workingdirectory argument, the SAS working directory is indeterminate and WebSphere DataStage automatically generates a warning message. See the -workingdirectory option below. The two options are mutually exclusive.</p>

Table 146. sas Operator Options (continued)

Option	Use
-options	<p>-options <i>sas_options</i></p> <p>Optionally specify a quoted string containing any options that can be specified to a SAS OPTIONS directive. These options are executed before the operator executes your SAS code. For example, you can use this argument to enable the SAS fullstimer.</p> <p>You can specify multiple options, separated by spaces.</p> <p>By default, the operator executes your SAS code with the SAS options notes and source. Specifying any string for <i>sas_options</i> configures the operator to execute your code using only the specified options. Therefore you must include notes and source in <i>sas_options</i> if you still want to use them.</p>
-output	<p>-output <i>out_port_# ods_name [-schemaFile schema_file_name -schema schema_definition]</i></p> <p>Optionally specify the name of the SAS data set, <i>ods_name</i>, writing its output to the data set connected to <i>out_port_#</i> of the operator.</p> <p>The operator uses -output to connect each output data set of the operator to an output of the SAS code executed by the operator.</p> <p>For example, your SAS code contains a DATA step whose output you want to write to output data set 0 of the operator. Here is the SAS expression contained within your code:</p> <pre>data liborch.parallel_out;</pre> <p>In this case, you would use -output and set the <i>out_port_#</i> to 0, and the <i>ods_name</i> to the member name <i>parallel_out</i>.</p> <p><i>ods_name</i> corresponds to the name of an SAS data set used as an output by the SAS code executed by the operator. You only need to specify the member name here; do not include any SAS library name prefix.</p> <p>When referencing <i>ods_name</i> as part of the SAS code executed by the operator, always prefix it with <i>liborch</i>, the name of the WebSphere DataStage SAS engine.</p> <p><i>out_port_#</i> is the number of an output data set of the operator. Output data sets are numbered starting from 0.</p> <p>You use the -schemaFile suboption to specify the name of a SAS file containing the metadata column information which WebSphere DataStage uses to generate a WebSphere DataStage schema; or you use the -schema suboption followed by the schema definition. See "Specifying an Output Schema" for more details.</p> <p>If both the -schemaFile option and the -sas_cs option are set, all of your SAS char fields are converted to WebSphere DataStage ustring values. If the -sas_cs option is not set, all of your SAS char values are converted to WebSphere DataStage string values. To obtain a mixture of string and ustring values use the -schema option. See "Specifying an Output Schema" for more details.</p> <p>Note: The -schemaFile and -schema suboptions are designated as optional because you should not specify them for the sas operator if you specify them for the sasout operator. It is an error to specify them for both the sas and sasout operators.</p>

Table 146. sas Operator Options (continued)

Option	Use
-sas_cs	<p>-sas_cs icu_character_set DBCSLANG</p> <p>When your WebSphere DataStage data includes ustring values, you can use the -sas_cs option to specify a character set that maps between WebSphere DataStage ustrings and the char data stored in SAS files. Use the same -sas_cs character setting for all the SAS-interface operators in your data flow. See "Using -sas_cs to Specify a Character Set" for more details.</p> <p>For information on national language support, reference this IBM ICU site: http://oss.software.ibm.com/icu/charset</p>
-workingdirectory or -wd	<p>-workingdirectory <i>dir_name</i></p> <p>Specifies the name of the working directory on all processing nodes executing the SAS application. All relative pathnames in your application are relative to the specified directory.</p> <p>If you omit this argument, the directory is indeterminate and WebSphere DataStage generates a warning message. You can use -noworkingdirectory to disable the warning.</p> <p>This option also determines the location of the fileconfig.sasversion. By default, the operator searches the directory specified by -workingdirectory, then your home directory, then the SAS install directory for config.sasversion.</p> <p>Legal values for <i>dir_name</i> are fully qualified pathnames (which must be valid on all processing nodes) or "." (period), corresponding to the name of the current working directory on the workstation invoking the application. Relative pathnames for <i>dir_name</i> are illegal.</p>

The sasout operator

You use the sasout operator to convert an SAS data set to the standard WebSphere DataStage data set format suitable for input to standard WebSphere DataStage operators. When your data includes ustring values, you must use the -sas_cs option of the SAS interface operators to specify the character set used by WebSphere DataStage to convert between the char data stored in SAS files and WebSphere DataStage ustring values. Use the same value for this option for the sasin, sas, and sasout operators.

This operator is only required when a standard WebSphere DataStage operator follows a SAS interface operator.

The sasout operator requires that you specify either the -schema option or the -schemaFile option to pass a record schema that defines the layout of its output data set. You supply a WebSphere DataStage schema definition to the -schema option. For the -schemaFile option, you specify the file path of a SAS file that has the same meta data description as the SAS output stream, and WebSphere DataStage derives the schema definition from that file. See "Specifying an Output Schema" for more details.

See the section "Data Flow Diagram" for information on the schema that is derived from a SAS file when you use the -schemaFile option.

As part of converting an SAS data set to a standard WebSphere DataStage data set, the sasout operator converts input SAS data types to the corresponding WebSphere DataStage data types using that record schema.

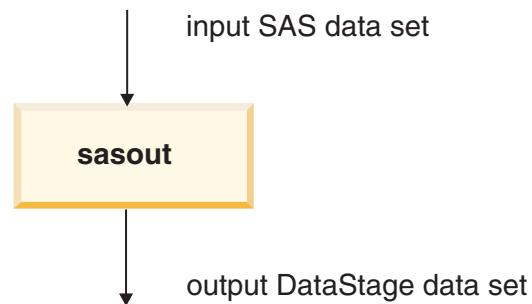
For example, if the WebSphere DataStage SAS data set contains a SAS numeric field named a_field that you want to convert to an int16, you include the following line as part of the sasout record schema:

```
record (
    ...
    a_field:int16;
    ...
)
```

If you want to convert the field to a decimal, you would use the appropriate decimal definition, including precision and scale.

When converting a SAS numeric field to a WebSphere DataStage numeric, you can get a numeric overflow or underflow if the destination data type is too small to hold the value of the SAS field. By default, WebSphere DataStage issues an error message and aborts the program if this occurs. However, if the record schema passed to sasout defines a field as nullable, the numeric overflow or underflow does not cause an error. Instead, the destination field is set to null and processing continues.

Data flow diagram



sasout: properties

Table 147. sasout Operator Properties

Property	Value
Number of input data sets	1 SAS data set
Number of output data sets	1 WebSphere DataStage data set
Input interface schema	none
Output interface schema	as specified by the -schema option or the -schemaFile option
Transfer behavior	none
Execution mode	parallel (default) or sequential
Partitioning method	any (parallel mode)
Collection method	any (sequential mode)
Preserve-partitioning flag in output data set	clear

Sasout: syntax and options

The syntax for the sasout operator is shown below. Either the -schema option or the -schemaFile option is required to supply an output schema. When using sasout, do not specify an output schema to the sas operator.

Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
sasout
-schema schema_definition | -schemaFile filepath
[-sas_cs icu_character_set | SAS_DBCLSLANG]
[-context prefix_string]
[-debug no | yes | verbose]
[-drop | -nodrop]
[-rename schema_field_name SAS_name ...]
[-report]
```

Table 148. sasout Operator Options

Option	Use
-schema	<p><i>schema schema_definition</i></p> <p>Specifies the record schema of the output WebSphere DataStage data set. Only those fields specified in the record schema are written to the output data set. You must specify either the -schema option or the -schemaFile option. See "Specifying an Output Schema" for more details.</p>
-schemaFile	<p><i>schemaFile filepath</i></p> <p>You use the -schemaFile option to specify the name of a SAS file containing the metadata column information which WebSphere DataStage uses to generate a WebSphere DataStage schema; or you use the -schema suboption followed by the schema definition. See "Specifying an Output Schema" for more details.</p> <p>If both the -schemaFile option and the -sas_cs option are set, all of your SAS char fields are converted to WebSphere DataStage ustring values. If the -sas_cs option is not set, all of your SAS char values are converted to WebSphere DataStage string values. To obtain a mixture of string and ustring values use the -schema option.</p>
-debug	<p><i>-debug -yes -no -verbose</i></p> <p>A setting of -debug -yes causes the operator to ignore errors in the SAS program and continue execution of the application. This allows your application to generate output even if a SAS step has an error.</p> <p>By default, the setting is -debug -no, which causes the operator to abort when it detects an error in the SAS program.</p> <p>Setting -debug -verbose is the same as -debug -yes, but in addition it causes the operator to echo the SAS source code executed by the operator.</p>
-drop	<p><i>-drop</i></p> <p>Specifies that sasout drop any input fields not included in the record schema. This is the default action of sasout. You can use the -nodrop option to cause sasout to pass all input fields to the output data set.</p> <p>The -drop and -nodrop options are mutually exclusive.</p>

Table 148. *sasout Operator Options (continued)*

Option	Use
-nodrop	-nodrop Specifies the failure of the step if there are fields in the input data set that are not included in the record schema passed to sasout. You can use the -drop option to cause sasout to drop all input fields not included in the record schema.
-rename	-rename <i>in_field_name WebSphere DataStage_field_name ...</i> Specifies a mapping of an input field name from the SAS data set to a WebSphere DataStage data set field name. For multiple mappings, use multiple -rename options. Aliases must be unique with all other alias specifications and with respect to the SAS data set field names.
-report	-report Causes the operator to output a report describing how it converts the input SAS data set to the WebSphere DataStage data set.
-sas_cs	-sas_cs <i>icu_character_set DBCSLANG</i> When your WebSphere DataStage data includes ustring values, you can use the -sas_cs option to specify a character set that maps between WebSphere DataStage ustrings and the char data stored in SAS files. Use the same -sas_cs character setting for all the SAS-interface operators in your data flow. See "Using -sas_cs to Specify a Character Set" for more details. For information on national language support, reference this IBM ICU site: http://oss.software.ibm.com/icu/charset

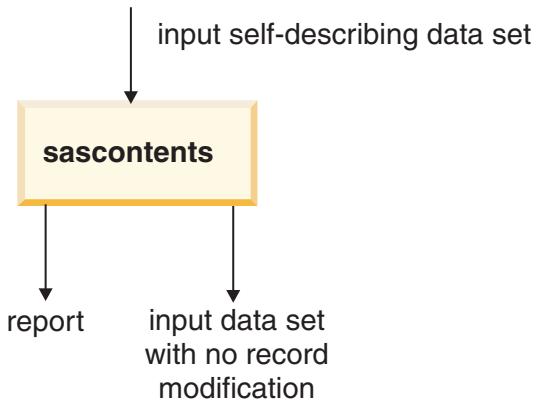
The **sascontents** operator

You use the sascontents operator to generate a report about an input self-describing data set.

The report is similar to the report generated by the SAS procedure PROC CONTENTS. It is written to the data stream. Use the peek operator in sequential mode to print the report to the screen.

The operator takes a single input data set and writes the generated report to output 0. Optionally, the input data set can be written to output 1 with no record modification.

Data flow diagram



sascontents: properties

Table 149. sascontents Operator Properties

Property	Value
Number of input data sets	1 self-describing data set
Number of output data sets	1 or 2 output 0: generated report output 1: an optional copy of the input data set with no record modification
Input interface schema	none
Output interface schema	none
Transfer behavior	input to output with no record modification
Execution mode	parallel (default) or sequential
Partitioning method	same (cannot be overridden)
Preserve-partitioning flag in output data set	output 0: not applicable -- there is a single partition output 1: set

sascontents: syntax and options

The syntax for the sascontents operator is shown below. No options are required. Terms in italic typeface are option strings you supply. When your option string contains a space or a tab character, you must enclose it in single quotes.

```
sascontents
  [-name name_string_for_report]  [-recordcount]
  [-schema]
```

Table 150. sascontents Operator Options

Option	Use
-name	<p>-name <i>name_string_for_report</i></p> <p>Optionally specifies a name string, typically the name of the data set. The name is used to make the generated report identifiable.</p>

Table 150. *sascontents Operator Options (continued)*

Option	Use
-recordcount	-recordcount Optionally specifies the inclusion of record and partition counts in the report.
-schema	-schema Optionally configures the operator to generate a WebSphere DataStage record schema for the self-describing input data set instead of a report.

Example reports

The names in these examples can contain a maximum of 8 characters for SAS 6.12 and a maximum of 32 characters for SAS 8.2.

- Report with no options specified:

```
Field Count: 2    Record Length: 13    Created 2003-05-17 15:22:34
Version: 4
```

Number	Name	Type	Len	Offset	Format	Label
1	A	Num	8	0	6.	a
2	B	Char	5	8		b

- Report with the -schema option specified:

```
Suggested schema for SAS dataset sasin[00].v:
record (A:int32; B:string[5])
```

- Report with the -schema and -recordcount options specified:

```
Suggested schema for SAS data set sasin[00].v
Number of partitions = 4
Number of data records = 10
Suggested schema for parallel data set sasin[00].v
record (A:int32; B:string[5])
Field descriptor records = 8
Total records in dataset = 22
```

- Report with the -schema, -recordcount and -name options specified. The name value is 'my data set':

```
Suggested schema for SAS dataset my data set
Number of partitions = 4
Number of data records = 10
Suggested schema for parallel dataset sasin[00].v
record (A:int32; B:string[5])
Field descriptor records = 8
Total records in dataset = 22
```

Chapter 16. The Oracle interface library

Four operators provide access to Oracle databases.

- The oraread operator reads records from an Oracle table and places them in a WebSphere DataStage data set.
- The orawrite operator sets up a connection to Oracle and inserts records into a table. The operator takes a single input data set. The write mode of the operator determines how the records of a data set are inserted into the table.

Accessing Oracle from WebSphere DataStage

This section assumes that WebSphere DataStage users have been configured to access Oracle using the Oracle configuration process outlined in *IBM Information Server Planning, Installation, and Configuration Guide*.

To access Oracle from WebSphere DataStage:

1. Set your ORACLE_HOME environment variable to your Oracle client installation.
2. Start Oracle.
3. Add ORACLE_HOME/bin to your PATH and ORACLE_HOME/lib to your LIBPATH, LD_LIBRARY_PATH, or SHLIB_PATH.

Note: APT_ORCHHOME/bin must appear before ORACLE_HOME/bin in your PATH.

4. Have login privileges to Oracle using a valid Oracle user name and corresponding password. These must be recognized by Oracle before you attempt to access it.

Changing library paths

Because an Oracle client often cannot connect to a different version of the Oracle database, you might find it necessary to change your Oracle library path.

To change your Oracle library path:

1. Set your ORACLE_HOME environment variable to your Oracle client installation which must include the Oracle Database Utilities and the Oracle network software.
2. From your \$APT_ORCHHOME/install directory, execute this command:
./install.liborchoracle
3. Verify that the Oracle library links in your \$APT_ORCHHOME/lib directory have the correct version.

Preserving blanks in fields

Setting the APT_ORACLE_PRESERVE_BLANKS environment variable causes the PRESERVE BLANKS option to be set in the control file. This option preserves leading and trailing spaces, and retains fields which contain only spaces. When PRESERVE BLANKS is not set, Oracle removes the spaces and considers fields with only spaces to be NULL values.

Handling # and \$ characters in Oracle column names

The Oracle operators accept the # and \$ characters for Oracle table column names. WebSphere DataStage converts these two reserved characters into an internal format when they are written to the Oracle database and reconverts them when they are read from Oracle.

The internal representation for the # character is '_035_' and the internal representation for the \$ character is '_036_'. You should avoid using these internal strings in your Oracle column names.

Using the external representation

Use the external representation (with the # and \$ characters) when referring to table column names for these options:

- The -selectlist options of oraread
- The -query option of oraread and oralookup
- The -insert and and -update options

Using the internal representation

Use the internal representation (with the _035_ and _036_ strings) when referring to table column names for these options:

- The -query option of db2lookup when referring to an ORCHESTRATE name. For example:
`-query 'MS##,D#$ FROM tablename WHERE (BS# + ORCHESTRATE.B_036_035)'`
- The -insert and -update options when referring to an ORCHESTRATE name. For example:
`-insert 'INSERT INTO tablename (A#,B$#)
VALUES (ORCHESTRATE.A_035_, ORCHESTRATE.B_036_035_)'
-update 'UPDATE tablename set B$# = ORCHESTRATE.B_036_035_ WHERE (A# =
ORCHESTRATE.A_035_)'`

National Language Support

WebSphere DataStage's National Language Support (NLS) makes it possible for you to process data in international languages using Unicode character sets. WebSphere DataStage uses International Components for Unicode (ICU) libraries to support NLS functionality. For information on National Language Support, see *WebSphere DataStage National Language Support Guide* and access the ICU home page:

<http://oss.software.ibm.com/developerworks/opensource/icu/project>

The WebSphere DataStage Oracle operators support Unicode character data in schema, table, and index names; in user names and passwords; column names; table and column aliases; SQL*Net service names; SQL statements; and file-name and directory paths.

ICU character set options

The operators have two options which optionally control character mapping:

- `-db_cs icu_character_set`
Specifies an ICU character set to map between Oracle char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to Oracle.
- `-nchar_cs icu_character_set`
Specifies an ICU character set to map between Oracle nchar and nvarchar2 values and WebSphere DataStage ustring data.

Mapping between ICU and Oracle character sets

To determine what Oracle character set corresponds to your ICU character-set specifications, WebSphere DataStage uses the table in \$APT_ORCHHOME/etc/oracle_cs.txt. The table is populated with some default values. Update this file if you are using an ICU character set that has no entry in the file; otherwise UTF8 is used. The Oracle character set is specified in the sqldr control file as the CHARACTERSET option for loading your data.

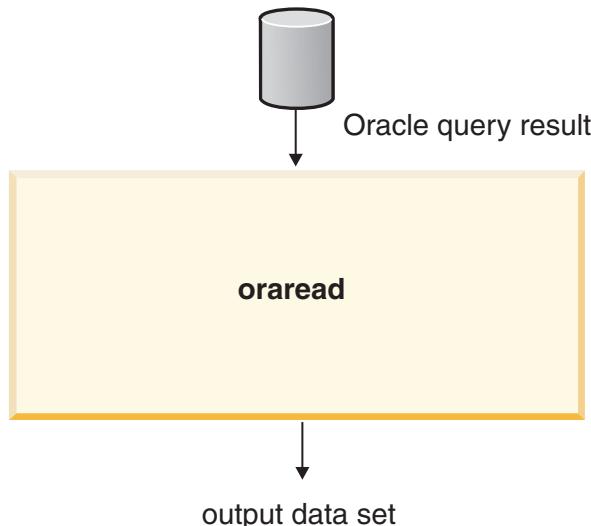
Here is a sample oracle_cs.txt table

ICU Character Set	Oracle Character Set
UTF-8	UTF8
UTF-16	AL16UTF16
ASCL_ISO8859-1	WE8ISO8859P1
ISO-8859-1	WE8ISO8859P1
ASCL_MS1252	WE8ISO8859P1
EUC-JP	JA16EUC
ASCL-JPN-EUC	JA16EUC
ASCL_JPN-SJIS	JA16SJIS
Shift_JIS	JA16SJIS
US-ASCII	US7ASCII
ASCL-ASCII	US7ASCII

The oraread operator

The oraread operator sets up a connection to an Oracle table, uses an SQL query to request rows (records) from the table, and outputs the result as a WebSphere DataStage data set.

Data flow diagram



oraread: properties

Table 151. oraread Operator Properties

Property	Value
Number of input data sets	0
Number of output data sets	1
Input interface schema	none

Table 151. oraread Operator Properties (continued)

Property	Value
Output interface schema	determined by the SQL query
Transfer behavior	none
Execution mode	sequential (default) or parallel
Partitioning method	not applicable
Collection method	not applicable
Preserve-partitioning flag in output data set	clear
Composite operator	no

Operator action

Here are the chief characteristics of the oraread operator:

- It reads only from non-partitioned and range-partitioned tables when in parallel mode.
- The operator functions sequentially, unless you specify the -part suboption of the -query or -table option.
- You can direct it to run in specific node pools. See "Where the oraread Operator Runs".
- It translates the query's result set (a two-dimensional array) row by row to a WebSphere DataStage data set, as discussed in "Column Name Conversion".
- Its output is an WebSphere DataStage data set that you can use as input to a subsequent WebSphere DataStage operator.
- Its translation includes the conversion of Oracle data types to WebSphere DataStage data types, as listed in "Data Type Conversion".
- The size of Oracle rows can be greater than that of WebSphere DataStage records. See "Oracle Record Size".
- The operator specifies either an Oracle table to read or an SQL query to carry out. See "Specifying the Oracle Table Name" and "Specifying an SQL SELECT Statement".
- It optionally specifies commands to be run on all processing nodes before the read operation is performed and after it has completed.
- You can perform a join operation between WebSphere DataStage data sets and Oracle data. See "Join Operations".

Note: An RDBMS such as Oracle does not guarantee deterministic ordering behavior unless an SQL operation constrains it to do so. Thus, if you read the same Oracle table multiple times, Oracle does not guarantee delivery of the records in the same order every time. While Oracle allows you to run queries against tables in parallel, not all queries should be run in parallel. For example, some queries perform an operation that must be carried out sequentially, such as a grouping operation or a non-collocated join operation. These types of queries should be executed sequentially in order to guarantee correct results.

Line 1. the operator.

Where the oraread operator runs

By default the oraread operator runs sequentially. However, you can direct it to read from multiple partitions and run on all processing nodes in the default node pool defined by your WebSphere DataStage configuration file. You do so by specifying the operator's -server option. The option takes a single argument defining a remote resource pool name. WebSphere DataStage runs the operator on all processing nodes with a resource of type Oracle in the pool specified by -server.

For example, the following WebSphere DataStage configuration file defines three processing nodes; two of them define resources of type Oracle:

```
{
    node "node0"
    {
        fastname "node0_css"
        pool "" "node0" "node0_css"                                "syncsort"
        resource disk "/orch/s0" {pool "" "export"}
        resource scratchdisk "/scratch" {}
        resource ORACLE node0 {pool "oracle_pool"}   }
    }

    node "node1"
    {
        fastname "node1_css"
        pool "" "node1" "node1_css"                                "syncsort"
        resource disk "/orch/s0" {pool "" "export"}
        resource scratchdisk "/scratch" {}
        resource ORACLE node0 {pool "oracle_pool"}   }
    }

    node "node2"
    {
        fastname "node2_css"
        pool "" "node2" "node2_css"
        resource disk "/orch/s0" {}
        resource scratchdisk "/scratch" {}
    }
}
```

In this case, the option -server oracle_pool configures the operator to execute only on node0 and node1.

Column name conversion

An Oracle result set is defined by a collection of rows and columns. The oraread operator translates the query's result set (a two-dimensional array) to a WebSphere DataStage data set. Here is how an Oracle query result set is converted to a WebSphere DataStage data set:

- The rows of an Oracle result set correspond to the records of a WebSphere DataStage data set.
- The columns of an Oracle row correspond to the fields of a WebSphere DataStage record; the name and data type of an Oracle column correspond to the name and data type of a WebSphere DataStage field.
- Names are translated exactly except when the Oracle column name contains a character that WebSphere DataStage does not support. In that case, the unsupported character is replaced by two underscore characters.
- Both Oracle columns and WebSphere DataStage fields support nulls, and a null contained in an Oracle column is stored as a null in the corresponding WebSphere DataStage field.

Data type conversion

The oraread operator converts Oracle data types to WebSphere DataStage data types, as in this table:

Table 152. oraread Operator Data Type Conversion

Oracle Data Type	Corresponding WebSphere DataStage Data Type
CHAR(<i>n</i>)	string[<i>n</i>] or ustring[n], a fixed-length string with length = <i>n</i>
DATE	timestamp
NUMBER	decimal[38,10]
NUMBER[<i>p,s</i>]	int32 if precision (<i>p</i>) < 11 and scale (<i>s</i>) = 0 decimal[<i>p,s</i>] if precision (<i>p</i>) >= 11 or scale > 0

Table 152. oraread Operator Data Type Conversion (continued)

Oracle Data Type	Corresponding WebSphere DataStage Data Type
RAW(<i>n</i>)	not supported
VARCHAR(<i>n</i>)	string[max= <i>n</i>] or ustring[max= <i>n</i>], a variable-length string with maximum length = <i>n</i>

Note: Data types that are not listed in the table above generate an error.

Oracle record size

The size of a WebSphere DataStage record is limited to 32 KB. However, Oracle records can be longer than 32 KB. If you attempt to read a record longer than 32 KB, WebSphere DataStage returns an error and terminates your application.

Targeting the read operation

When reading an Oracle table, you can either specify the table name and allow WebSphere DataStage to generate a default query that reads the table or you can explicitly specify the query.

Specifying the Oracle table name

If you choose the -table option, WebSphere DataStage issues the following SQL SELECT statement to read the table:

```
select [selectlist]
      from table_name
        and (filter)];
```

You can specify optional parameters to narrow the read operation. They are as follows:

- The selectlist specifies the columns of the table to be read; by default, WebSphere DataStage reads all columns.
- The filter specifies the rows of the table to exclude from the read operation; by default, WebSphere DataStage reads all rows.

You can optionally specify -open and -close option commands. These commands are executed by Oracle on every processing node containing a partition of the table before the table is opened and after it is closed.

See "Example 1: Reading an Oracle Table and Modifying a Field Name" for an example of using the table option.

Specifying an SQL SELECT statement

If you choose the -query option, you pass an SQL query to the operator. The query specifies the table and the processing that you want to perform on the table as it is read into WebSphere DataStage. The SQL statement can contain joins, views, database links, synonyms, and so on. However, the following restrictions apply to -query:

- The -query might not contain bind variables.
- If you want to include a filter or select list, you must specify them as part of the query.
- The query runs sequentially by default, but can be run in parallel if you specify the -part option.
- You can specify an optional open and close command. Oracle runs these commands immediately before the database connection is opened and after it is closed.

Note: Complex queries executed in parallel might cause the database to consume large amounts of system resources.

See "Example 2: Reading from an Oracle Table in Parallel with the query Option" for an example of using the -query option.

Join operations

You can perform a join operation between WebSphere DataStage data sets and Oracle data. First invoke the oraread operator and then invoke either the lookup operator or a join operator. See "Lookup Operator" and c_deeadvrf_The_Oracle_Interface_Library.dita, "The Join Library." Alternatively, you can use the oralookup operator described in "The oralookup Operator".

Oraread: syntax and options

The syntax for oraread follows. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
oraread
  -query sql_query |
  -table table_name [-filter filter]
           [-selectlist list]

  -dboptions '{user = username, password = password}'
             |
             '{user = \@file_name\}'

  [-close close_command]
  [-db_cs icu_character_set] [-nchar_cs icu_character_set]
  [-open open_command]
  [-ora8partition partition_name] [-part table_name]
  [-server remote_server_name]
```

You must specify either the -query or -table option. You must also specify -dboptions and supply the necessary information. The table on the next page lists the options.

Table 153. oraread Operator Options

Option	Use
-close	<p>-close <i>close_command</i></p> <p>Specify a command, enclosed in single quotes, to be parsed and executed by Oracle on all processing nodes after WebSphere DataStage completes processing the Oracle table and before it disconnects from Oracle.</p> <p>If you do not specify a <i>close_command</i>, WebSphere DataStage terminates its connection to Oracle. There is no default <i>close_command</i>.</p> <p>You can include an Oracle stored procedure as part of <i>close_command</i>, in the form:</p> <p>"execute <i>procedureName</i>;"</p>

Table 153. oraread Operator Options (continued)

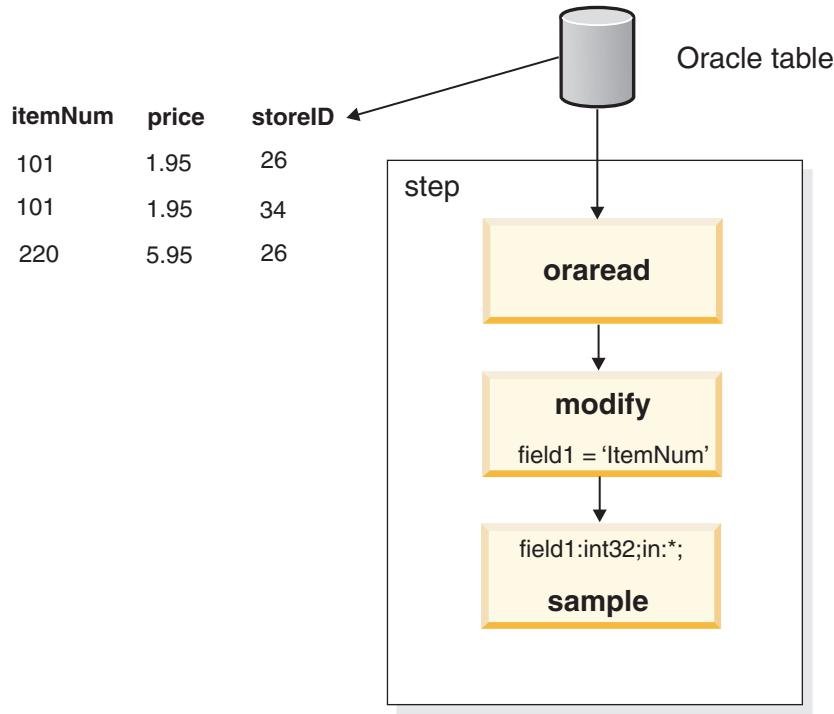
Option	Use
-db_cs	<p>-db_cs <i>icu_character_set</i> Specify an ICU character set to map between Oracle char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to Oracle. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data. If this option is specified, the -nchar option must also be specified.</p> <p>WebSphere DataStage maps your ICU character setting to its Oracle character-set equivalent. See "Mapping Between ICU and Oracle Character Sets" for the details.</p> <p>For information on national language support, reference this IBM ICU site:</p> <p>http://www.oss.software.ibm.com/icu/charset</p>
-nchar_cs	<p>-nchar_cs <i>icu_character_set</i></p> <p>Specify an ICU character set to map between Oracle nchar and nvarchar2 values and WebSphere DataStage ustring data. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data. If this option is specified, the -db_cs option must also be specified.</p> <p>WebSphere DataStage maps your ICU character setting to its Oracle character-set equivalent. See "Mapping Between ICU and Oracle Character Sets" for the details.</p> <p>For information on national language support, reference this IBM ICU site:</p> <p>http://www.oss.software.ibm.com/icu/charset</p>
-dboptions	<p>-dboptions '{user=username, password=password}' '{user='\@filename\'}' [arraysize = <i>num_records</i>]</p> <p>Specify either a user name and password for connecting to Oracle or a file containing the user name and password.</p> <p>You can optionally use -arraysize to specify the number of records in each block of data read from Oracle. WebSphere DataStage reads records in as many blocks as required to read all source data from Oracle.</p> <p>By default, the array size is 1000 records. You can modify this parameter to tune the performance of your application.</p>
-open	<p>-open <i>open_command</i></p> <p>Specify a command, enclosed in single quotes, to be parsed and executed by Oracle on all processing nodes before the table is opened.</p>
-ora8partition	<p>-ora8partition <i>partition_name</i></p> <p>Specify the name of the specific Oracle table partition that contains the rows you want to read.</p>

Table 153. oraread Operator Options (continued)

Option	Use
-part	<p>-part <i>table_name</i></p> <p>Specifies running the read operator in parallel on the processing nodes in the default node pool. The default execution mode is sequential.</p> <p>The <i>table_name</i> must be the name of the table specified by -table.</p>
-query	<p>-query <i>sql_query</i> [-part <i>table_name</i>]</p> <p>Specify an SQL query, enclosed in single quotes, to read a table. The query specifies the table and the processing that you want to perform on the table as it is read into WebSphere DataStage. This statement can contain joins, views, database links, synonyms, and so on.</p>
-server	<p>-server <i>remote_server_name</i></p> <p><i>remote_server_name</i> must specify a remote connection. To specify a local connection, set your ORACLE_SID environment variable to a local server.</p> <p>See "Where the oraread Operator Runs" for more information.</p>
-table	<p>-table <i>table_name</i> [-filter <i>filter</i>] [-selectlist <i>list</i>]</p> <p>Specify the name of the Oracle table. The table must exist and you must have SELECT privileges on the table. The table name might contain a view name only if the operator executes sequentially.</p> <p>If your Oracle user name does not correspond to that of the owner of the table, prefix the table name with that of the table owner in the form:</p> <p><i>tableowner.table_name</i></p> <p>The -filter suboption optionally specifies a conjunction, enclosed in single quotes, to the WHERE clause of the SELECT statement to specify the rows of the table to include or exclude from reading into WebSphere DataStage. See "Targeting the Read Operation" for more information.</p> <p>The suboption -selectlist optionally specifies an SQL select list, enclosed in single quotes, that can be used to determine which fields are read. You must specify the fields in <i>list</i> in the same order as the fields are defined in the record schema of the input table.</p>

Oraread example 1: reading an Oracle table and modifying a field name

The following figure shows an Oracle table used as input to an WebSphere DataStage operator:



The Oracle table contains three columns whose data types the operator converts as follows:

- itemNum of type NUMBER[3,0] is converted to int32
- price of type NUMBER[6,2] is converted to decimal[6,2]
- storeID of type NUMBER[2,0] is converted to int32

The schema of the WebSphere DataStage data set created from the table is also shown in this figure. Note that the WebSphere DataStage field names are the same as the column names of the Oracle table.

However, the operator to which the data set is passed has an input interface schema containing the 32-bit integer field field1, while the data set created from the Oracle table does not contain a field of the same name. For this reason, the modify operator must be placed between oraread and sampleOperator to translate the name of the field, itemNum, to the name field1. See "Modify Operator" for more information.

Here is the osh syntax for this example:

```
$ modifySpec="field1 = itemNum;"  
$ osh "oraread -table 'table_1'  
      -dboptions {'user = user101, password = userPword'}  
      | modify '$modifySpec' | ... "
```

Example 2: reading from an Oracle table in parallel with the query option

The following query configures oraread to read the columns itemNum, price, and storeID from table_1 in parallel. One instance of the operator runs for each partition of the table.

Here is the osh syntax for this example:

```
$ osh "oraread -query 'select itemNum, price, storeID from table_1'  
      -part 'table_1'  
      -dboptions {'user = user101, password = userPword'}  
      ..."
```

The orawrite operator

The orawrite operator sets up a connection to Oracle and inserts records into a table. The operator takes a single input data set. The write mode of the operator determines how the records of a data set are inserted into the table.

Writing to a multibyte database

Specifying chars and varchars

Specify chars and varchars in bytes, with two bytes for each character. This example specifies 10 characters:

```
create table orch_data(col_a varchar(20));
```

Specifying nchar and nvarchar2 column size

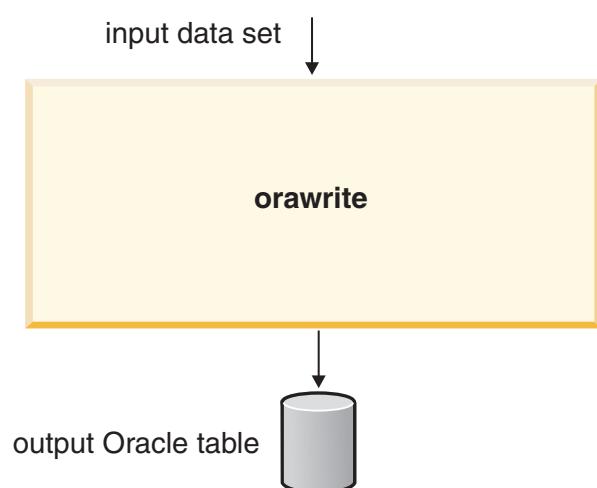
Specify nchar and nvarchar2 columns in characters. For example, this example specifies 10 characters:

```
create table orch_data(col_a nvarchar2(10));
```

Loading delimited files with the orawrite operator

You can specify a field delimiter character for the orawrite operator using the APT_ORACLE_LOAD_DELIMITED environment variable. Setting this variable makes it possible for orawrite to load fields with trailing or leading blank characters. When this variable is set, but does not have a value, the comma (,) character is used as the default delimiter.

Data flow diagram



orawrite: properties

Table 154. orawrite Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	0
Input interface schema	none
Output interface schema	derived from the input data set.
Transfer behavior	none
Execution mode	parallel (default) or sequential
Partitioning method	same (Note that you can override this partitioning method. However, a partitioning method of <i>entire</i> is not allowed.)
Collection method	any
Preserve-partitioning flag in output data set	not applicable
Composite operator	yes

Operator action

Here are the main characteristics of the orawrite operator:

- It translates a WebSphere DataStage data set record by record to an Oracle table using Oracle's SQL*Loader Parallel Direct Path Load method.
- Translation includes the conversion of WebSphere DataStage data types to Oracle data types. See "Data Type Conversion".
- The operator appends records to an existing table, unless you set another mode of writing. See "Write Modes".
- When you write to an existing table, the schema of the table determines the operator's input interface, and the input data set schema must be compatible with the table's schema. See "Matched and Unmatched Fields".
- Each instance of a parallel write operator running on a processing node writes its partition of the data set to the Oracle table. If any instance fails, all fail.

You can optionally specify Oracle commands to be parsed and executed on all processing nodes before the write operation runs or after it completes.

Indexed tables

Because the Oracle write operator writes to a table using the Parallel Direct Path Load method, the operator cannot write to a table that has indexes defined on it unless you include either the -index rebuild or the -index maintenance option.

If you want to write to such a table and not use an -index option you must first delete the indexes and recreate them after orawrite has finished processing. The operator checks for indexes and returns an error if any are found. You can also write to an indexed table if the operator is run in sequential mode and the environment variable APT_ORACLE_LOAD_OPTIONS is set to 'OPTIONS (DIRECT=TRUE, PARALLEL=FALSE)'.

Note that if you define the environment variable APT_ORACLE_LOAD_OPTIONS, WebSphere DataStage allows you to attempt to write to an indexed table, regardless of how the variable is defined.

Where the write operator runs

The default execution mode of the orawrite operator is parallel. The default is that the number of processing nodes is based on the configuration file. However, if the environment variable APT_ORA_IGNORE_CONFIG_FILE_PARALLELISM is set, the number of players is set to the number of data files in the table's tablespace.

To direct the operator to run sequentially:

Specify the [seq] framework argument.

You can optionally specify the resource pool in which the operator runs by choosing the operator's -server option, which takes a single argument defining a resource pool name. WebSphere DataStage then runs the operator on all processing nodes with a resource of type Oracle in the pool specified by server.

Data conventions on write operations to Oracle

Oracle columns are named identically as WebSphere DataStage fields, with these restrictions:

- Oracle column names are limited to 30 characters. If an WebSphere DataStage field name is longer, you can do one of the following:
 - Choose the -truncate option to configure the operator to truncate WebSphere DataStage field names to 30 characters.
 - Use the modify operator to modify the WebSphere DataStage field name. See "Modify Operator" for information.
- An WebSphere DataStage data set written to Oracle might not contain fields of certain types. If it does, an error occurs and the corresponding step terminates. However WebSphere DataStage offers operators that modify certain data types to ones Oracle accepts

Incompatible Type	Operator Used to Change It
Strings, fixed or variable length, longer than 2000 bytes	modify See "Modify Operator" .
Subrecord	promotesubrec See "The promotesubrec Operator" .
Tagged aggregate	tagbatch See "The tagbatch Operator" .

Data type conversion

WebSphere DataStage data types are converted to Oracle data types as listed in the next table:

WebSphere DataStage Data Type	Oracle Data Type
date	DATE
decimal[p,s] p is decimal's precision and s is decimal's scale	NUMBER[p,s]
int8/uint8	NUMBER[3,0]
int16/uint16	NUMBER[5,0]
int32/uint32	NUMBER[10,0]

WebSphere DataStage Data Type	Oracle Data Type
int64	NUMBER[19,0]
uint64	NUMBER[20,0]
sfloat	NUMBER
dfloat	NUMBER
raw	not supported
fixed-length string, in the form string[n] or ustring[n], length <= 255 bytes	CHAR(n) where n is the string length
variable-length string, in the form string[max=n] or ustring[max=n] maximum length <= 2096 bytes	VARCHAR(maximum = n) where n is the maximum string length
variable-length string in the form string or ustring	VARCHAR(32)*
* The default length of VARCHAR is 32 bytes. This means all records of the table have 32 bytes allocated for each variable-length string field in the input data set. If an input field is longer than 32 bytes, the operator issues a warning. The -stringlength option modifies the default length.	
string, 2096 bytes < length	not supported
time	DATE (does not support microsecond resolution)
timestamp	DATE (does not support microsecond resolution)

WebSphere DataStage data types not listed in this table generate an error. Invoke the modify operator to perform type conversions. See "Modify Operator". All WebSphere DataStage and Oracle integer and floating-point data types have the same range and precision, and you need not worry about numeric overflow or underflow.

Write modes

The write mode of the operator determines how the records of the data set are inserted into the destination table. The write mode can have one of the following values:

- append: This is the default mode. The table must exist and the record schema of the data set must be compatible with the table. The write operator appends new rows to the table. The schema of the existing table determines the input interface of the operator.
- create: The operator creates a new table. If a table exists with the same name as the one you want to create, the step that contains the operator terminates with an error. The schema of the new table is determined by the schema of the WebSphere DataStage data set. The table is created with simple default properties. To create a table that is partitioned, indexed, in a non-default table space, or in some other non-standard way, you can use the -createtstmt option with your own create table statement.
- replace: The operator drops the existing table and creates a new one in its place. If a table exists of the same name as the one you want to create, it is overwritten. The schema of the new table is determined by the schema of the WebSphere DataStage data set.
- truncate: The operator retains the table attributes but discards existing records and appends new ones. The schema of the existing table determines the input interface of the operator.

Note: If a previous write operation fails, you can retry your application specifying a write mode of replace to delete any information in the output table that might have been written by the previous attempt to run your program.

Each mode requires the specific user privileges shown in the table below:

Write Mode	Required Privileges
append	INSERT on existing table
create	TABLE CREATE
replace	INSERT and TABLE CREATE on existing table
truncate	INSERT on existing table

Matched and unmatched fields

The schema of the Oracle table determines the operator's interface schema. Once the operator determines this, it applies the following rules to determine which data set fields are written to the table:

1. Fields of the input data set are matched by name with fields in the input interface schema.
WebSphere DataStage performs default data type conversions to match the input data set fields with the input interface schema.
You can also use the modify operator to perform explicit data type conversions. See "Modify Operator" for more information.
2. If the input data set contains fields that do not have matching components in the table, the operator generates an error and terminates the step.
This rule means that WebSphere DataStage does not add new columns to an existing table if the data set contains fields that are not defined in the table. Note that you can use either the orawrite -drop option or the modify operator to drop extra fields from the data set. See "Modify Operator" for more information.
3. Columns in the Oracle table that do not have corresponding fields in the input data set are set to their default value, if one is specified in the Oracle table. If no default value is defined for the Oracle column and it supports nulls, it is set to null. Otherwise, WebSphere DataStage issues an error and terminates the step.

WebSphere DataStage data sets support nullable fields. If you write a data set to an existing table and a field contains a null, the Oracle column must also support nulls. If not, WebSphere DataStage issues an error message and terminates the step. However, you can use the modify operator to convert a null in an input field to another value. See the topic on the modify operator for more information.

Orawrite: syntax and options

Syntax for the orawrite operator is given below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes. Exactly one occurrence of the -dboptions option and the -table option are required.

```
orawrite
 -dboptions
  '{user = username, password = password}'
  |
  '{user = \@file_name\}'
[-close close_command]
[-computeStats]
[-createtestmt create_statement]
[-drop]
[-db_cs icu_character_set] [-disableConstraints]
[-exceptionsTable exceptionsTableName]
[-index rebuild
    -computeStats
    -nologging
    |]
```

```

[-index maintenance]
[-mode create | replace | append | truncate]
[-nchar_cs icu_character_set]
[-open open_command]
[-ora8partition ora8part_name]
[-primaryKeys fieldname_1, fieldname_2, ... fieldname_n]
[-server remote_server_name]
[-stringlength length]
[-truncate]
[-useNchar]

```

Table 155. orawrite Operator Options

Option	Use
-close	<p>-close <i>close_command</i></p> <p>Specify any command, enclosed in single quotes, to be parsed and run by the Oracle database on all processing nodes when WebSphere DataStage completes the processing of the Oracle table.</p> <p>You can include an Oracle stored procedure as part of <i>close_command</i>, in the form:</p> <p>"execute <i>procedureName</i>;"</p>
-computeStats	<p>-computeStats</p> <p>This option adds Oracle's COMPUTE STATISTICS clause to the -index rebuild command.</p> <p>The -computeStatus and the -nologging options must be used in combination with the -index rebuild option as shown below:</p> <p>-index rebuild -computeStats -nologging</p>

Table 155. orawrite Operator Options (continued)

Option	Use
-createstmt	<p>-createstmt <i>create_statement</i></p> <p>Creates a table. You use this option only in create and replace modes. You must supply the create table statement, otherwise WebSphere DataStage attempts to create it based on simple defaults.</p> <p>Here is an example create statement:</p> <pre>'create table test1 (A number, B char(10), primary key (A))'</pre> <p>You do not add the ending semicolon, as would normally be acceptable with an SQL statement.</p> <p>When writing to a multibyte database, specify chars and varchars in bytes, with two bytes for each character. This example specifies 10 characters:</p> <pre>'create table orch_data(col_a varchar(20))'</pre> <p>When specifying nchar and nvarchar2 column types, specify the size in characters: This example, specifies 10 characters:</p> <pre>'create table orch_data(col_a nvarchar2(10))'</pre> <p>This option is mutually exclusive with the -primaryKeys option.</p>
-db_cs	<p>-db_cs <i>icu_character_set</i></p> <p>Specify an ICU character set to map between Oracle char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to Oracle. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data. If this option is specified, you must also specify the -nchar option.</p> <p>WebSphere DataStage maps your ICU character setting to its Oracle character-set equivalent. See "Mapping Between ICU and Oracle Character Sets" for the details.</p> <p>For information on national language support, reference this IBM ICU site:</p> <p>http://www.oss.software.ibm.com/icu/charset</p>

Table 155. orawrite Operator Options (continued)

Option	Use
-nchar_cs	<p>-nchar_cs <i>icu_character_set</i></p> <p>Specify an ICU character set to map between Oracle nchar and nvarchar2 values and WebSphere DataStage ustring data. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data. If this option is specified, you must also specify the -db_cs option.</p> <p>WebSphere DataStage maps your ICU character setting to its Oracle character-set equivalent. See "Mapping Between ICU and Oracle Character Sets" for the details.</p> <p>For information on national language support, reference this IBM ICU site:</p> <p>http://www.oss.software.ibm.com/icu/charset</p>
-dboptions	<p>-dboptions '{user=<i>username</i>, password=<i>password</i>} '{user='@\ifilename\'}'</p> <p>Specify either a user name and password for connecting to Oracle or a file containing the user name and password. These options are required by the operator.</p>
-disableConstraints	<p>-disableConstraints</p> <p>Disables all enabled constraints on a table, and then attempts to enable them again at the end of a load.</p> <p>When disabling the constraints, the cascade option is included. If Oracle cannot enable all the constraints, warning messages are displayed.</p> <p>If the -exceptionsTable option is included, ROWID information on rows that violate constraints are inserted into an exceptions table. In all cases, the status of all constraints on the table are displayed at the end of the operator run.</p> <p>This option applies to all write modes.</p>
-drop	<p>-drop</p> <p>Specify dropping unmatched fields of the WebSphere DataStage the data set. An unmatched field is a field for which there is no identically named field in the Oracle table.</p>

Table 155. orawrite Operator Options (continued)

Option	Use
-exceptionsTable	<p>-exceptionsTable <i>exceptionsTableName</i></p> <p>Specify an exceptions table. <i>exceptionsTableName</i> is the name of a table where record ROWID information is inserted if a record violates a table constraint when an attempt is made to enable the constraint.</p> <p>This table must already exist. It is not created by the operator. Your Oracle installation should contain a script that can be executed to create an exceptions table named exceptions.</p> <p>This option can be included only in conjunction with the -disableConstraints option.</p>
-index	<p>-index rebuild -computeStatus -nologging -maintenance</p> <p>Lets you perform a direct parallel load on an indexed table without first dropping the index. You can choose either the -maintenance or -rebuild option, although special rules apply to each (see below). The -index option is applicable only in append and truncate write modes, and in create mode only if a -createtestmt option is provided.</p> <p>rebuild: Skips index updates during table load and instead rebuilds the indexes after the load is complete using the Oracle alter index rebuild command. The table must contain an index, and the indexes on the table must not be partitioned.</p> <p>The -computeStatus and the -nologging options can be used only with the -index rebuild command. The -computeStatus option adds Oracle's COMPUTE STATISTICS clause to the -index rebuild command; and the -nologging option adds Oracle's NOLOGGING clause to the -index rebuild command.</p> <p>maintenance: Results in each table partition being loaded sequentially. Because of the sequential load, the table index that exists before the table is loaded is maintained after the table is loaded. The table must contain an index and be partitioned, and the index on the table must be a local range-partitioned index that is partitioned according to the same range values that were used to partition the table. Note that in this case sequential means sequential per partition, that is, the degree of parallelism is equal to the number of partitions.</p> <p>This option is mutually exclusive with the -primaryKeys option.</p>

Table 155. *orawrite* Operator Options (continued)

Option	Use
-mode	<p>-mode create replace append truncate</p> <p>Specify the write mode of the operator.</p> <ul style="list-style-type: none"> • append (default): New records are appended to the table. • create: Create a new table. WebSphere DataStage reports an error if the Oracle table already exists. You must specify this mode if the Oracle table does not exist. • truncate: The existing table attributes (including schema) and the Oracle partitioning keys are retained but existing records are discarded. New records are then appended to the table. • replace: The existing table is dropped and a new table is created in its place. Oracle uses the default partitioning method for the new table. <p>See "Write Modes" for a table of required privileges for each mode.</p>
-nologging	<p>-nologging</p> <p>This option adds Oracle's NOLOGGING clause to the -index rebuild command.</p> <p>This option and the -computeStats option must be used in combination with the -index rebuild option as shown below:</p> <p><code>-index rebuild -computeStats -nologging</code></p>
-open	<p>-open <i>open_command</i></p> <p>Specifies any command, enclosed in single quotes, to be parsed and run by the Oracle database on all processing nodes. WebSphere DataStage causes Oracle to run this command before opening the table. There is no default <i>open_command</i>.</p> <p>You can include an Oracle stored procedure as part of <i>open_command</i> in the form:</p> <p><code>"execute procedureName;"</code></p>
-ora8partition	<p>-ora8partition <i>ora8part_name</i></p> <p>Name of the Oracle 8 table partition that records are written to. The operator assumes that the data provided is for the partition specified.</p>

Table 155. orawrite Operator Options (continued)

Option	Use
-primaryKeys	<p>-primaryKeys <i>fieldname_1,fieldname_2,...fieldname_n</i></p> <p>This option can be used only with the create and replace write modes. The -createstmt option should not be included.</p> <p>Use the external column names for this option.</p> <p>By default, no columns are primary keys.</p> <p>This clause is added to the create table statement:</p> <p>constraint tablename_PK primary key (<i>fieldname_1,fieldname_2,..fieldname_n</i>)</p> <p>The operator names the primary key constraint "tablename_PK, where <i>tablename</i> is the name of the table. If the -disableConstraints option is not also included, the primary key index will automatically be rebuilt at the end of the load.</p> <p>This option is mutually exclusive with the -index and -createstmt option.</p>
-server	<p>-server <i>remote_server_name</i></p> <p><i>remote_server_name</i> must specify a remote connection.</p> <p>To specify a local connection, set your ORACLE_SID environment variable to a local server.</p>
-stringlength	<p>-stringlength <i>string_len</i></p> <p>Set the default string length for variable-length strings written to an Oracle table. If you do not specify a length, WebSphere DataStage uses a default size of 32 bytes. Variable-length strings longer than the set length cause an error.</p> <p>You can set the maximum length up to 2000 bytes.</p> <p>Note that the operator always allocates <i>string_len</i> bytes for a variable-length string. In this case, setting <i>string_len</i> to 2000 allocates 2000 bytes. Set <i>string_len</i> to the expected maximum length of your longest string.</p>
-table	<p>-table <i>table_name</i></p> <p>Specify the name of the Oracle table.</p> <p>If you are writing to the table, and the output mode is create, the table must not exist. If the output mode is append, replace, or truncate, the table must exist.</p> <p>The Oracle write operator cannot write to a table that has indexes defined on it. If you want to write to such a table, you must first delete the index(es) and recreate them after orawrite completes. The operator checks for indexes and returns an error if one is found.</p>

Table 155. orawrite Operator Options (continued)

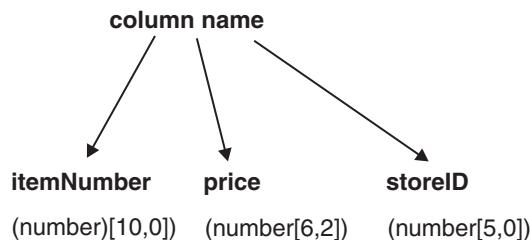
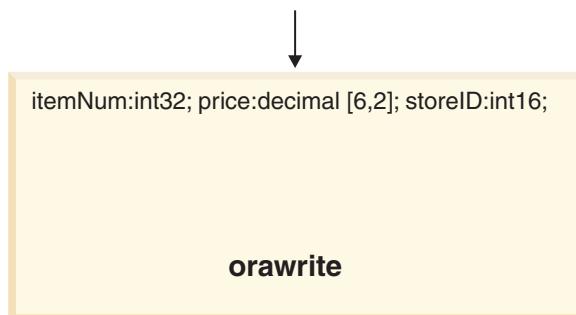
Option	Use
-truncate	<p>-truncate</p> <p>Configure the operator to truncate WebSphere DataStage field names to 30 characters.</p> <p>Oracle has a limit of 30 characters for column names. By default, if an WebSphere DataStage field name is too long, WebSphere DataStage issues an error. The -truncate option overrides this default.</p>
-useNchar	Allows the creation of tables with nchar and nvarchar2 fields. This option only has effect when the -mode option is create. If you do not specify the -useNchar option, the table is created with char and varchar fields.

Example 1: writing to an existing Oracle table

When an existing Oracle table is written to:

- The column names and data types of the Oracle table determine the input interface schema of the write operator.
- This input interface schema then determines the fields of the input data set that is written to the table.

For example, the following figure shows the orawrite operator writing to an existing table:



The record schema of the WebSphere DataStage data set and the row schema of the Oracle table correspond to one another, and field and column names are identical. Here are the input WebSphere DataStage record schema and output Oracle row schema:

Input WebSphere DataStage Record	Output Oracle Table
itemNum :int32; price:decimal[3,2]; storeID:int16;	price NUMBER[10,0] itemNum NUMBER[6,2] storeID NUMBER[5,0]

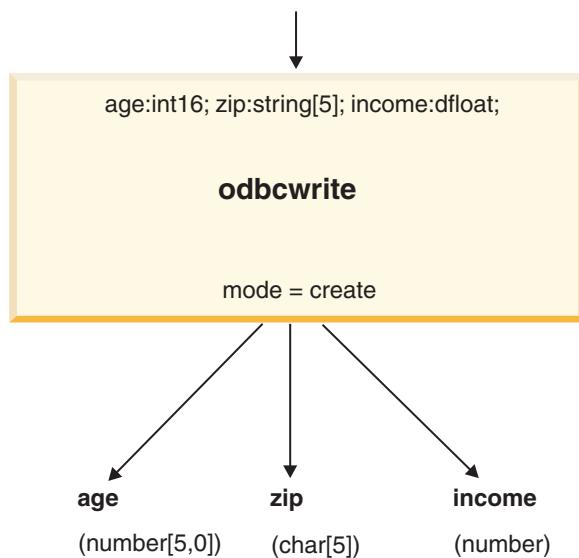
Here is the osh syntax for this example:

```
$ osh "... op1 | orawrite -table 'table_2'
    -dboptions {'user = user101, password =
        userPword'}"
```

Note that since the write mode defaults to append, the mode option does not appear in the command.

Example 2: creating an Oracle table

To create a table, specify a write mode of either create or replace. The next figure is a conceptual diagram of the create operation.



Here is the osh syntax for this operator:

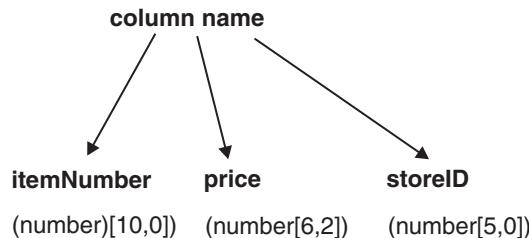
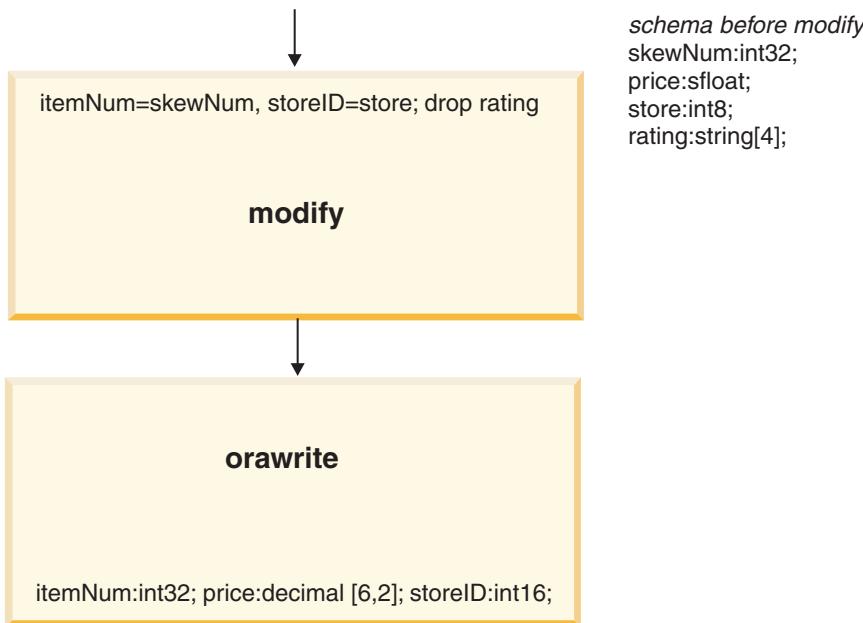
```
$ osh "... orawrite -table table_2
    -mode create
    -dboptions {'user = user101, password =
        userPword'} ..."
```

The orawrite operator creates the table, giving the Oracle columns the same names as the fields of the input WebSphere DataStage data set and converting the WebSphere DataStage data types to Oracle data types. See "Data Type Conversion" for a list of WebSphere DataStage-to-Oracle conversions.

Example 3: writing to an Oracle table using the modify operator

The modify operator allows you to drop unwanted fields from the write operation and to translate the name or data type of a field of the input data set to match the input interface schema of the operator.

The next example uses the modify operator:



In this example, you use the modify operator to:

- Translate field names of the input data set to the names of corresponding fields of the operator's input interface schema, that is skewNum to itemNum and store to storeID.
- Drop the unmatched rating field, so that no error occurs.

Note that WebSphere DataStage performs automatic type conversion of store, promoting its int8 data type in the input data set to int16 in the orawrite input interface.

Here is the osh syntax for this operator:

```
$ modifySpec="itemNum = skewNum, storeID = store;drop rating"
$ osh "... op1 | modify '$modifySpec'
| orawrite -table table_2
-dboptions {'user = user101, password =
userPword'}"
```

See "Modify Operator" information on using that operator.

The oraupsert operator

The oraupsert operator inserts and updates Oracle table records with data contained in a WebSphere DataStage data set. You provide the insert statement using the -insert option and provide the update and delete SQL statements using the -update option.

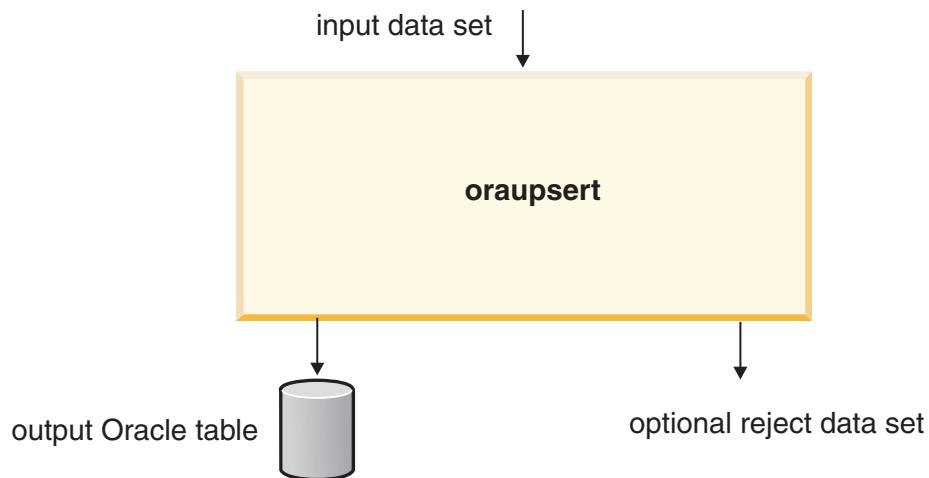
An example -update delete statement is:

```
-update 'delete from tablename where A = ORCHESTRATE.A'
```

This operator receives a single data set as input and writes its output to an Oracle table. You can request an optional output data set that contains the records that fail to be inserted or updated. The output reject link sqlcode field contains an int32 value which corresponds to the Oracle error code for the record.

By default, oraupsert uses Oracle host-array processing to optimize the performance of inserting records.

Data flow diagram



oraupsert: properties

Table 156. oraupsert properties

Property	Value
Number of input data sets	1
Number of output data sets	by default, none; 1 when you select the -reject option
Input interface schema	derived from your insert and update statements
Transfer behavior	Rejected update records are transferred to an output data set when you select the -reject option.
Execution mode	parallel by default, or sequential
Partitioning method	same You can override this partitioning method; however, a partitioning method of entire cannot be used.
Collection method	any
Combinable operator	yes

Operator Action

Here are the main characteristics of oraupsert:

- An -update statement is required. The -insert and -reject options are mutually exclusive with using an -update option that issues a delete statement.
- If an -insert statement is included, the insert is executed first. Any records that fail to be inserted because of a unique-constraint violation are then used in the execution of the update statement.

- WebSphere DataStage uses host-array processing by default to enhance the performance of insert array processing. Each insert array is executed with a single SQL statement. Update records are processed individually.

- You use the `-insertArraySize` option to specify the size of the insert array. For example:

```
-insertArraySize 250
```

The default length of the insert array is 500. To direct WebSphere DataStage to process your insert statements individually, set the insert array size to 1:

```
-insertArraySize 1
```

- Your record fields can be variable-length strings. You can specify a maximum length or use the default maximum length of 80 characters.

This example specifies a maximum length of 50 characters:

```
record(field1:string[max=50])
```

The maximum length in this example is, by default, 80 characters:

```
record(field1:string)
```

- When an insert statement is included and host array processing is specified, a WebSphere DataStage update field must also be a WebSphere DataStage insert field.

- The `oraupsert` operator converts all values to strings before passing them to Oracle. The following WebSphere DataStage data types are supported:

- `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `int64`, and `uint64`
- `dfloat` and `sfloat`
- `decimal`
- strings of fixed and variable length
- `timestamp`
- `date`

- By default, `oraupsert` produces no output data set. By using the `-reject` option, you can specify an optional output data set containing the records that fail to be inserted or updated. Its syntax is:

```
-reject filename
```

For a failed insert record, these sqlcodes cause the record to be transferred to your reject data set:

```
-1400: cannot insert NULL
-1401: inserted value too large for column
-1438: value larger than specified precision allows for this column
-1480: trailing null missing from string bind value
```

Note: An insert record that fails because of a unique constraint violation (sqlcode of -1) is used for updating.

For a failed update record, these sqlcodes cause the record to be transferred to your reject data set:

```
-1: unique constraint violation
-1401: inserted value too large for column
-1403: update record not found
-1407: cannot update to null
-1438: value larger than specified precision allows for this column
-1480: trailing null missing from string bind value
```

A -1480 error occurs when a variable length string is truncated because its length is not specified in the input data set schema and it is longer than the default maximum length of 80 characters.

Note: When a record fails with an sqlcode other than those listed above, `oraupsert` also fails. Therefore, you must backup your Oracle table before running your data flow.

Associated environment variables

There are two environment variables associated with the `oraupsert` operator. They are:

- APT_ORAUPSERT_COMMIT_TIME_INTERVAL: You can reset this variable to change the time interval between Oracle commits; the default value is 2 seconds.
- APT_ORAUPSERT_COMMIT_ROW_INTERVAL: You can reset this variable to change the record interval between Oracle commits; the default value is 5000 records.

Commits are made whenever the time interval period has passed or the row interval is reached, whichever occurs first.

Oraupsert: syntax and options

The syntax for oraupsert is shown below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
oraupsert
-dboptions '{user = username, password = password}'
| '{user = \@file_name\}'
-update update_or_delete_statement
[-update_first]
[-db_cs icu_character_set] [-nchar_cs icu_character_set]
[-insert insert_statement]
[-insertArraySize n]
[-reject]
[-server remote_server_name]
```

Exactly one occurrence of the -dboptions option and exactly one occurrence of the -update are required. All other options are optional.

Table 157. Oraupsert options

Options	Value
-dboptions	-dboptions '{user= <i>username</i> , password= <i>password</i> }' '{user=\ '@filename\''}
-db_cs	-db_cs <i>icu_character_set</i> Specify an ICU character set to map between Oracle char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to Oracle. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data. If this option is specified, the -nchar option must also be specified. WebSphere DataStage maps your ICU character setting to its Oracle character-set equivalent. See "Mapping Between ICU and Oracle Character Sets" for the details. For information on national language support, reference this IBM ICU site: http://www.oss.software.ibm.com/icu/charset

Table 157. Oraupsert options (continued)

Options	Value
-nchar_cs	<p>-nchar_cs <i>icu_character_set</i></p> <p>Specify an ICU character set to map between Oracle nchar and nvarchar2 values and WebSphere DataStage ustring data. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data. If this option is specified, the -db_cs option must also be specified.</p> <p>WebSphere DataStage maps your ICU character setting to its Oracle character-set equivalent. See "Mapping Between ICU and Oracle Character Sets" for the details.</p> <p>For information on national language support, reference this IBM ICU site:</p> <p>http://www.oss.software.ibm.com/icu/charset</p>
-insert	<p>-insert <i>insert_statement</i></p> <p>Optionally specify the insert statement to be executed.</p> <p>This option is mutually exclusive with using an -update option that issues a delete statement.</p>
-insertArraySize	<p>-insertArraySize <i>n</i></p> <p>Specify the size of the insert host array. The default size is 500 records. If you want each insert statement to be executed individually, specify 1 for this option.</p>
-reject	<p>-reject <i>filename</i></p> <p>If this option is set, records that fail are written to a reject data set. You must designate an output data set for this purpose.</p> <p>This option is mutually exclusive with using an -update option that issues a delete statement.</p>
-server	<p>-server <i>remote_server_name</i></p> <p><i>remote_server_name</i> must specify a remote connection.</p> <p>To specify a local connection, set your ORACLE_SID environment variable to a local server.</p>
-update	<p>-update <i>update_or_delete_statement</i></p> <p>Use this required option to specify the update or delete statement to be executed. An example delete statement is:</p> <p>-update 'delete from tablename where A = ORCHESTRATE.A'</p> <p>A delete statement cannot be issued when using the -insert or -reject option.</p>
-update_first	<p>-update_first</p> <p>Specify this option to have the operator update first and then insert.</p>

Example

This example updates an Oracle table that has two columns: acct_id and acct_balance, where acct_id is the primary key. Input is from a text file.

Note: The records are hashed and sorted on the unique key, acct_id. This ensures that records with the same account ID are in the same partition which avoids non-deterministic processing behavior that can lead to deadlocks.

Two of the records cannot be inserted because of unique key constraints; instead, they are used to update existing records. One record is transferred to the reject data set because its acct_id generates an -1438 sqlcode due to its length exceeding the schema-specified length.

Summarized below are the state of the Oracle table before the dataflow is run, the contents of the input file, and the action WebSphere DataStage performs for each record in the input file.

Table before dataflow		Input file contents		WebSphere DataStage action
acct_id	acct_balance	acct_id	acct_balance	
073587	45.64	873092	67.23	update
873092	2001.89	865544	8569.23	insert
675066	3523.62	566678	2008.56	update
566678	89.72	678888	7888.23	insert
		073587	82.56	update
		995666	75.72	insert

osh syntax

```
$ osh "import -schema record(acct_id:string[6]; acct_balance:dfloat;)
      -file input.txt |
      hash -key acct_id |
      tsort -key acct_id |
      oraupsert -dboptions '{user=apt, password=test}' |
      -insert 'insert into accounts
              values(ORCHESTRATE.acct_id,
                     ORCHESTRATE.acct_balance)'
      -update 'update accounts
              set acct_balance = ORCHESTRATE.acct_balance
              where acct_id = ORCHESTRATE.acct_id'
      -reject '/user/home/reject/reject.ds'"
```

Table after dataflow

acct_id	acct_balance
073587	82.56
873092	67.23
675066	3523.62
566678	2008.56
865544	8569.23
678888	7888.23
7888.23	
995666	75.72

The oralookup operator

With the oralookup operator, you can perform a join between one or more Oracle tables and a WebSphere DataStage data set. The resulting output data is a WebSphere DataStage data set containing WebSphere DataStage and Oracle data.

You perform this join by specifying either an SQL SELECT statement, or by specifying one or more Oracle tables and one or more key fields on which to do the lookup.

This operator is particularly useful for sparse lookups, that is, where the WebSphere DataStage data set you are matching is much smaller than the Oracle table. If you expect to match 90% of your data, using the oraread and lookup operators is probably more efficient.

Because oralookup can do lookups against more than one Oracle table, it is useful for joining multiple Oracle tables in one query.

The -statement option command corresponds to an SQL statement of this form:

```
select a,b,c from data.testtbl  
  where  
    orchestrate.b = data.testtbl.c  
  and  
    orchestrate.name = "Smith"
```

The operator replaces each `orchestrate.fieldname` with a field value, submits the statement containing the value to Oracle, and outputs a combination of Oracle and WebSphere DataStage data.

Alternatively, you can use the -key/-table options interface to specify one or more key fields and one or more Oracle tables. The following osh options specify two keys and a single table:

```
-key a -key b -table data.testtbl
```

you get the same result as you would by specifying:

```
select * from data.testtbl  
where  
orchestrate.a = data.testtbl.a  
and  
orchestrate.b = data.testtbl.b
```

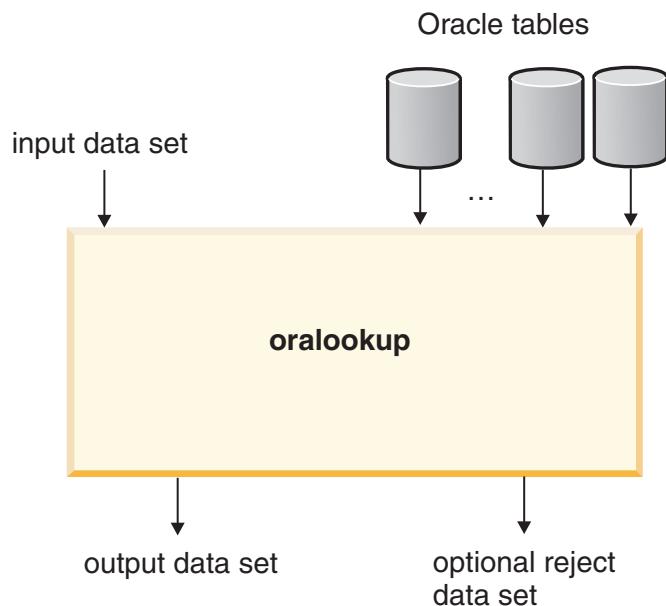
The resulting WebSphere DataStage output data set includes the WebSphere DataStage records and the corresponding rows from each referenced Oracle table. When an Oracle table has a column name that is the same as a WebSphere DataStage data set field name, the Oracle column is renamed using the following syntax:

`APT_integer_fieldname`

An example is `APT_0_lname`. The integer component is incremented when duplicate names are encountered in additional tables.

Note: If the Oracle table is not indexed on the lookup keys, the performance of this operator is likely to be poor.

Data flow diagram



Properties

Table 158. `oralookup` Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1; 2 if you include the <code>-ifNotFound</code> reject option
Input interface schema	determined by the query
Output interface schema	determined by the SQL query
Transfer behavior	transfers all fields from input to output
Execution mode	sequential or parallel (default)
Preserve-partitioning flag in output data set	clear
Composite operator	no

Oralookup: syntax and options

The syntax for the `oralookup` operator is given below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
oralookup
  -dboptions
    '{user = username, password = password}'
    |
    '{user = \0file_name\}'
  -table table_name -key field [-key field ...]
    [-table table_name -key field [-key field ...]]
  |
  -query sql_query
  [-part table_name]
  [-db_cs icu_character_set]
  [-nchar_cs icu_character_set]
  [-ifNotFound fail | drop | reject filename | continue]
  [-server remote_server]
```

You must specify either the -query option or one or more -table options with one or more -key fields. Exactly one occurrence of the -dboptions is required.

Table 159. oralookup Operator Options

Option	Use
<code>-dboptions</code>	<code>-dboptions</code> <code>'{user=username, password=password}' </code> <code>'{user=@filename\}'</code>
<code>-db_cs</code>	<code>-db_cs icu_character_set</code> Specify an ICU character set to map between Oracle char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to Oracle. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data. If this option is specified, you must also specify the -nchar option. WebSphere DataStage maps your ICU character setting to its Oracle character-set equivalent. See "Mapping Between ICU and Oracle Character Sets" for the details. For information on national language support, reference this IBM ICU site: http://www.oss.software.ibm.com/icu/charset
<code>-nchar_cs</code>	<code>-nchar_cs icu_character_set</code> Specify an ICU character set to map between Oracle nchar and nvarchar2 values and WebSphere DataStage ustring data. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data. If this option is specified, you must also specify the -db_cs option. WebSphere DataStage maps your ICU character setting to its Oracle character-set equivalent. See "Mapping Between ICU and Oracle Character Sets" for the details. For information on national language support, reference this IBM ICU site: http://www.oss.software.ibm.com/icu/charset
<code>-ifNotFound</code>	<code>-ifNotFound fail drop reject filename continue</code> Optionally determines what action to take in case of a lookup failure. The default is fail. If you specify reject, you must designate an additional output data set for the rejected records.

Table 159. *oralookup* Operator Options (continued)

Option	Use
-query	<p>-query <i>sql_query</i></p> <p>Specify an SQL query, enclosed in single quotes, to read a table. The query specifies the table and the processing that you want to perform on the table as it is read into WebSphere DataStage. This statement can contain joins, views, database links, synonyms, and so on.</p> <p>-statement '<i>statement</i>'</p> <p>A select statement corresponding to this lookup.</p>
-server	<p>-server <i>remote_server</i></p> <p><i>remote_server_name</i> must specify a remote connection.</p> <p>To specify a local connection, set your ORACLE_SID environment variable to a local server.</p>
-table	<p>-table <i>table_name</i> -key1 <i>field1</i> ...</p> <p>Specify a table and key field(s) from which a SELECT statement is created. This is equivalent to specifying the SELECT statement directly using the -statement option</p> <p>You can specify multiple instances of this option.</p>

Example

Suppose you want to connect to the APT81 server as user user101, with the password test. You want to perform a lookup between a WebSphere DataStage data set and a table called target, on the key fields lname, fname, and DOB. You can configure oralookup in either of two ways to accomplish this.

Here is the osh command using the -table and -key options:

```
$ osh " oralookup -dboptions {'user = user101, password = test'}
      -server APT81
      -table target -key lname -key fname -key DOB
      < data1.ds > data2.ds "
```

Here is the equivalent osh command using the -query option:

```
$ osh " oralookup -dboptions {'user = user101, password = test'}
      -server APT81
      -query 'select * from target
              where lname = Orchestrate.lname
              and fname = Orchestrate.fname
              and DOB = Orchestrate.DOB'
      < data1.ds > data2.ds "
```

WebSphere DataStage prints the lname, fname, and DOB column names and values from the WebSphere DataStage input data set and also the lname, fname, and DOB column names and values from the Oracle table.

If a column name in the Oracle table has the same name as a WebSphere DataStage output data set schema fieldname, the printed output shows the column in the Oracle table renamed using this format:
APT_integer_fieldname

For example, lname might be renamed to APT_0_lname.

Chapter 17. The DB2 interface library

The DB2 library contains five operators used to access DB2 databases.

- The db2read operator sets up a connection to a DB2 server, sends it a query, and converts the resulting two-dimensional array (the DB2 result set) to an WebSphere DataStage data set. This operator allows you to read an existing DB2 table or to explicitly query the DB2 database.
- The db2write and db2load operator write data to DB2.
- The db2upsert operator updates DB2 table records with data contained in a WebSphere DataStage data set.
- The db2part operator allows an operator to partition an input data set just as DB2 does.
- With the db2lookup operator, you can perform a join between one or more DB2 tables and a WebSphere DataStage data set.

All DB2 operators support real time integration, and jobs using them can be deployed as services.

Configuring WebSphere DataStage access

This section assumes that WebSphere DataStage users have been configured to access DB2 using the DB2 configuration process.

User privileges and WebSphere DataStage settings must be correct to ensure DB2 access. You must make sure your environment is configured correctly. WebSphere DataStage connects to DB2 using your WebSphere DataStage user name and password. Users who perform read and write operations must have valid accounts and appropriate privileges on the databases to which they connect. The following table lists the required DB2 privileges.

Table 160. Required DB2 Privileges

Operator	Required Privileges
db2read	SELECT on the table to be read
db2write, db2load, and db2upsert	For append or truncate mode, INSERT on an existing table For create mode, TABLE CREATE For replace mode, INSERT and TABLE CREATE on an existing table
db2load	DBADM on the database written by the operator If you are a DB2 administrator, you can grant this privilege in several ways. One way is to start DB2, connect to a database, and grant DBADM privilege to a user: <pre>db2> CONNECT TO db_name db2> GRANT DBADM ON DATABASE TO USER user_name</pre> where <i>db_name</i> is the name of the DB2 database and <i>user_name</i> is the login name of the WebSphere DataStage user. If you specify the -msgFile option, the database instance must have read/write privilege on the file.

The DB2 environment variable, DB2INSTANCE, specifies the user name of the owner of the DB2 instance. DB2 uses DB2INSTANCE to determine the location of db2nodes.cfg. For example, if you set DB2INSTANCE to "Mary", the location of db2nodes.cfg is ~Mary/sqllib/db2nodes.cfg.

The three methods of specifying the default DB2 database are listed here in order of precedence:

1. The -dbname option of the DB2 Interface read and write operators
2. The WebSphere DataStage environment variable APT_DBNAME
3. The DB2 environment variable DB2DBDFT

Establishing a remote connection to a DB2 server

You make a remote connection to DB2 on the conductor node. The players are run locally. In order to remotely connect from WebSphere DataStage to a remote DB2 server, WebSphere DataStage and DB2 need to be configured for remote-connection communication.

Follow these steps:

1. In your configuration file, include the node for the client and the node for the remote platform where DB2 is installed.
2. Set the client instance name using the -client_instance option.
3. Optionally set the server using the -server option. If the server is not set, you must set the DB2INSTANCE environment variable to the DB2 instance name.
4. Set the remote server database name using the -dbname option.
5. Set the client alias name for the remote database using the -client_dbname option.
6. Set the user name and password for connecting to DB2, using the -user and -password suboptions.

Note: If the name in the -user suboption of the -client_instance option differs from your UNIX user name, use an owner-qualified name when specifying the table for the -table, -part, -upsert, -update, and -query options. The syntax is:

owner_name.table_name

For example:

db2instance1.data23A.

Handling # and \$ characters in DB2 column names

The DB2 operators accept the # and \$ characters for DB2 table column names. WebSphere DataStage converts these two reserved characters into an internal format when they are written to the DB2 database and reconverts them when they are read from DB2.

The internal representation for the # character is '_035_' and the internal representation for the \$ character is '_036_'. You should avoid using these internal strings in your DB2 column names.

Using the external representation

Use the external representation (with the # and \$ characters) when referring to table column names for these options:

- The where and selectlist options of db2read and db2lookup
- The selectlist options of db2write and db2load
- The query option of db2read and db2lookup
- The upsert and update options

Using the internal representation

- Use the internal representation (with the _035_ and _036_ strings) when referring to table column names for these options:

- The query option of db2lookup when referring to a WebSphere DataStage name. For example:

```
-query 'select * from tablename where $A# =
ORCHESTRATE._036_A_035_
and ##B$ = ORCHESTRATE._035__035_B_036_'
```
- The upsert and update options when referring to a WebSphere DataStage name. For example:

```
-insert 'INSERT INTO tablename (A#,B$#)
VALUES (ORCHESTRATE.A_035_, ORCHESTRATE.B_036__035_)'
-update 'UPDATE tablename set B$# = ORCHESTRATE.B_036__035_
WHERE (A# =
ORCHESTRATE.A_035_)'
```

Using the -padchar option

Use the -padchar option of the db2upsert and db2lookup operators to pad string and ustring fields that are less than the length of the DB2 CHAR column.

Use this option for string and ustring fields that are inserted in DB2 or are used in the WHERE clause of an UPDATE, DELETE, or SELECT statement when all three of these conditions are met:

1. The UPDATE or SELECT statement contains string or ustring fields that map to CHAR columns in the WHERE clause.
2. The length of the string or ustring field is less than the length of the CHAR column.
3. The padding character for the CHAR columns is not the null terminator.

For example, if you add rows to a table using an INSERT statement in SQL, DB2 automatically pads CHAR fields with spaces. When you use the db2upsert operator to update the table or the db2lookup operator to query the table, you must use the -padchar option with the value of a space in order to produce the correct results. Use this syntax:

```
-padchar ' ' | 0xS20
```

The db2write operator automatically pads with null terminators, and the default pad character for the db2upsert and db2lookup operators is the null terminator. Therefore, you do not need to include the -padchar option when the db2write operator is used to insert rows into the table and the db2upsert or db2lookup operators are used to update or query that table.

The -padchar option should not be used if the CHAR column in the table is a CHAR FOR BIT type, and any of the input records contain null terminators embedded within the string or ustring fields.

Running multiple DB2 interface operators in a single step

Using WebSphere DataStage's remote-connection capabilities, you can run multiple DB2-interface operators in the same step, where each operator can be connected to its own DB2 server. Here is an osh example of this functionality:

```
osh "db2read -table table_1 -server DB2instance_name_1
-database database_1
|
db2lookup -table table_2 -key account
-client_instance DB2instance_1
-server remote_server_DB2instance_2
-database remote_database_1
-client_dbname alias_database_2
-user username -password passwd
-ifNotFound reject >| lookup.out
|
db2load -table table_3 -mode replace
-server remote_server_DB2instance_3
-database remote_database_2 -client_instance DB2instance_1
-client_dbname alias_database_3 -user username
-password passwd"
```

In this example, db2read reads from table_1 of database_1 in DB2instance_1.

The results read from table_1 are then piped to db2lookup to table_2 of remote database_1 in remote_server_DB2instance_2.

The records rejected by db2lookup are then piped to db2load where they are put in table_3 of remote_database_3 in remote_server_DB2instance_3.

National Language Support

WebSphere DataStage National Language Support (NLS) makes it possible for you to process data in international languages using Unicode character sets.

WebSphere DataStage uses International Components for Unicode (ICU) libraries to support NLS functionality. For information on national language support, see *WebSphere DataStage NLS Guide* and access the ICU home page:

<http://oss.software.ibm.com/developerworksopensource/icu/project>

Specifying character settings

In DB2 these components can contain multi-byte Unicode characters: database names; database objects such as tables, views, and indices; host variables; cursors; program labels; CHAR and Varchar columns; and Graphic and Vargraphic columns.

The -db_cs option of the DB2 read and write operators let you determine which ICU character set WebSphere DataStage uses to map between DB2 char and Varchar values and WebSphere DataStage ustring values, and to map DLL and query statements for output to DB2.

In \$APT_ORCHHOME/etc/db2_cs.txt, these mappings between ICU character sets and DB2 code pages are listed:

ICU character setting	DB2 code page
eucJP	954
ISO-8859-1	819
UTF-8	1208

If your -db_cs ICU character setting is listed in db2_cs.txt, WebSphere DataStage sets the DB2CODEPAGE environment variable to the corresponding code page. If your ICU character setting is not in db2_cs.txt, you can add it to that file or set the DB2CODEPAGE environment variable yourself.

If there is no code-page mapping in db2_cs.txt and DB2CODEPAGE is not set, WebSphere DataStage uses the DB2 defaults. DB2 converts between your character setting and the DB2 code page, provided that the appropriate language support is set in your operating system; however, no conversion is performed for the data of the db2load operator. Refer to your DB2 documentation for DB2 national language support.

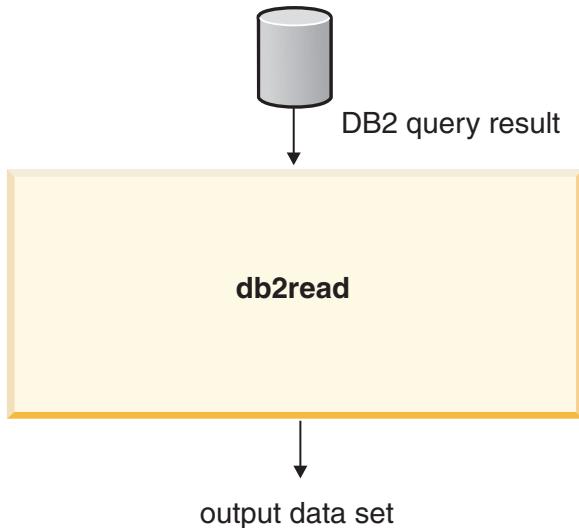
Preventing character-set conversion

The -use_strings option of the db2read and db2lookup operators direct WebSphere DataStage to import DB2 char and Varchars to WebSphere DataStage as WebSphere DataStage strings without converting them from their ASCII or binary form. This option overrides the db_cs option which converts DB2 char and Varchars as ustrings using a specified character set.

The db2read operator

The db2read operator sets up a connection to a DB2 server, sends it a query, and converts the resulting two-dimensional array (the DB2 result set) to a WebSphere DataStage data set. The operator allows you to either read an existing DB2 table or to explicitly query the DB2 database.

Data flow diagram



db2read: properties

Table 161. db2read Operator Properties

Property	Value
Number of input data sets	0
Number of output data sets	1
Input interface schema	none
Output interface schema	determined by the SQL query.
Transfer behavior	none
Execution mode	determined by options passed to it (see "Targeting the Read Operation")
Partitioning method	not applicable
Collection method	not applicable
Preserve-partitioning flag in output data set	clear
Composite operator	no

Operator action

Here are the chief characteristics of the db2read operator:

- The read operation is carried out on either the default database or a database specified by another means. See "Configuring WebSphere DataStage Access".
- Its output is an WebSphere DataStage data set that you can use as input to a subsequent WebSphere DataStage operator. The output data set is partitioned in the same way as the input DB2 table.

- It translates the query's result set row by row to a WebSphere DataStage data set, as described in "Conversion of a DB2 Result Set to a WebSphere DataStage Data Set".
- The translation includes the conversion of DB2 data types to WebSphere DataStage data types, as listed in "DB2 Interface Operator Data Type Conversions".
- It specifies either a DB2 table to read or an SQL query to carry out.
 - If it specifies a read operation of a table, the operation is executed in parallel, one instance of db2read on every partition of the table. See "Specifying the DB2 Table Name".
 - If it specifies a query, the operation is executed sequentially, unless the query directs it to operate in parallel. See "Explicitly Specifying the Query".
- It optionally specifies commands to be executed on all processing nodes before the read operation is performed and after it has completed.

Note: An RDBMS such as DB2 does not guarantee deterministic ordering behavior unless an SQL operation constrains it to do so. Thus, if you read the same DB2 table multiple times, DB2 does not guarantee delivery of the records in the same order every time. In addition, while DB2 allows you to run queries against tables in parallel, not all queries should be run in parallel. For example, some queries perform an operation that must be carried out sequentially, such as a group-by operation or a non-collocated join operation. These types of queries should be executed sequentially in order to guarantee correct results and finish in a timely fashion.

Conversion of a DB2 result set to a WebSphere DataStage data set

A DB2 result set is defined by a collection of rows and columns. Here is how the db2read operator translates the query's result set to a WebSphere DataStage data set:

- The rows of a DB2 result set correspond to the records of a WebSphere DataStage data set.
- The columns of a DB2 row correspond to the fields of a WebSphere DataStage record, and the name and data type of a DB2 column correspond to the name and data type of a WebSphere DataStage field.
- Names are translated exactly except when a component of a DB2 column name is not compatible with WebSphere DataStage naming conventions which place no limit on field-name length, but have the following restrictions:
 - The name must start with a letter or underscore (_) character.
 - The name can contain only alphanumeric and underscore characters.
 - The name is case insensitive.

When there is an incompatibility, WebSphere DataStage converts the DB2 column name as follows:

- If the DB2 column name does not begin with a letter or underscore, the string "APT_column#" (two underscores) is prepended to the column name, where *column#* is the number of the column. For example, if the third DB2 column is named 5foo, the WebSphere DataStage field is named *APT_35foo*.
- If the DB2 column name contains a character that is not alphanumeric or an underscore, the character is replaced by two underscore characters.
- Both DB2 columns and WebSphere DataStage fields support nulls, and a null contained in a DB2 column is stored as a null in the corresponding WebSphere DataStage field.
- The DB2 read operators convert DB2 data types to WebSphere DataStage data types, as in the next table:

Table 162. DB2 Interface Operator Data Type Conversions

DB2 Data Type	WebSphere DataStage Data Type
CHAR(<i>n</i>)	string[<i>n</i>] or ustring[<i>n</i>]
CHARACTER VARYING(<i>n,r</i>)	string[max = <i>n</i>] or ustring[max = <i>n</i>]
DATE	date

Table 162. DB2 Interface Operator Data Type Conversions (continued)

DB2 Data Type	WebSphere DataStage Data Type
DATETIME	time or timestamp with corresponding fractional precision for time If the DATETIME starts with a year component, the result is a timestamp field. If the DATETIME starts with an hour, the result is a time field.
DECIMAL[p,s]	decimal[p,s] where p is the precision and s is the scale The maximum precision is 32, and a decimal with floating scale is converted to a dfloat.
DOUBLE-PRECISION	dfloat
FLOAT	dfloat
INTEGER	int32
MONEY	decimal
GRAPHIC(n)	string[n] or ustring[n]
VARGRAPHIC(n)	string[max = n] or ustring[max = n]
LONG VARGRAPHIC	ustring
REAL	sfloat
SERIAL	int32
SMALLFLOAT	sfloat
SMALLINT	int16
VARCHAR(n)	string[max = n] or ustring[max = n]

Note: Data types that are not listed in the table shown above generate an error.

Targeting the read operation

You can read a DB2 table using one of these methods:

- Specify the table name and allow WebSphere DataStage to generate a default query that reads the entire table.
- Explicitly specify the query.

Specifying the DB2 table name

If you use the -table option to pass the table name to db2read, the operator functions in parallel, one instance for each partition of the table.

WebSphere DataStage issues the following SQL SELECT statement to read the table:

```
select [selectlist]
      from tableName
     where nodenumber(colName)=current node
       [and (filter)];
```

You can specify optional parameters to narrow the read operation. They are as follows:

- selectlist specifies the columns of the table to be read. By default, WebSphere DataStage reads all columns.
- filter specifies the rows of the table to exclude from the read operation. By default, WebSphere DataStage reads all rows.

Note that the default query's WHERE clause contains the predicate:

```
nodenumber(colName)=current node
```

where *colName* corresponds to the first column of the table or select list. This predicate is automatically generated by WebSphere DataStage and is used to create the correct number of DB2 read operators for the number of partitions in the DB2 table.

You can optionally specify the -open and -close options. These commands are executed by DB2 on every processing node containing a partition of the table before the table is opened and after it is closed.

Explicitly specifying the query

If you choose the -query option, you pass an SQL query to the operator. The query specifies the table and the processing that you want to perform on the table as it is read into WebSphere DataStage. The SQL statement can contain joins, views, database links, synonyms, and so on. However, the following restrictions apply to -query:

- It cannot contain bind variables.
- By default, the query is executed sequentially (on a single processing node). It can also be executed in parallel when you specify the -part option.

If you want to include a filter or select list, you must specify them as part of the query. You can optionally specify the -open and -close options. These commands are executed by DB2 on every processing node containing a partition of the table before the table is opened and after it is closed.

If you choose the -query option, the operator functions sequentially by default. It functions in parallel if two conditions are satisfied:

- The SQL query contains a WHERE clause providing information that enables WebSphere DataStage to run the query in parallel
- The -part *table_name* option and argument are specified, so that one instance of the operator is created on every processing node containing a partition of *table_name*.

Note: Complex queries, such as joins, executed in parallel might cause the database to consume large amounts of system resources.

Specifying open and close commands

You can optionally specify the -open and -close command options. These commands are executed by DB2 on every processing node containing a partition of the table before the table is opened and after it is closed. Their syntax is:

```
-open open_command-close close_command
```

If you do not specify an open command and the read operation is carried out in parallel, WebSphere DataStage runs the following default command:

```
lock table table_name in share mode;
```

where *table_name* is the table specified by the -table option.

This command locks the table until WebSphere DataStage finishes reading the table. The table cannot be written to when it is locked; however, if you specify an explicit open command, the lock statement is not run. When DB2 is accessed sequentially, WebSphere DataStage provides no default open command.

The close command is executed by the operator after WebSphere DataStage finishes reading the DB2 table and before it disconnects from DB2. If you do not specify a close command, the connection is immediately terminated after WebSphere DataStage finishes reading the DB2 table.

If DB2 has been accessed in parallel and -close is not specified, WebSphere DataStage releases the table lock obtained by the default *open_command*.

Db2read: syntax and options

You must specify either the -table or the -query option. All others are optional. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose the value in single quotes.

```
db2read
  -table table_name [-filter filter] [-selectlist list] |      -query sql_query
  [-part table_name]
  [-client_instance client_instance_name -client_dbname database
    -user user_name -password password]
  [-close close_command]
  [-db_cs character_set]
  [-dbname database_name]
  [-open open_command]
  [-server server_name]
  [-use_strings]
```

Table 163. db2read Options

Option	Use
-client_instance	<p>-client_instance <i>client_instance_name</i> [-client_dbname <i>database</i>] -user <i>user_name</i> -password <i>password</i></p> <p>Specifies the client DB2 instance name. This option is required for a remote connection.</p> <p>The -client_dbname suboption specifies the client database alias name for the remote server database. If you do not specify this option, WebSphere DataStage uses the value of the -dbname option, or the value of the APT_DBNAME environment variable, or DB2DBDFT; in that order.</p> <p>The required -user and -password suboptions specify a user name and password for connecting to DB2.</p>
-close	<p>-close <i>close_command</i></p> <p>Specify a command to be parsed and executed by DB2 on all processing nodes accessed by the operator after WebSphere DataStage finishes reading the DB2 table and before it disconnects from DB2.</p> <p>See "Specifying Open and Close Commands" for more details.</p>
-db_cs	<p>-db_cs <i>character_set</i></p> <p>Specify the character set to map between DB2 char and Varchar values and WebSphere DataStage ustring schema types and to map SQL statements for output to DB2. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data.</p> <p>For information on national language support, reference this IBM ICU site:</p> <p>http://oss.software.ibm.com/icu/charset</p>

Table 163. db2read Options (continued)

Option	Use
-dbname	<p>-dbname <i>database_name</i></p> <p>Specifies the name of the DB2 database to access.</p> <p>By default, the operator uses the setting of the environment variable APT_DBNAME, if defined, and DB2DBDFT otherwise. Specifying -dbname overrides APT_DBNAME and DB2DBDFT.</p> <p>See the DataStage 7.5 Installation and Administration Manual for information on creating the DB2 configuration.</p>
-open	<p>-open <i>open_command</i></p> <p>Specifies any command to be parsed and executed by DB2. WebSphere DataStage causes DB2 to execute this command on all processing nodes to be accessed before opening the table. See "Specifying Open and Close Commands" for more details.</p>
-query	<p>-query <i>sql_query</i> [-part <i>table_name</i>]</p> <p>Specifies an SQL query to read one or more tables. The query specifies the tables and the processing that you want to perform on the tables as they are read into WebSphere DataStage. This statement can contain joins, views, database links, synonyms, and so on.</p> <p>The -part suboption specifies execution of the query in parallel on the processing nodes containing a partition of <i>table_name</i>. If you do not specify -part, the operator executes the query sequentially on a single node.</p> <p>If the name in the -user suboption of the -client_instance option differs from your UNIX user name, use an owner-qualified name when specifying the table. The syntax is:</p> <p style="padding-left: 2em;"><i>owner_name.table_name</i></p> <p>The default open command for a parallel query uses the table specified by the -part option. Either the -table or -query option must be specified.</p>
-server	<p>-server <i>server_name</i></p> <p>Optionally specifies the DB2 instance name for the table or query read by the operator.</p> <p>By default, WebSphere DataStage uses the setting of the DB2INSTANCE environment variable. Specifying -server overrides DB2INSTANCE.</p>

Table 163. db2read Options (continued)

Option	Use
-table	<p>-table <i>table_name</i> [-filter <i>filter</i>] [-selectlist <i>list</i>]</p> <p>-table specifies the name of the DB2 table. The table must exist and you must have SELECT privileges on the table. If your DB2 user name does not correspond to the name of the owner of <i>table_name</i>, you can prefix <i>table_name</i> with a table owner in the form:</p> <p><i>table_owner.table_name</i></p> <p>If you do not include the table owner, the operator uses the following statement to determine your user name:</p> <pre>select user from sysibm.sysables;</pre> <p>-filter specifies a conjunction to the WHERE clause of the SELECT statement to specify the rows of the table to include or exclude from reading into WebSphere DataStage. If you do not supply a WHERE clause, all rows are read. -filter does not support the use of bind variables.</p> <p>-selectlist specifies an SQL select list that determines which columns are read. Select list items must have DB2 data types that map to WebSphere DataStage data types. You must specify fields in the same order as they are defined in the record schema of the input table. If you do not specify a list, all columns are read. The -selectlist option does not support the use of bind variables.</p> <p>Either the -table or -query option must be specified.</p>
-use_strings	<p>-use_strings</p> <p>Directs WebSphere DataStage to import DB2 char and Varchar values to WebSphere DataStage as WebSphere DataStage strings without converting them from their ASCII or binary form. This option overrides the db_cs option which converts DB2 char and Varchar values as ustrings using a specified character set.</p>

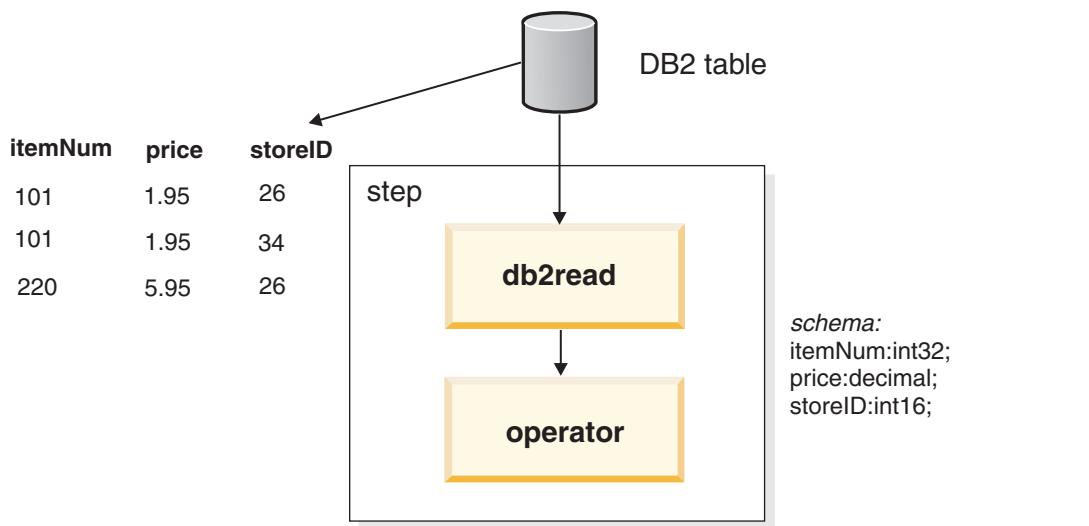
Db2read example 1: reading a DB2 table with the table option

The following figure shows table_1, a DB2 table used as an input to the db2read WebSphere DataStage operator.

In this example:

- The read operation is performed in parallel, with one instance of the operator for each partition of the table, because the -table option is specified.
- The entire table is read because neither a -filter nor a -selectlist suboption is specified.

The db2read operator writes the table to a WebSphere DataStage data set. The schema of the data set is also shown in this figure. The data set is the input of the next operator.



The DB2 table contains three columns. Each DB2 column name becomes the name of a WebSphere DataStage field. The operator converts the data type of each column to its corresponding WebSphere DataStage data type, as listed in the this table:

Column/Field Name	DB2 Data Type	Converted to WebSphere DataStage Data Type
itemNum	INTEGER	int32
price	DECIMAL	decimal
storeID	SMALLINT	int16

Here is the osh syntax for this example:

```
$ osh "db2read -table table_1 ... "
```

Example 2: reading a DB2 table sequentially with the -query option

As in Example 1 above, this query reads a DB2 table named table_1. In this example, the query explicitly specifies three columns to read from the table, itemNum, price, and storeID. The read operation is executed sequentially. This is the default execution mode of db2read when the -query option is chosen.

Here is the osh syntax for this example:

```
$ osh "db2read -query 'select itemNum, price, storeID from table_1' ..."
```

Example 3: reading a table in parallel with the -query option

The following query reads every row and column of a DB2 table named table_1. The -part option and table_1 argument direct WebSphere DataStage to create an instance of the operator on every processing node containing a partition of this table.

Here is the osh syntax for this example:

```
$ osh "db2read
      -query 'select * from table_1 where
      node'
      -part table_1 ... "
           nodenumber(itemNum)=current
```

where itemNum is the name of the first column of the table or select list. This predicate is automatically generated by WebSphere DataStage and is used to create the correct number of DB2 read operators for the number of partitions in the DB2 table.

The db2write and db2load operators

The WebSphere DataStage operators that write to DB2 are db2write and db2load. They function identically with these exceptions:

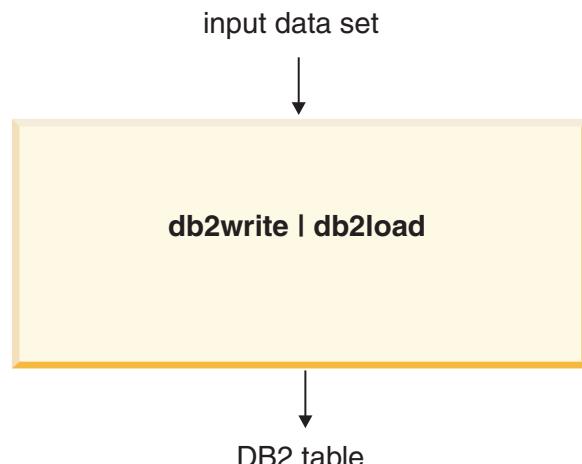
- The db2load operator takes advantage of the fast DB2 loader technology for writing data to the database.
- The db2load operator accepts four options that db2write does not: -ascii, -cleanup, -msgfile, and -nonrecoverable.
- The db2load operator requires that WebSphere DataStage users have DBADM privilege on the DB2 database written by the operator.

The operators set up a connection to DB2 and insert records into a table. These operators take a single input data set. The write mode of the operators determines how the records of a data set are inserted into the table.

Both operators are documented as the write operators. The particular aspects of db2load are indicated in "Syntax and Options" and "db2load Special Characteristics".

Also see the section on the db2upsert operator which inserts and updates DB2 table records with data contained in a WebSphere DataStage data set. It is described in "The db2upsert Operator".

Data flow diagram



db2write and db2load: properties

Table 164. db2write and db2load Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	0
Input interface schema	derived from the input data set; must match that of the table, if it already exists.
Output interface schema	none

Table 164. db2write and db2load Operator Properties (continued)

Property	Value
Transfer behavior	none
Execution mode	parallel (default) or sequential
Partitioning method	db2part See "The db2part Operator". You can override this partitioning method; however, a partitioning method of entire is not allowed.
Collection method	any
Preserve-partitioning flag in output data set	not applicable
Composite operator	no

Actions of the write operators

Here are the chief characteristics of the WebSphere DataStage operators that write WebSphere DataStage data sets to DB2 tables:

- They translate the WebSphere DataStage data set record by record to DB2 table rows by means of an SQL INSERT statement, as described in "How WebSphere DataStage Writes the Table: the Default SQL INSERT Statement".
- The fields of the WebSphere DataStage record correspond to the columns of the DB2 table. Fields might be only of certain data types, as described in "Field Conventions in Write Operations to DB2".
- Translation includes the conversion of WebSphere DataStage data types to DB2 data types, as listed in "Data Type Conversion".
- The write operators append records to an existing table, unless you set another mode of writing. "Write Modes" discusses these modes.
- When you write to an existing table, the schema of the table determines the operator's input interface and the data set schema must be compatible with the table's schema. See "Matched and Unmatched Fields" and "Syntax and Options".
- Each instance of a parallel write operator running on a processing node writes its partition of the data set to the DB2 table. A failure by one instance aborts the write operation.

Operators optionally specify DB2 commands to be parsed and executed on all processing nodes before the write operation runs using the -open option or after it finishes using the -close option.

How WebSphere DataStage writes the table: the default SQL INSERT statement

You cannot explicitly define an SQL statement for the write operation. Instead, WebSphere DataStage generates an SQL INSERT statement that writes to the table. However, optional parameters of the write operators allow you to narrow the operation.

When you specify the DB2 table name to a write operator, WebSphere DataStage creates this INSERT statement:

```
insert into table_name [(selectlist)] values (?, ?, ?, ...);
```

where:

- *table_name* specifies the name of a DB2 table. By default, if *table_name* does not exist, the step containing the write operator terminates, unless you use the -mode create option.

To set the table name, you specify the -table option.

- *selectlist* optionally specifies a clause of the INSERT statement to determine the fields of the data set to be written.
If you do not specify *selectlist*, WebSphere DataStage writes all fields in the data set.
You specify the write operator's -selectlist suboption of -table to set the select list.
- *???, ...* contains one input parameter for each column written by WebSphere DataStage. By default, this clause specifies that all fields in the input data set are written to DB2.
However, when you specify a *selectlist*, the default is modified to contain only the columns defined by the *selectlist*.

Field conventions in write operations to DB2

The record schema of the input WebSphere DataStage data set defines the field name and data type of each field. When the write operators write this data set to DB2, they do not modify the field names but translate the WebSphere DataStage data types to DB2 data types, with these limitations:

- DB2 column names might have a length limit. If an WebSphere DataStage field name is longer than that limit, which varies by DB2 release, do one of the following:
 - Specify -truncate to the write operator to truncate WebSphere DataStage field names to 18 characters, or specify -truncate and -truncationLength *n*, to set the field name to some other length.
 - Invoke the modify operator to modify the field name. See "Modify Operator".
- The order of the fields of the WebSphere DataStage record and that of the columns of the DB2 rows can differ but are nonetheless written successfully.
- The size of records is limited to 32 KB, but a DB2 table row can be 4005 bytes. If you try to write a WebSphere DataStage record of greater length, DB2 returns an error and terminates the step.
- The maximum number of columns allowed in a UDB table is 500, and the maximum allowed in a view is 5000.
- A data set written to DB2 might not contain fields of certain types. If it does, an error occurs and the corresponding step terminates. However WebSphere DataStage offers operators that modify certain data types to types DB2 accepts, as in the next table:

Table 165. Operators that modify WebSphere DataStage Types for DB2

Incompatible Type	Operator That Changes It
Raw	field_import; see "The field_import Operator"
Strings, fixed or variable length, longer than 4000 bytes	none
Subrecord	promotesubrec; see "The promotesubrec Operator".
Tagged aggregate	tagbatch; see "The tagbatch Operator".
Unsigned integer of any size	modify; see "Modify Operator"

Data type conversion

The db2write and db2load operators convert WebSphere DataStage data types to DB2 data types, as listed in the next table:

Table 166. db2write and db2load Type Conversions

WebSphere DataStage Data Type	DB2 Data Type
date	DATE
decimal[p,s] <i>p</i> is decimal's precision and <i>s</i> is its scale	DECIMAL[p,s]
int8	SMALLINT
int16	SMALLINT

Table 166. db2write and db2load Type Conversions (continued)

WebSphere DataStage Data Type	DB2 Data Type
int32	INTEGER
sfloat	FLOAT
dfloat	FLOAT
fixed-length string in the form string[n] and ustring[n]; length <= 254 bytes	CHAR(n) where n is the string length
fixed-length string in the form string[n] and ustring[n]; 255 <= length <= 4000 bytes	VARCHAR(n) where n is the string length
variable-length string, in the form string[max=n] and ustring[max=n]; maximum length <= 4000 bytes	VARCHAR(n) where n is the maximum string length
variable-length string in the form string and ustring	VARCHAR(32)*
string and ustring, 4000 bytes < length	Not supported
time	TIME
timestamp	TIMESTAMP

* The default length of VARCHAR is 32 bytes. That is, 32 bytes are allocated for each variable-length string field in the input data set. If an input variable-length string field is longer than 32 bytes, the operator issues a warning. You can use the -stringlength option to modify the default length.

WebSphere DataStage data types not listed in this table generate an error and terminate the step. Use the modify operator to perform type conversions. See "Modify Operator".

Any column in the DB2 table corresponding to a nullable data set field will be nullable.

WebSphere DataStage and DB2 integer and floating-point data types have the same range and precision. You need not worry about numeric overflow or underflow.

Write modes

The write mode of the operator determines how the records of the data set are inserted into the destination table. The write mode can have one of the following values:

- append: The table must exist and the record schema of the data set must be compatible with the table. The write operator appends new rows to the table. This is the default mode. The schema of the existing table determines the input interface of the operator. See "Example 1: Appending Data to an Existing DB2 Table".
- create: The operator creates a new table. If a table exists of the same name as the one you want to create, the step that contains the operator terminates with an error. You must specify either create mode or replace mode if the table does not exist. The schema of the new table is determined by the schema of the WebSphere DataStage data set.
- By default, WebSphere DataStage creates the table on all processing nodes in the default table space and uses the first column in the table, corresponding to the first field in the input data set, as the partitioning key. You can override these options for partitioning keys and table space by means of the -dboptions option.
- replace: The operator drops the existing table and creates a new one in its place. If a table exists of the same name as the one you want to create, it is overwritten. DB2 uses the default partitioning method for the new table. The schema of the new table is determined by the schema of the WebSphere DataStage data set.

- truncate: The operator retains the table attributes but discards existing records and appends new ones. The schema of the existing table determines the input interface of the operator. See "Example 2: Writing Data to a DB2 Table in truncate Mode" .

Note: If a previous write operation fails, you can retry your application specifying a write mode of replace to delete any information in the output table that might have been written by the previous attempt to run your program.

Each mode requires specific user privileges, as listed in this table:

Table 167. Privileges Required for db2write and db2load Write Modes

Write Mode	Required Privileges
append	INSERT on existing table
create	TABLE CREATE
replace	INSERT and TABLE CREATE on existing table
truncate	INSERT on existing table

Matched and unmatched fields

The schema of the DB2 table determines the operator's interface schema. Once the operator determines this, it applies the following rules to determine which data set fields are written to the table:

1. Fields of the input data set are matched by name with fields in the input interface schema. WebSphere DataStage performs default data type conversions to match the input data set fields with the input interface schema. You can also use the modify operator to perform explicit data type conversions. See "Modify Operator" .
2. If the input data set contains fields that do not have matching components in the table, the operator causes an error and terminates the step.

This rule means that WebSphere DataStage does not add new columns to an existing table if the data set contains fields that are not defined in the table. Note that you can use either the -drop option or the modify operator with the db2write operator to drop extra fields from the data set. See "Example 3: Handling Unmatched WebSphere DataStage Fields in a DB2 Write Operation" and "Modify Operator"

3. Columns in the DB2 table that do not have corresponding fields in the input data set are set to their default value, if one is specified in the DB2 table. If no default value is defined for the DB2 column and it supports nulls, it is set to null. Otherwise, WebSphere DataStage issues an error and terminates the step. See "Example 4: Writing to a DB2 Table Containing an Unmatched Column" .
4. WebSphere DataStage data sets support nullable fields. If you write a data set to an existing table and a field contains a null, the DB2 column must also support nulls. If not, WebSphere DataStage issues an error message and terminates the step.

You can use the modify operator to convert a null in an input data set field to a different value in order to avoid the error. See "Modify Operator" for more information on handling nulls.

Db2write and db2load: syntax and options

Following is the syntax of the db2write and db2load operators. Exactly one occurrence of the -table option is required. All other options are optional. Most options apply to both operators; the exceptions are noted.

```
db2write | db2load
  -table table_name [-selectlist selectlist]
    [-ascii] (db2load only)
    [-cleanup] (db2load only)
```

```

[-client_instance client_instance_name
 -client_dbname database
 -user user_name
 -password password]
[-close close-command]
[-cpu integer] (db2load only)
[-rowCommitInterval integer] (db2write only)
[-db_cs character_set]
[-dbname database_name]
[-dboptions
  '{[ = table_space,]
   [key = field0, . . . key = fieldN]}']
[-drop]
[-exceptionTable table_name] (db2load only)
[-mode create | replace | append | truncate]
[-msgfile msgFile] (db2load only)
[-nonrecoverable] (db2load only)
[-omitPart]
[-open open_command]
[-server server_name]
[-statistics stats_none | stats_exttable_only | stats_extindex_only | stats_exttable_index |
stats_index | stats_table | stats_extindex_table | stats_all | stats_both] (db2load only)
[-stringlength string_length]
[-truncate]
[-truncationLength n]
```

Table 168. db2write and db2load Options

Option	Use
-ascii	<p>-ascii</p> <p>db2load only. Specify this option to configure DB2 to use the ASCII-delimited format for loading binary numeric data instead of the default ASCII-fixed format.</p> <p>This option can be useful when you have variable-length fields, because the database does not have to allocate the maximum amount of storage for each variable-length field. However, all numeric fields are converted to an ASCII format by DB2, which is a CPU-intensive operation. See the DB2 reference manuals for more information.</p>

Table 168. db2write and db2load Options (continued)

Option	Use
-cleanup	<p>-cleanup</p> <p>db2load only. Specify this option to deal with operator failures during execution that leave the loading tablespace that it was loading in an inaccessible state. For example, if the following osh command was killed in the middle of execution:</p> <pre>db2load -table upcdata -dbname my_db</pre> <p>The tablespace in which this table resides will probably be left in a quiesced exclusive or load pending state. In order to reset the state to normal, run the above command specifying -cleanup, as follows:</p> <pre>db2load -table upcdata -dbname my_db -cleanup</pre> <p>The cleanup procedure neither inserts data into the table nor deletes data from it. You must delete rows that were inserted by the failed execution either through the DB2 command-level interpreter or by running the operator subsequently using the replace or truncate modes.</p>
-client_instance	<p>-client_instance <i>client_instance_name</i> [-client_dbname <i>database</i>] -user <i>user_name</i> -password <i>password</i></p> <p>Specifies the client DB2 instance name. This option is required for a remote connection.</p> <p>The -client_dbname suboption specifies the client database alias name for the remote server database. If you do not specify this option, WebSphere DataStage uses the value of the -dbname option, or the value of the APT_DBNAME environment variable, or DB2DBDFT; in that order.</p> <p>The required -user and -password suboptions specify a user name and password for connecting to DB2.</p>
-close	<p>-close <i>close_command</i></p> <p>Specifies any command to be parsed and executed by the DB2 database on all processing nodes after WebSphere DataStage finishes processing the DB2 table.</p>

Table 168. db2write and db2load Options (continued)

Option	Use
-cpu	<p>[cpu <i>integer</i>] [-anyorder]</p> <p>Specifies the number of processes to initiate on every node. Its syntax is:</p> <p style="padding-left: 2em;">[-cpu <i>integer</i>] [-anyorder]</p> <p>The default value for the -cpu option is 1. Specifying a 0 value allows db2load to generate an algorithm that determines the optimal number based on the number of CPUs available at runtime. Note that the resulting value might not be optimal because DB2 does not take into account the WebSphere DataStage workload.</p> <p>The -anyorder suboption allows the order of loading to be arbitrary for every node, potentially leading to a performance gain. The -anyorder option is ignored if the value of the -cpu option is 1.</p>
-rowCommitInterval	<p>-rowCommitInterval <i>integer</i></p> <p>db2write only. Specifies the size of a commit segment. Specify an integer that is 1 or larger. The specified number must be a multiple of the input array size. The default size is 2000.</p> <p>You can also use the APT_RDBMS_COMMIT_ROWS environment to specify the size of a commit.</p>
-db_cs	<p>-db_cs <i>character_set</i></p> <p>Specify the character set to map between DB2 char and Varchar values and WebSphere DataStage ustring schema types and to map SQL statements for output to DB2. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data.</p> <p>For information on national language support, reference this IBM ICU site:</p> <p>http://oss.software.ibm.com/icu/charset</p>
-dbname	<p>-dbname <i>database_name</i></p> <p>Specifies the name of the DB2 database to access. By default, the operator uses the setting of the environment variable APT_DBNAME, if defined, and DB2DBDFT otherwise. Specifying -dbname overrides APT_DBNAME and DB2DBDFT.</p>

Table 168. db2write and db2load Options (continued)

Option	Use
-dboptions	<p>-dboptions { [-tablespace = <i>table_space</i>,] [-key = <i>field</i> ...]}</p> <p>Specifies an optional table space or partitioning key to be used by DB2 to create the table. You can specify this option only when you perform the write operation in either create or replace mode.</p> <p>The partitioning key must be the external column name. If the external name contains # or \$ characters, the entire column name should be surrounded by single backslashed quotes. For example:</p> <pre>-dboptions '{key=\`B##B\\$\'}'</pre> <p>By default, WebSphere DataStage creates the table on all processing nodes in the default table space and uses the first column in the table, corresponding to the first field in the input data set, as the partitioning key.</p> <p>You specify arguments to -dboptions as a string enclosed in braces. This string can contain a single -tablespace argument, a single -nodegroup argument, and multiple -key arguments, where:</p> <ul style="list-style-type: none"> -tablespace defines the DB2 table space used to store the table. -nodegroup specifies the name of the node pool (as defined in the DataStage configuration file) of nodes on which DB2 is installed. -key specifies a partitioning key for the table.
-drop	<p>-drop</p> <p>Causes the operator to silently drop all input fields that do not correspond to fields in an existing DB2 table. By default, the operator reports an error and terminates the step if an input field does not have a matching column in the destination table.</p>
-exceptionTable	<p>[-exceptionTable <i>table_name</i>]</p> <p>db2write only. Specifies a table for inserting records which violate load-table constraints. The exceptions table should be consistent with the FOR EXCEPTION option of the db2load utility. Refer to the DB2 documentation for instructions on how to create this table.</p> <p>The -exceptionTable option cannot be used with the create or replace modes because WebSphere DataStage cannot recreate the table with all its applicable data constraints.</p> <p>When the mode is truncate, the tables for the -table and -exceptionTable options are truncated.</p>

Table 168. db2write and db2load Options (continued)

Option	Use
-mode	<p>-mode create replace append truncate Specifies the write mode of the operator. append (default): New records are appended to an existing table. create: Create a new table. WebSphere DataStage reports an error and terminates the step if the DB2 table already exists. You must specify this mode if the DB2 table does not exist. truncate: The existing table attributes (including schema) and the DB2 partitioning keys are retained, but any existing records are discarded. New records are then appended to the table. replace: The existing table is first dropped and an entirely new table is created in its place. DB2 uses the default partitioning method for the new table.</p>
-msgfile	<p>-msgfile <i>msgFile</i> db2load only. Specifies the file where the DB2 loader writes diagnostic messages. The <i>msgFile</i> can be an absolute path name or a relative path name. Regardless of the type of path name, the database instance must have read/write privilege to the file. By default, each processing node writes its diagnostic information to a separate file named APT_DB2_LOADMSG_<nodenumber> where <nodenumber> is the DB2 node number of the processing node. Specifying -msgFile configures DB2 to write the diagnostic information to the file <i>msgFile_nodenumber</i>. If you specify a relative path name, WebSphere DataStage attempts to write the files to the following directories, in this order: 1. The file system specified by the default resource scratchdisk, as defined in the WebSphere DataStage configuration file. 2. The directory specified by the tmpdir parameter to the WebSphere DataStage server. 3. The directory /tmp.</p>
-nonrecoverable	<p>-nonrecoverable db2load only. Specifies that your load transaction is marked as nonrecoverable. It will not be possible to recover your transaction with a subsequent roll forward action. The roll forward utility skips the transaction, and marks the table into which data was being loaded as "invalid". The utility also ignores any subsequent transactions against the table. After a roll forward is completed, the table can only be dropped. Table spaces are not put in a backup pending state following the load operation, and a copy of the loaded data is not made during the load operation.</p>

Table 168. db2write and db2load Options (continued)

Option	Use
-open	<p>-open <i>open_command</i></p> <p>Specifies any command to be parsed and executed by the DB2 database on all processing nodes. DB2 runs this command before opening the table.</p>
-omitPart	<p>[-omitPart]</p> <p>Specifies that that db2load and db2write should not automatically insert a partitioner. By default, the db2partitioner is inserted.</p>
-server	<p>-server <i>server_name</i></p> <p>Optionally specifies the DB2 instance name for the table. By default, WebSphere DataStage uses the setting of the DB2INSTANCE environment variable. Specifying -server overrides DB2INSTANCE.</p>
-statistics	<p>[<i>-statistics stats_none stats_exttable_only stats_extindex_only stats_exttable_index stats_index stats_table stats_extindex_table stats_all stats_both</i>]</p> <p>db2load only. Specifies which statistics should be generated upon load completion.</p>
-stringlength	<p>-stringlength <i>string_length</i></p> <p>Sets the default string length of variable-length strings written to a DB2 table. If you do not specify a length, WebSphere DataStage uses a default size of 32 bytes. Variable-length strings longer than the set length cause an error. The maximum length you can set is 4000 bytes.</p> <p>Note that the operator always allocates <i>string_length</i> bytes for a variable-length string. In this case, setting <i>string_length</i> to 4000 allocates 4000 bytes for every string. Therefore, you should set <i>string_length</i> to the expected maximum length of your largest string and no larger.</p>

Table 168. db2write and db2load Options (continued)

Option	Use
-table	<p>-table <i>table_name</i> [-selectlist <i>list</i>]</p> <p>Specifies the name of the DB2 table.</p> <p>If the output mode is create, the table must not exist and you must have TABLE CREATE privileges. By default, WebSphere DataStage creates the table without setting partitioning keys. By default, DB2 uses the first column as the partitioning key. You can use -dboptions to override this default operation.</p> <p>If the output mode is append or truncate, the table must exist and you must have INSERT privileges.</p> <p>In replace mode, the table might already exist. In this case, the table is dropped and the table is created in create mode.</p> <p>The table name might not contain a view name.</p> <p>If your DB2 user name does not correspond to the owner of the table, you can optionally prefix <i>table_name</i> with that of a table owner in the form <i>tableOwner.tableName</i>. The <i>tableOwner</i> string must be 8 characters or less. If you do not specify <i>tableOwner</i>, WebSphere DataStage uses the following statement to determine your user name:</p> <pre>select user from sysibm.systables;</pre> <p>-selectlist optionally specifies an SQL select list used to determine which fields are written. The list does not support the use of bind variables.</p>
-truncate	<p>-truncate</p> <p>Select this option to configure the operator to truncate WebSphere DataStage field names to 18 characters. To specify a length other than 18, use the -truncationLength option along with this one. See "Field Conventions in Write Operations to DB2" for further information.</p>
-truncationLength	<p>-truncationLength <i>n</i></p> <p>Use this option with the -truncate option to specify a truncation length other than 18 for DataStage field names too long to be DB2 column names.</p>

db2load special characteristics

During the table load operation, db2load keeps an exclusive lock on the entire tablespace into which it loads data and no other tables in that tablespace can be accessed. The operator issues the QUIESCE TABLESPACES command to get and retain the lock.

The DB2 load operator performs a non-recoverable load. That is, if the load operation is terminated before it is completed, the contents of the table are unusable and the tablespace is left in a load pending state. You must run db2load in truncate mode to clear the load pending state.

The db2load operator requires that WebSphere DataStage users, under their WebSphere DataStage login name, have DBADM privilege on the DB2 database written by the operator.

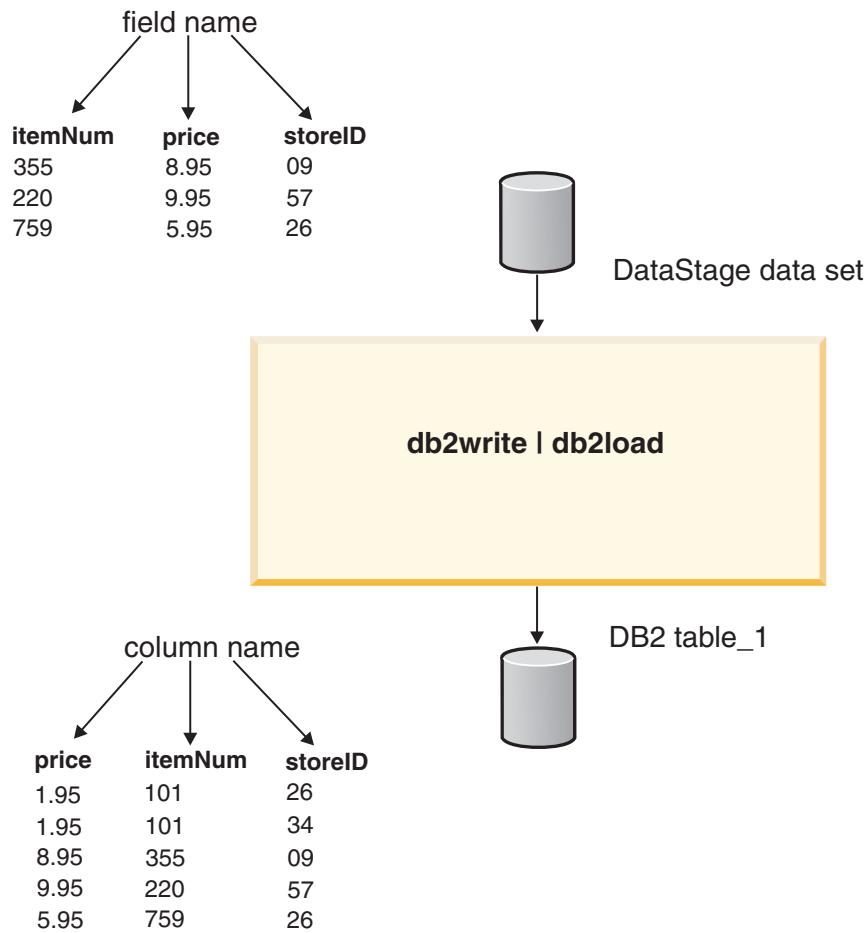
Db2write example 1: Appending Data to an Existing DB2 Table

In this example, a DB2 write operator appends the contents of a WebSphere DataStage data set to an existing DB2 table. This is the default action of DB2 write operators.

The record schema of the WebSphere DataStage data set and the row schema of the DB2 table correspond to one another, and field and column names are identical (though they are not in the same order). Here are the input WebSphere DataStage record schema and output DB2 row schema:

Input WebSphere DataStage Record	Output DB2 Table Schema
itemNum int32	price DECIMAL[3,2],
price:decimal[3,2]	itemNum INTEGER,
storeID:int16;	storeID SMALLINT,

The following figure shows the data flow and field and column names for this example. Columns that the operator appends to the DB2 table are shown in boldface type. Note that the order of the fields of the WebSphere DataStage record and that of the columns of the DB2 rows are not the same but are nonetheless written successfully.



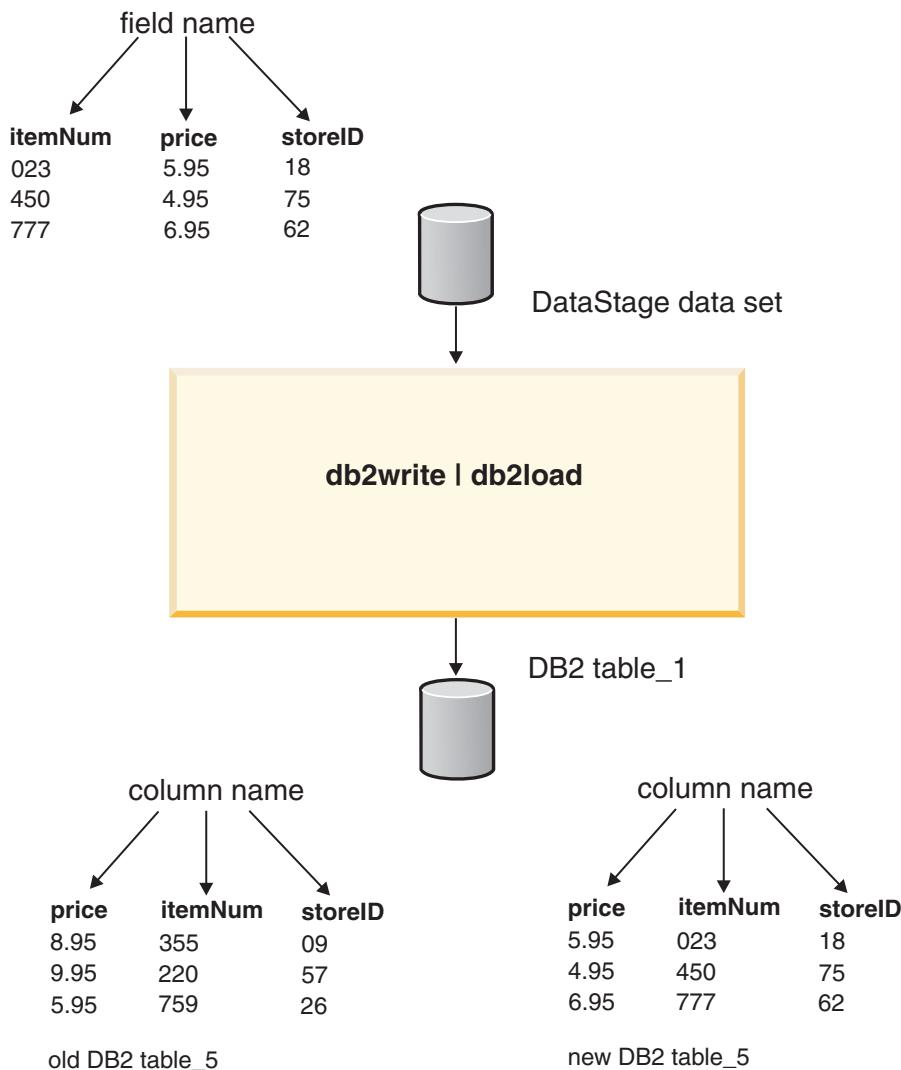
Here is the osh command for this example:

```
$ osh "... db2_write_operator -table table_1
          -server db2Server
          -dbname my_db ..."
```

Example 2: writing data to a DB2 table in truncate mode

In this example, the DB2 write operators write entirely new data to table_5 whose schema and column names it retains. It does so by specifying a write mode of truncate. The operator deletes old data and writes new data under the old column names and data types.

As in "Example 1: Appending Data to an Existing DB2 Table" , the record schema of the WebSphere DataStage data set and the row schema of the DB2 table correspond to one another, and field and column names are identical. The attributes of old and new DB2 tables are also identical.



Here is the osh command for this example:

```
$ osh "...db2_write_operator -table table_5
          -server db2Server
          -dbname my_db -mode truncate ..."
```

Example 3: handling unmatched WebSphere DataStage fields in a DB2 write operation

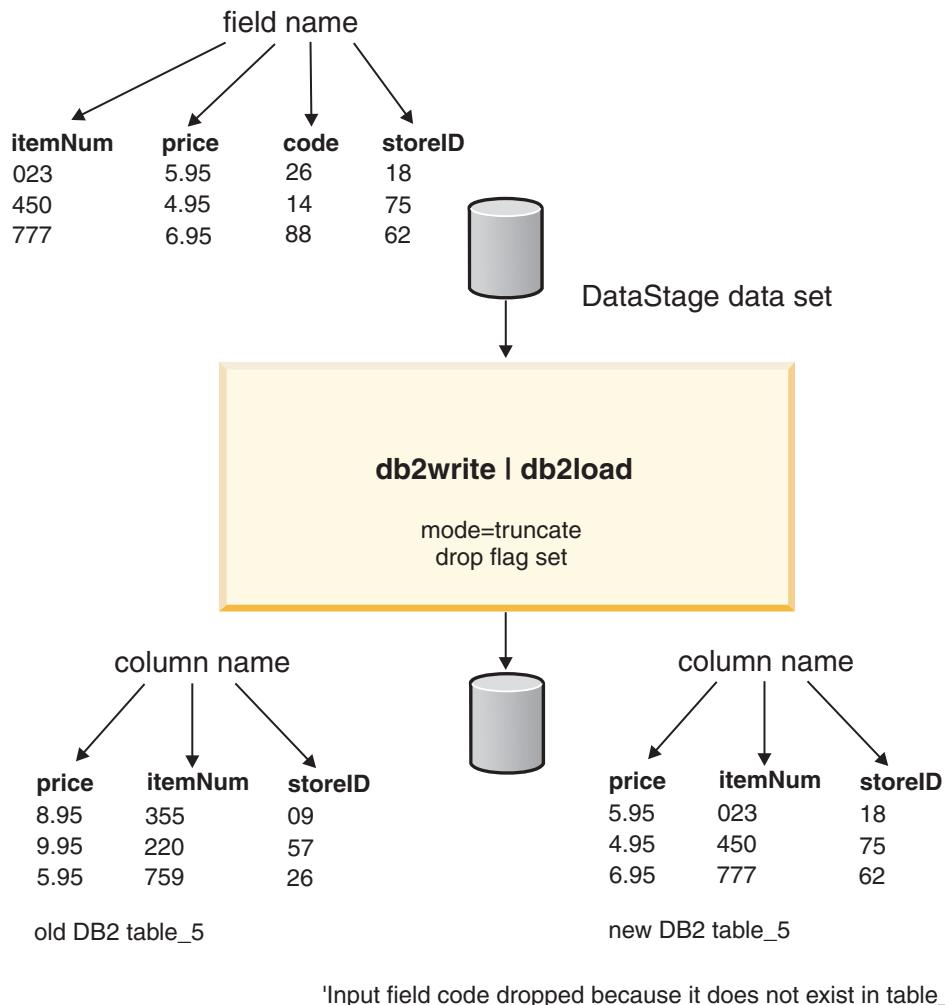
In this example, the records of the WebSphere DataStage data set contain a field that has no equivalent column in the rows of the DB2 table. If the operator attempts to write a data set with such an unmatched field, the corresponding step returns an error and terminate. The -drop option of the operator removes the unmatched field from processing and writes a warning to the message stream:

Input field *field_name* dropped because it does not exist in table *table_name*

where:

- *field_name* is the name of the WebSphere DataStage field that is dropped from the operation.
- *table_name* is the name of the DB2 table to which data is being written.

In the example, which operates in truncate mode with the -drop option chosen, the field named code is dropped and a message to that effect is displayed.



Here is the osh code for this example:

```
$ osh " ...db2_write_operator -table table_5  
-server db2server  
-dbname my_db  
-mode truncate -drop ..."
```

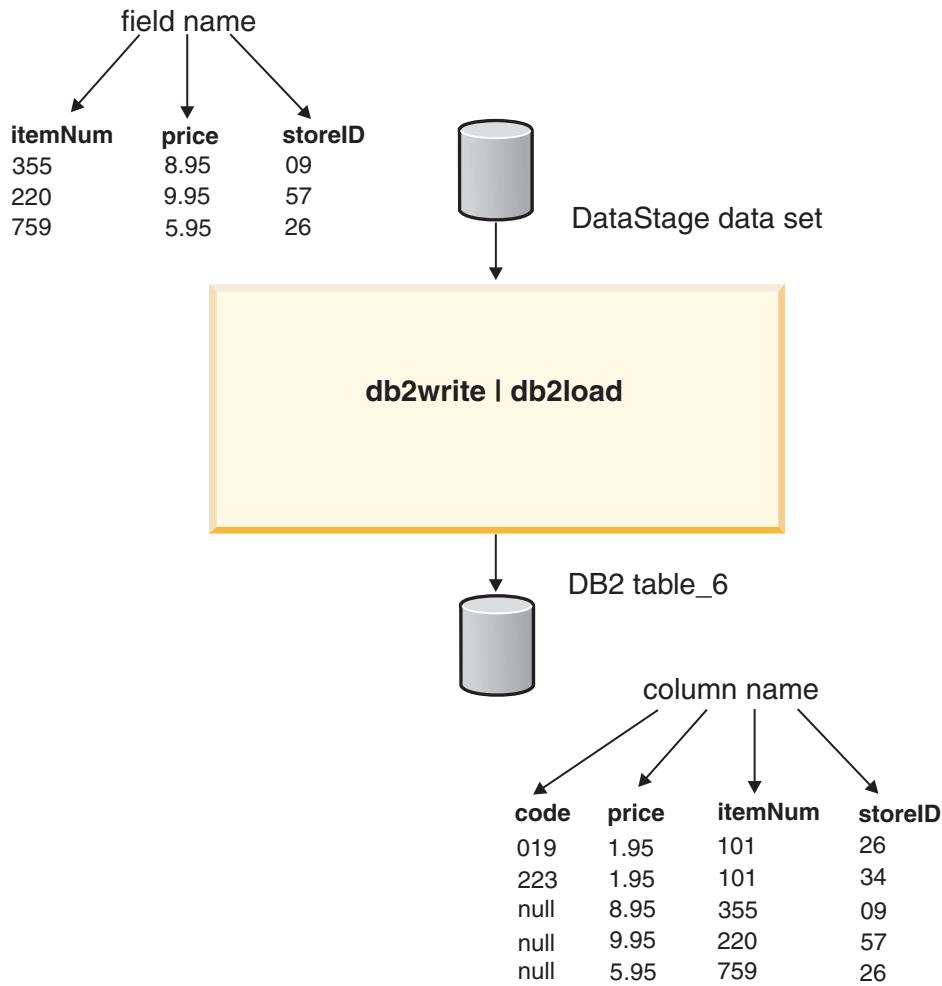
Example 4: writing to a DB2 table containing an unmatched column

In this example, the DB2 column does not have a corresponding field in the WebSphere DataStage data set, and the operator writes the DB2 default to the unmatched column. It does so without user intervention. The operator functions in its default write mode of append.

This table lists the WebSphere DataStage records and DB2 row schemas:

Input WebSphere DataStage Record	Output DB2 Table Row
	code SMALLINT,
itemNum:int32;	price DECIMAL[3,2],
price:decimal[3,2];	itemNum INTEGER,
storeID:int16;	storeID SMALLINT,

The next figure shows the write operator writing to the DB2 table. As in the previous example, the order of the columns is not necessarily the same as that of the fields of the data set.



Here is the osh code for this example:

```
$ osh "...db2_write_operator -dbname my_db  
-server db2server  
-table table_6 ..."
```

The db2upsert operator

The db2upsert operator inserts, updates, and deletes DB2 table records with data contained in a DB2 data set. You provide the insert statements using the -insert option, and provide the delete and update SQL statements using the -update option.

An example -update delete statement is:

```
-update 'delete from tablename where A = ORCHESTRATE.A'
```

The db2upsert operator takes advantage of the DB2 CLI (Call Level Interface) system to optimize performance. It builds an array of records and then executes them with a single statement.

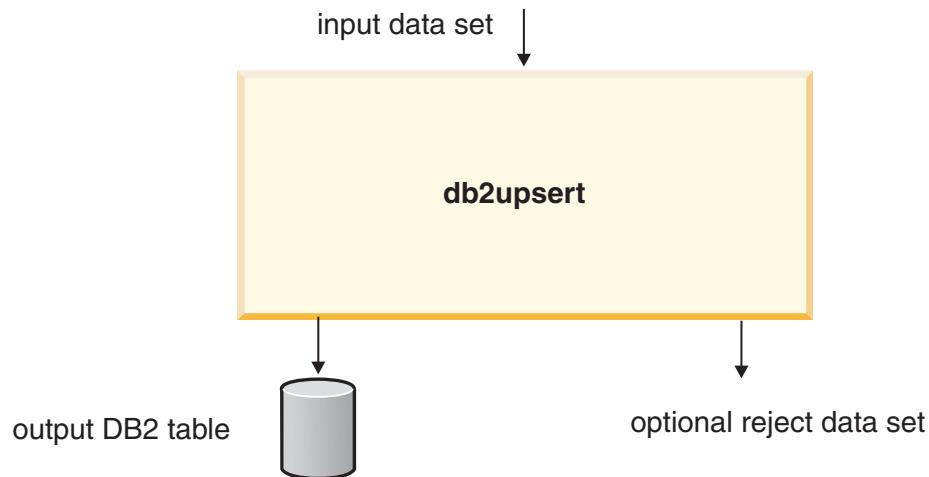
This operator receives a single data set as input and writes its output to a DB2 table. You can request an optional output data set that contains the records that fail to be inserted or updated.

Partitioning for db2upsert

To ensure that insert and update results are correct, the db2upsert operator automatically inserts the db2part partitioner which partitions the input data set the same way the table is partitioned using the partitioning key of the table. You can prevent this automatic insertion by explicitly inserting another partitioner, but the results might not be correct.

In order for the db2upsert to partition correctly, the partitioning key of the table must be a WebSphere DataStage bind variable in the WHERE clause of the update statement. If this is not possible, db2upsert should be run sequentially.

Data flow diagram



db2upsert: properties

Table 169. db2upsert Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	by default, none; 1 when you select the -reject option

Table 169. db2upsert Operator Properties (continued)

Property	Value
Input interface schema	derived from your insert and update statements
Transfer behavior	Rejected update records are transferred to an output data set when you select the -reject option. A state field is added to each record. It contains a five-letter SQL code which identifies the reason the record was rejected.
Execution mode	parallel by default, or sequential
Partitioning method	any Specifying any allows the db2part partitioner to be automatically inserted.
Collection method	any
Combinable operator	yes

Operator action

Here are the main characteristics of db2upsert:

- An -update statement is required. The -insert and -reject options are mutually exclusive with using an -update option that issues a delete statement.
- WebSphere DataStage uses the DB2 CLI (Call Level Interface) to enhance performance by executing an array of records with a single SQL statement. In most cases, the CLI system avoids repetitive statement preparation and parameter binding.
- The -insert statement is optional. If it is included, it is executed first. An insert record that does not receive a row status of SQL_PARAM_SUCCESS or SQL_PARAM_SUCCESS_WITH_INFO is then used in the execution of an update statement.
- When you specify the -reject option, any update record that receives a status of SQL_PARAM_ERROR is written to your reject data set. Its syntax is:
`-reject filename`
- You use the -arraySize option to specify the size of the input array. For example:
`-arraySize 600`

The default array size is 2000.

- The db2upsert operator commits its input record array, and then waits until a specified number of records have been committed or a specified time interval has elapsed, whichever occurs first, before committing the next input array. You use the -timeCommitInterval and -rowCommitInterval options to control intervals between commits.

To specify between-commit intervals by time, specify a number of seconds using the -timeCommitInterval option. For example:

```
-timeCommitInterval 3
```

The default time-commit interval is 2 seconds.

To specify between-commit intervals by the number of records committed, use the -rowCommitInterval option. Specify a multiple of the input array size. The default value is 2000.

- You use the -open and -close options to define optional DB2 statements to be executed before and after the processing of the insert array.
- At the end of execution, the db2upsert operator prints the number of inserted, updated, and rejected records for each processing node used in the step.

- Columns in the DB2 table that do not have corresponding fields in the input data set are set to their default value, if one is specified in the DB2 table. If no default value is defined for the DB2 column and it supports nulls, it is set to null. Otherwise, WebSphere DataStage issues an error and terminates the step.
 - The db2upsert operator converts WebSphere DataStage values to DB2 values using the type conversions detailed in “Data type conversion” on page 647.
- The required DB2 privileges are listed in “Write modes” on page 648.

Db2upsert: syntax and options

The syntax for db2upsert is shown below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes. Exactly one occurrence of the -update option is required; all other options are optional.

```
db2upsert
  -update update_or_delete_statement
  [-arraySize n]
  [-close close_statement]
  [-client_instance client_instance_name]
  -client_dbname database
  -user user_name
  -password password
  [-db_cs character_set]
  [-dbname database_name]
  [-insert insert_statement]
  [-open open_statement]
  [-omitPart]
  [-padchar string]
  [-reject]
  [-rowCommitInterval n]
  [-server server_name]
  [-timeCommitInterval n]
```

Table 170. db2upsert Operator Options

Options	Value
-arraySize	-arraySize <i>n</i> Optionally specify the size of the insert array. The default size is 2000 records.
-close	-close <i>close_command</i> Optionally specify an SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statements are executed only once on the Conductor node.

Table 170. db2upsert Operator Options (continued)

Options	Value
-client_instance	<pre>-client_instance client_instance_name [-client_dbname database] -user user_name -password password</pre> <p>Specifies the client DB2 instance name. This option is required for a remote connection.</p> <p>The -client_dbname suboption specifies the client database alias name for the remote server database. If you do not specify this option, WebSphere DataStage uses the value of the -dbname option, or the value of the APT_DBNAME environment variable, or DB2DBDFT; in that order.</p> <p>The required -user and -password suboptions specify a user name and password for connecting to DB2.</p>
-db_cs	<pre>-db_cs character_set</pre> <p>Specify the character set to map between DB2 char and Varchar values and DataStage ustring schema types and to map SQL statements for output to DB2. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data.</p> <p>For information on national language support, reference this IBM ICU site:</p> <p>http://oss.software.ibm.com/icu/charset</p>
-dbname	<pre>-dbname database_name</pre> <p>Specifies the name of the DB2 database to access. By default, the operator uses the setting of the environment variable APT_DBNAME, if defined, and DB2DBDFT otherwise. Specifying -dbname overrides APT_DBNAME and DB2DBDFT.</p>
-insert	<pre>-insert insert_statement</pre> <p>Optionally specify the insert statement to be executed.</p> <p>This option is mutually exclusive with using an -update option that issues a delete statement.</p>
-omitPart	<pre>[-omitPart]</pre> <p>Specifies that db2upsert should not automatically insert a partitioner. By default, the db2partitioner is inserted.</p>
-open	<pre>-open open_command</pre> <p>Optionally specify an SQL statement to be executed before the insert array is processed. The statements are executed only once on the Conductor node.</p>
-padchar	<pre>-padchar string</pre> <p>Specify a string to pad string and ustring fields that are less than the length of the DB2 CHAR column. See "Using the -padchar Option" for more information on how to use this option.</p>

Table 170. db2upsert Operator Options (continued)

Options	Value
<code>-reject</code>	<code>-reject</code> If this option is set, records that fail to be updated or inserted are written to a reject data set. You must designate an output data set for this purpose. This option is mutually exclusive with using an <code>-update</code> option that issues a delete statement.
<code>-rowCommitInterval</code>	<code>-rowCommitInterval n</code> Specifies the number of records that should be committed before starting a new transaction. The specified number must be a multiple of the input array size. The default is 2000. You can also use the APT_RDBMS_COMMIT_ROWS environment to specify the size of a commit.
<code>-server</code>	<code>-server server_name</code> Specify the name of the DB2 instance name for the table name. By default, WebSphere DataStage uses the setting of the DB2INSTANCE environment variable. Specifying <code>-server</code> overrides DB2INSTANCE.
<code>-timeCommitInterval</code>	<code>-timeCommitInterval n</code> Specifies the number of seconds WebSphere DataStage should allow between committing the input array and starting a new transaction. The default time period is 2 seconds
<code>-update</code>	<code>-update update_or_delete_statement</code> Use this required option to specify the update or delete statement to be executed. An example delete statement is: <code>-update 'delete from tablename where A = ORCHESTRATE.A'</code> A delete statement cannot be issued when using the <code>-insert</code> or <code>-reject</code> option.

The db2part operator

You can use the db2part operator when you are creating a custom operator. The db2part operator replicates the DB2 partitioning method of a specific DB2 table.

WebSphere DataStage uses the partitioning method specified by a parallel operator to partition a data set for processing by that operator. If the operator is supplied by WebSphere DataStage, the operator has a predefined partitioning method in most cases. However, if you create an operator, you can choose its partitioning method. DB2 also implements partitioning. DB2 hash partitions the rows of a DB2 table based on one or more columns of each row. Each DB2 table defines the columns used as keys to hash partition it.

Using db2part can optimize operator execution. For example, if the input data set contains information that updates an existing DB2 table, you can partition the data set so that records are sent to the processing node that contains the corresponding DB2 rows. When you do, the input record and the DB2 row are local to the same processing node, and read and write operations entail no network activity.

Db2upsert: syntax and options

The db2part operator has the osh syntax below. The -table option is required; all other options are optional. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose the value in single quotes.

```
db2part
  -table table_name
  [-owner ownername]
  [-client_instance client_instance_name]
  -client_dbname database
  -user user_name
  -password password
  [-db_cs character_set]
  [-dbname database_name]
  [-server server_name]
```

Table 171. dbpart Operator Options

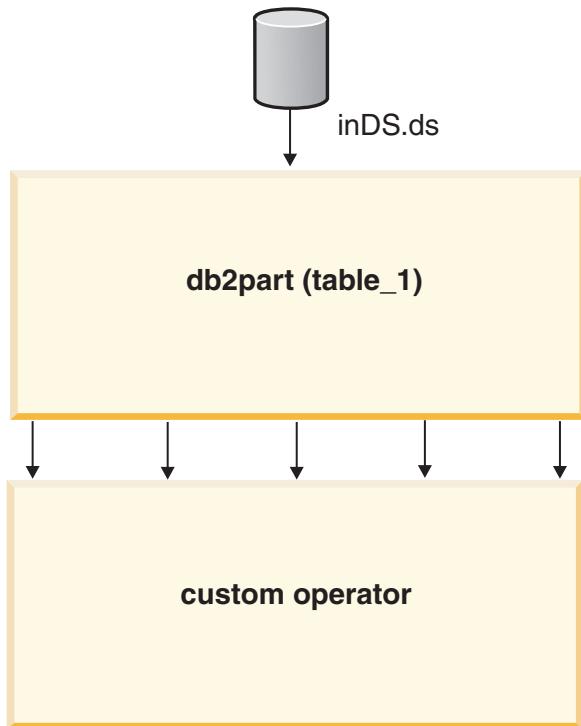
Option	Use
-table	<p>-table <i>table_name</i> [-owner <i>owner_name</i>]</p> <p>Specify the table name of the table whose partitioning method you want to replicate. Optionally supply the owner of the table.</p>
-client_instance	<p>-client_instance <i>client_instance_name</i> [-client_dbname <i>database</i>] -user <i>user_name</i> -password <i>password</i></p> <p>Specifies the client DB2 instance name. This option is required for a remote connection.</p> <p>The -client_dbname suboption specifies the client database alias name for the remote server database. If you do not specify this option, WebSphere DataStage uses the value of the -dbname option, or the value of the APT_DBNAME environment variable, or DB2DBDFT; in that order.</p> <p>The required -user and -password suboptions specify a user name and password for connecting to DB2.</p>
-db_cs	<p>-db_cs <i>character_set</i></p> <p>Specify the character set to map between DB2 char and Varchar values and WebSphere DataStage cstring schema types and to map SQL statements for output to DB2. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data.</p> <p>For information on national language support, reference this IBM ICU site:</p> <p>http://oss.software.ibm.com/icu/charset</p>

Table 171. dbpart Operator Options (continued)

Option	Use
-dbname	-dbname <i>database_name</i> Specify the name of the DB2 database to access. By default, the partitioner uses the setting of the environment variable APT_DBNAME, if defined, and DB2DBDFT otherwise. Specifying -dbname overrides APT_DBNAME and DB2DBDFT.
-server	-server <i>server_name</i> Type the name of the DB2 instance name for the table name. By default, WebSphere DataStage uses the setting of the DB2INSTANCE environment variable. Specifying -server overrides DB2INSTANCE.

Example

Following is an example using the db2part operator, which partitions the input data set, inDS.ds, according to the partitioning method of table_1 and then inputs the resulting records to a custom operator:



Here is the osh code for this example:

```
$ "osh db2part -table table_1 < inDS.ds | custom_operator ..."
```

The db2lookup operator

With the db2lookup operator, you can perform a join between one or more DB2 tables and a WebSphere DataStage data set. The resulting output data is a WebSphere DataStage data set containing WebSphere DataStage and DB2 data.

You perform this join by specifying either an SQL SELECT statement, or by specifying one or more DB2 tables and one or more key fields on which to do the lookup.

This operator is particularly useful for sparse lookups, that is, where the WebSphere DataStage data set you are matching is much smaller than the DB2 table. If you expect to match 90% of your data, using the db2read and lookup operators is probably more efficient.

Because db2lookup can do lookups against more than one DB2 table, it is useful for joining multiple DB2 tables in one query.

The -query option command corresponds to an SQL statement of this form:

```
select a,b,c from data.testtbl  
  where  
    Orchestrate.b = data.testtbl.c  
    and  
    data.testtbl.name = "Smith"
```

The operator replaces each `orchestrate.fieldname` with a field value, submits the statement containing the value to DB2, and outputs a combination of DB2 and WebSphere DataStage data.

Alternatively, you can use the -key/-table options interface to specify one or more key fields and one or more DB2 tables. The following osh options specify two keys and a single table:

```
-key a -key b -table data.testtbl
```

you get the same result as you would by specifying:

```
select * from data.testtbl  
where  
Orchestrate.a = data.testtbl.a  
and  
Orchestrate.b = data.testtbl.b
```

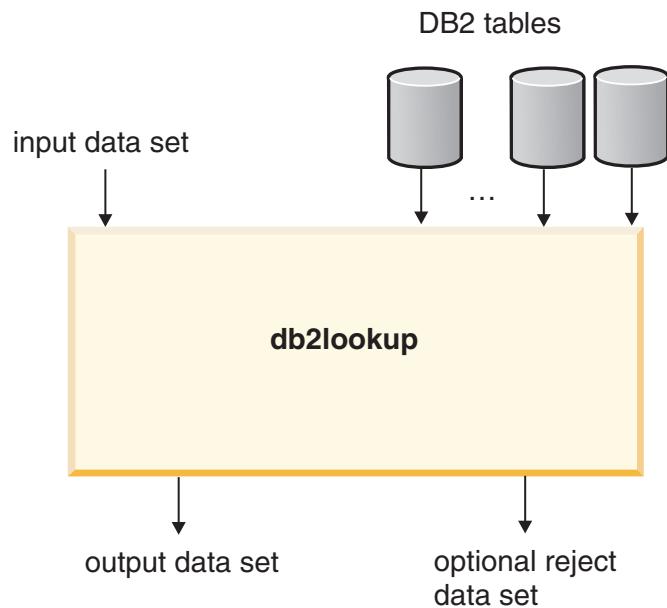
The resulting WebSphere DataStage output data set includes the WebSphere DataStage records and the corresponding rows from each referenced DB2 table. When an DB2 table has a column name that is the same as a WebSphere DataStage data set field name, the DB2 column is renamed using the following syntax:

`APT_integer_fieldname`

An example is `APT_0_lname`. The integer component is incremented when duplicate names are encountered in additional tables.

Note: If the DB2 table is not indexed on the lookup keys, this operator's performance is likely to be poor.

Data flow diagram



db2lookup: properties

Table 172. db2lookup Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1; 2 if you include the -ifNotFound reject option
Input interface schema	determined by the query
Output interface schema	determined by the SQL query
Transfer behavior	transfers all fields from input to output
Execution mode	sequential or parallel (default)
Preserve-partitioning flag in output data set	clear
Composite operator	no

Db2lookup: syntax and options

The syntax for the db2lookup operator is given below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes. You must supply either the -table option or the -query option.

```
db2lookup
  -table table_name [-selectlist selectlist] [-filter filter]
    -key field [-key field ...]
  [-table table_name [-selectlist selectlist] [-filter filter] ...]
    -key field [-key field ...] ...
  |
  [-query sql_query]
  [-close close_command]
  [-client_instance client_instance_name]
    [-client_dbname database]
    -user user_name
    -password password]
  [-db_cs character_set]
```

```

[-dbname dbname]
[-ifNotFound fail | drop| reject | continue]
[-open open_command]
[-padchar char]
[-server remote_server_name]

```

You must specify either the -query option or one or more -table options with one or more -key fields.

Table 173. db2lookup Operator Options

Option	Use
-table	<p>-table <i>table_name</i> [-selectlist <i>selectlist</i>] [-filter <i>filter</i>] -key <i>field</i> [-key <i>field</i> ...]</p> <p>Specify a table and key field(s) from which a SELECT statement is created.</p> <p>Specify either the -table option or the -query option.</p>
-query	<p>-query <i>sql_query</i></p> <p>Specify an SQL query, enclosed in single quotes, to read a table. The query specifies the table and the processing that you want to perform on the table as it is read into WebSphere DataStage. This statement can contain joins, views, database links, synonyms, and so on.</p>
-close	<p>-close <i>close_command</i></p> <p>Optionally apply a closing command to execute on a database after WebSphere DataStage execution.</p>
-client_instance	<p>-client_instance <i>client_instance_name</i> [-client_dbname <i>database</i>] -user <i>user_name</i> -password <i>password</i></p> <p>Specifies the client DB2 instance name. This option is required for a remote connection.</p> <p>The -client_dbname suboption specifies the client database alias name for the remote server database. If you do not specify this option, WebSphere DataStage uses the value of the -dbname option, or the value of the APT_DBNAME environment variable, or DB2DBDFT; in that order.</p> <p>The required -user and -password suboptions specify a user name and password for connecting to DB2.</p>
-db_cs	<p>-db_cs <i>character_set</i></p> <p>Specify the character set to map between DB2 char and varchar values and WebSphere DataStage ustring schema types and to map SQL statements for output to DB2. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data.</p> <p>For information on national language support, refer to this IBM ICU site:</p> <p>http://oss.software.ibm.com/icu/charset</p>
-dbname	<p>-dbname <i>dbname</i></p> <p>Optionally specify a database name.</p>

Table 173. db2lookup Operator Options (continued)

Option	Use
-ifNotFound	-ifNotFound fail drop reject <i>dataset</i> continue Determines what action to take in case of a lookup failure. The default is fail. If you specify reject, you must designate an additional output data set for the rejected records.
-open	-open <i>open_command</i> Optionally apply an opening command to execute on a database prior to WebSphere DataStage execution.
-padchar	padchar <i>string</i> Specify a string to pad string and ustring fields that are less than the length of the DB2 CHAR column. For more information see "Using the -padchar Option".
-server	-server <i>remote_server_name</i> Optionally supply the name of a current DB2 instance.
-use_strings	-use_strings Directs WebSphere DataStage to import DB2 char and varchar values to WebSphere DataStage as WebSphere DataStage strings without converting them from their ASCII or binary form. This option overrides the db_cs option which converts DB2 char and Varchar values as ustrings using the specified character set.

Example

Suppose you want to connect to the APT81 server as user user101, with the password test. You want to perform a lookup between a WebSphere DataStage data set and a table called target, on the key fields lname, fname, and DOB. You can configure db2lookup in either of two ways to accomplish this.

Here is the osh command using the -table and -key options:

```
$ osh " db2lookup -server APT81
      -table target -key lname -key fname -key DOB
      < data1.ds > data2.ds "
```

Here is the equivalent osh command using the -query option:

```
$ osh " db2lookup -server APT81
      -query 'select * from target
              where lname = Orchestrate.lname
              and fname = Orchestrate.fname
              and DOB = Orchestrate.DOB'
      < data1.ds > data2.ds "
```

WebSphere DataStage prints the lname, fname, and DOB column names and values from the WebSphere DataStage input data set and also the lname, fname, and DOB column names and values from the DB2 table.

If a column name in the DB2 table has the same name as a WebSphere DataStage output data set schema fieldname, the printed output shows the column in the DB2 table renamed using this format:

APT_integer_fieldname

For example, *lname* might be renamed to *APT_0_lname*.

Considerations for reading and writing DB2 tables

Data translation anomalies

Translation anomalies exist under the following write and read operations:

Write operations

A WebSphere DataStage data set is written to a DB2 table in create mode. The table's contents are then read out and reconverted to the WebSphere DataStage format. The final WebSphere DataStage field types differ from the original, as shown in the next table:

Table 174. DB2 Table Write Operations Translation Anomaly

Original WebSphere DataStage Data Type	Converted to DB2 Type	Read Back and Reconverted
fixed-length string field, 255 <= fixed length <= 4000 bytes	VARCHAR(<i>n</i>), where <i>n</i> is the length of the string field	string[max= <i>n</i>] and ustring[max= <i>n</i>] a variable-length string with a maximum length = <i>n</i>
int8	SMALLINT	int16
sfloat	FLOAT	dfloat

Invoke the modify operator to convert the data type of the final WebSphere DataStage field to its original data type. See "Modify Operator".

Read Operations

A DB2 table is converted to an WebSphere DataStage data set and reconverted to DB2 format. The final DB2 column type differs from the original, as shown in the next table:

Table 175. DB2 Table Read Operations Translation Anomaly

Original DB2 Data Type	Converted to WebSphere DataStage Type	Read Back and Reconverted
DOUBLE	dfloat	FLOAT (synonymous with DOUBLE)

Using a node map

Use a WebSphere DataStage node map to run an operator other than the DB2 read and write operators on only the processing nodes containing partitions of a DB2 table. A node map specifies the processing nodes on which an operator runs. This operation is useful when you read a DB2 table, process its contents by means of another operator, and write the results back to DB2.

To constrain an operator to run only on nodes mapped in a node map:

Add this code to the command that runs the operator processing the data that has been read from DB2:
[nodemap ('db2nodes -table *table_name* -dbname *db* -server *s_name*')]

where:

- `-table table_name` specifies the DB2 table that determines the processing nodes on which the operator runs. WebSphere DataStage runs the operator on all processing nodes containing a partition of *table_name*.

- `-dbname db` optionally specifies the name of the DB2 database to access.

By default, WebSphere DataStage uses the setting of APT_DBNAME, if it is defined; otherwise, WebSphere DataStage uses the setting of DB2DBDFT. Specifying `-dbname` overrides both APT_DBNAME and DB2DBDFT.

- `-server s_name` optionally specifies the DB2 instance name for *table_name*.

By default, WebSphere DataStage uses the setting of the DB2INSTANCE environment variable. Specifying `-server` overrides DB2INSTANCE.

The next example reads a table from DB2, then uses db2nodes to constrain the statistics operator to run on the nodes containing a partition of the table:

Here is the osh syntax for this example:

```
$ osh "db2read -table table_1 |  
       statistics -results /stats/results -fields itemNum, price  
       [nodemap ('db2nodes -table table_1')]" > outDS.ds "
```

In this case, all data access on each node is local, that is, WebSphere DataStage does not perform any network activity to process the table.

Chapter 18. The Informix interface library

The Informix interface library contains six operators, grouped in three pairs of read and write operators that share data with specific versions of the IBM Informix dynamic server.

The read operators create a connection to an Informix database, query the database, and translate the resulting two-dimensional array (the Informix result set) to a WebSphere DataStage data set. The write operators are a high-speed method for writing data to an Informix table. Use the operators for the version of Informix that you want to access data from:

Informix versions 9.4 and 10.0

hlpread and hplwrite operators

Informix versions 9.4 and 10.0

infhread and infxwrite operators

Informix version 8.5

xpsread and xpswrite operators

Configuring the INFORMIX user environment

In order to access Informix, the WebSphere DataStage users should configure the Informix user environment by using the Informix configuration process.

Users who perform read and write operations must have valid accounts and appropriate privileges for the Informix databases to which they connect. WebSphere DataStage connects to Informix by using your DataStage user name. This user name must have read and write privileges to the Informix tables that you access. The following activities require Resource privileges:

- using the -partition option on the xpsread option
- In write operations, writing in the create and replace modes

Read operators for Informix

A WebSphere DataStage operator that reads data from an Informix database sets up a connection to an Informix server, sends a query, and converts the resulting two-dimensional array (the Informix result set) to a WebSphere DataStage data set.

The chief characteristics of the WebSphere DataStage operators that perform read operations of Informix result sets are listed below:

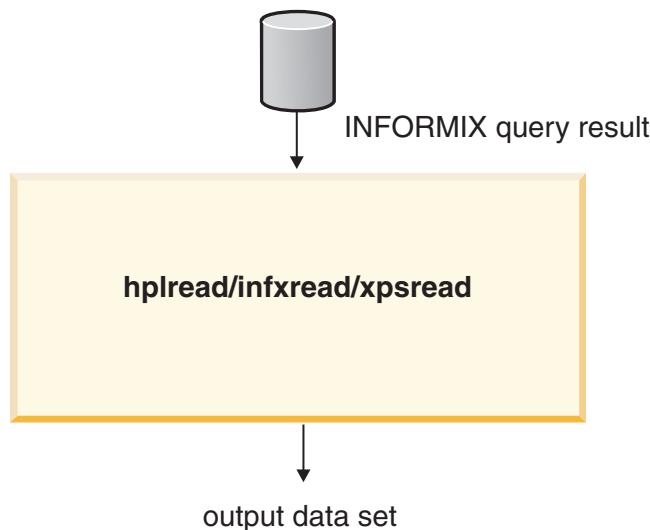
- They specify either an Informix table to read data from or an SQL query to run. A table specification is translated to an SQL select*from table statement.
- They translate the query result set, which is a two-dimensional array, row by row to a WebSphere DataStage data set..
- The translation includes the conversion of Informix data types to WebSphere DataStage data types.

The xpsread operator runs parallel; the infhread operator run in parallel only when you read data by using Informix version 9.4 and 10.0.

Operator output is a WebSphere DataStage data set that you can use as input to a subsequent WebSphere DataStage operator.

A read operation must specify the database on which the operation is performed. A read operation must also indicate either an SQL query of the database or a table to read. Read operations can specify Informix SQL statements to be parsed and run on all processing nodes before the read operation is prepared and run, or after the selection or query is completed.

Data flow diagram



Read operator action

Here are the chief characteristics of the WebSphere DataStage operators that perform read operations of INFORMIX result sets:

- They specify either an INFORMIX table to read (see "Example 1: Reading All Data from an INFORMIX Table") or an SQL query to carry out. A table specification is translated to an SQL select*from table statement.
- They translate the query's result set, which is a two-dimensional array, row by row to a WebSphere DataStage data set, as described in "Column Name Conversion" .
- The translation includes the conversion of INFORMIX data types to WebSphere DataStage data types, as listed in "Data Type Conversion" .

The xpsread operator executes in parallel; the infxread operator executes in parallel only when using versions INFORMIX 8.3 and 8.4.

Operator output is a WebSphere DataStage data set that you can use as input to a subsequent WebSphere DataStage operator.

Read operations must specify the database on which the operation is performed and must also indicate either an SQL query of the database or a table to read. Read operations can, optionally, specify INFORMIX SQL statements to be parsed and executed on all processing nodes before the read operation is prepared and executed, or after the selection or query is completed.

The discussion of each operator in subsequent sections includes an explanation of properties and options.

Execution mode

If you are using Informix version 9.4 or 10.0, you can control the processing nodes, which corresponds to Informix servers, on which the read operators are run. You use the -part suboption of the -table or -query option to control the processing nodes.

Column name conversion

An Informix result set is defined by a collection of rows and columns.

An Informix query result set is converted to a WebSphere DataStage data set, this is shown in the following table

Table 176.

Informix result set corresponds to	What in WebSphere Data Stage data set
Rows of an Informix result set	Records of a WebSphere DataStage data set
Columns of an Informix row	Fields of a WebSphere DataStage
Name and data type of an Informix column	Name and data type of a WebSphere DataStage field
A null value in an Informix column	A null value in WebSphere DataStage field.

Data type conversion

The INFORMIX read operators convert INFORMIX data types to WebSphere DataStage data types.

The following table shows the data type conversions from Informix to WebSphere DataStage data types.

Table 177. Informix interface read operators data-type conversion

Informix data type	WebSphere DataStage data type
CHAR(n)	string[n]
CHARACTER VARYING(n,r)	string[max = n]
DATE	date
DATETIME	date, time, or timestamp with corresponding fractional precision for time. In INFORMIX versions 9.4 and 10.0, if the date starts with a year component and ends with a month, the result is a date field. If the date starts with a year component, the result is a timestamp field. If the time starts with an hour, the result is a time field.
DECIMAL[p,s]	decimal(p,s) where p is precision and s is the scale The maximum precision is 32. A decimal with floating scale is converted to a data type.
DOUBLE-PRECISION	dfloat
FLOAT	dfloat
INTEGER	int32
MONEY	decimal
NCHAR(n,r)	string[n]
NVARCHAR(n,r)	string[max = n]
REAL	sfloat
SERIAL	int32

Table 177. Informix interface read operators data-type conversion (continued)

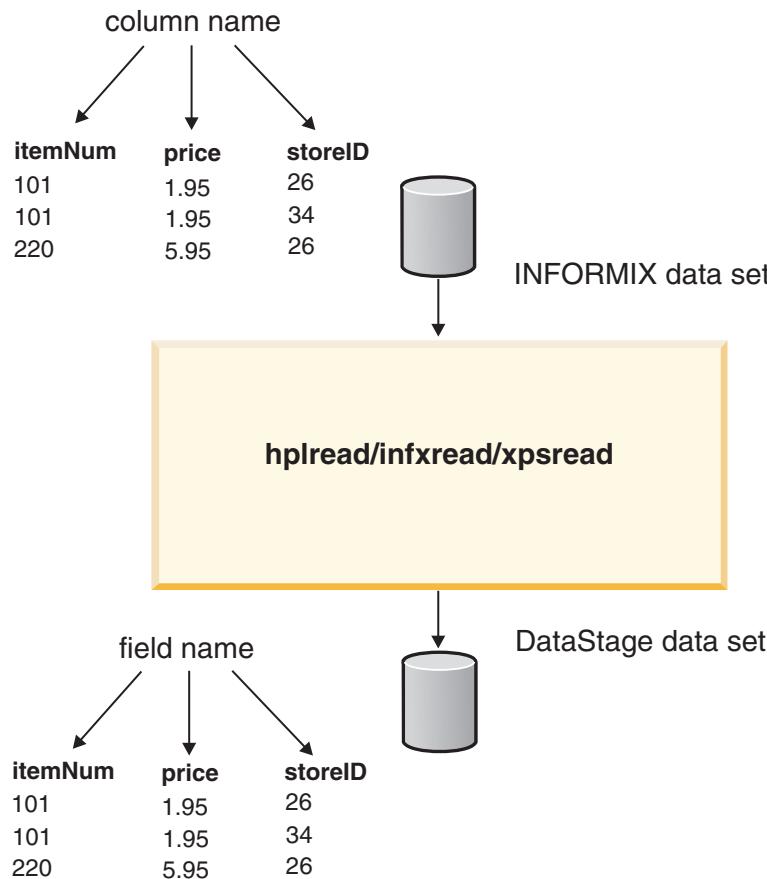
Informix data type	WebSphere DataStage data type
SMALLFLOAT	sfloat
SMALLINT	int16
VARCHAR(<i>n</i>)	string[max = <i>n</i>]

Informix example 1: Reading all data from an Informix table

WebSphere Data Stage can read all the data in an Informix table using an option of Informix read operator.

The following figure shows an example of a read operation of an Informix result set. The operator specifies the -table option, with no further qualification, so that a single table is read in its entirety from Informix. The table is read into a result set, and the operator translates the result set into a WebSphere DataStage data set.

In this example, the table contains three rows. All rows are read.



The following table shows the schemas of both the input Informix table rows and output WebSphere DataStage records.

Table 178. Schemas for Input Informix table row and corresponding WebSphere DataStage result set

Input Informix table row	Output WebSphere DataStage record
itemNum INTEGER not null	itemNum:int32;
price DECIMAL(3,2) not null	price:decimal[3,2];
storeID SMALLINT not null	storeID:int16;

In the above example the WebSphere DataStage operator converts the Informix result set, which is a two-dimensional array, to a WebSphere DataStage data set. The schema of the Informix result set corresponds to the record schema of the WebSphere DataStage data set. Each row of the INFORMIX result set corresponds to a record of the WebSphere DataStage data set; each column of each row corresponds to a field of the WebSphere DataStage record. The WebSphere DataStage field names are the same as the column names of the Informix table.

Here is the osh command for this example:

```
$ osh "...read_operator -dbname my_db
-server IXServer
-table table_2 ..."
```

Write operators for Informix

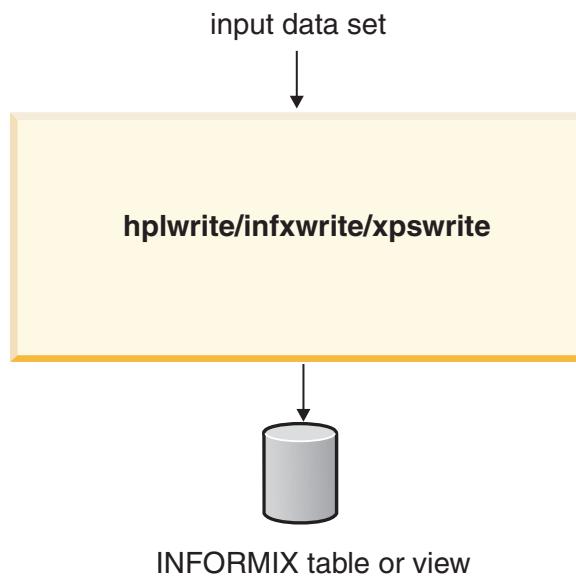
A WebSphere DataStage operator that writes to an Informix database sets up a connection to an Informix database and writes data from an WebSphere DataStage data set to a database table or view.

The chief characteristics of the WebSphere DataStage operators that perform write operation to Informix tables are listed below

- They specify the name of the database to write to.
- They translate the WebSphere DataStage data set record-by-record to INFORMIX table rows.
- The translation includes the conversion of Informix data types to WebSphere DataStage data types.
- They append records to an existing table, unless another mode of writing has been set.

Operators specify such information as the mode of the write operation and Informix commands to be parsed and executed on all processing nodes before the write operation is executed or after it has completed.

Data flow diagram



Operator action

Here are the chief characteristics of the WebSphere DataStage operators that write WebSphere DataStage data sets to INFORMIX tables:

- They specify the name of the database to write to.
- They translate the WebSphere DataStage data set record-by-record to INFORMIX table rows, as discussed in "Column Name Conversion".
- The translation includes the conversion of INFORMIX data types to WebSphere DataStage data types, as listed in "Data Type Conversion".
- They append records to an existing table, unless another mode of writing has been set. "Write Modes" discusses these modes.

Operators optionally specify such information as the mode of the write operation and INFORMIX commands to be parsed and executed on all processing nodes before the write operation is executed or after it has completed.

The discussion of each operator in subsequent sections includes an explanation of properties and options.

Execution mode

The infxwrite and hplwrite operator executes sequentially, writing to one server. The xpswrit operator executes in parallel.

Column name conversion

WebSphere DataStage data sets are converted to Informix tables in the following situations

- A WebSphere DataStage data set corresponds to an Informix table.
- The records of the WebSphere DataStage data set correspond to the rows of the table.
- The fields of the WebSphere DataStage record correspond to the columns of each row.
- The name and data type of a WebSphere DataStage field correspond to the name and data type of the corresponding Informix column.

- Both WebSphere DataStage fields and Informix columns support null values, and a null value in a WebSphere DataStage field is stored as a null value in the corresponding Informix column.

Data type conversion

The Informix interface write operators convert WebSphere DataStage data types to Informix data types. The following table shows how they are converted:

Table 179. Informix interface write operators data type conversion

WebSphere DataStage Data Type	Informix Data Type
date	DATE; For Informix 9.4 and 10.0, date type can also be converted to an Informix datetime data type if the column in the table is a datetime year-to-month data type.
decimal[p,s] (p=precision; s=scale)	DECIMAL[p,s]
dfloat	FLOAT
int8	SMALLINT
int16	SMALLINT
int32	INTEGER
raw[n] (fixed length)	CHAR(n)
raw [max] (variable length)	VARCHAR(n) (default is 32 bytes; max <= 255)
sfloat	FLOAT
string[n] (fixed length)	CHAR(n)
string[max] (variable length)	VARCHAR(n) (default is 32 bytes; max <= 255)
subrec	Unsupported
tagged aggregate	Unsupported
time	DATETIME The DATETIME data type starts with an hour and contains 5 fractional digits if the WebSphere DataStage time field resolves to microseconds.
timestamp	DATETIME The DATETIME data type starts with a year and contains 5 fractional digits if the timestamp field resolves to microseconds. For Informix versions 9.4 and 10.0, the timestamp data type can also be converted to a datetime year-to-month type.
uint8, uint16, uint32	Not supported

Write modes

The write mode of the operator determines how the records of the data set are inserted into the destination table.

The write mode can have one of the following values:

- append:** The table must exist and the record schema of the data set must be compatible with the table. A write operator appends new rows to the table; and the final order of rows in the table is determined by INFORMIX. This is the default mode. See "Example 2: Appending Data to an Existing INFORMIX Table" .

- **create**: The operator creates a new table. If a table exists with the same name as the one you want to create, the step that contains the operator terminates with an error. You must specify either create mode or replace mode if the table does not exist.
- **replace**: The operator drops the existing table and creates a new one in its place. If a table exists with the same name as the one you want to create, it is overwritten.
- **truncate**: The operator retains the table attributes but discards existing records and appends new ones.

Each mode requires specific user privileges.

Matching WebSphere DataStage fields with columns of Informix table

The names and data types of fields in the WebSphere DataStage data set must match the names and datatypes of the columns of the Informix table. However, the fields do not have to appear in the same order.

The following rules determine which fields in a WebSphere DataStage data set are written to an Informix table:

- If the WebSphere DataStage data set contains fields for which there are no matching columns in the Informix table, the step that contains the operator returns an error and stops. However, you can remove an unmatched field from processing either by specifying the -drop option or by using a modify operator to drop the extra field or fields.
- If the Informix table contains a column that does not have a corresponding field in the WebSphere DataStage data set, the write operator writes a null if the column is nullable. Otherwise, WebSphere DataStage issues an error and terminates the step.
- If an Informix write operator tries to write a data set whose fields contain a data type listed above, the write operation terminates and the step containing the operator returns an error. You can convert unsupported data types by means of the modify operator.

Note: Important: The default length of WebSphere DataStage variable-length strings is 32 bytes, so all records of the Informix table have 32 bytes allocated for each variable-length string field in the input data set. If a variable-length field is longer than 32 bytes, the write operator issues an error, but you can use the -stringlength option to modify the default length, up to a maximum of 255 bytes.

Note: Remember:

1. The names of WebSphere DataStage fields can be of any length, but the names of Informix columns cannot exceed 128 characters for versions 9.4 and 10.0. If you write WebSphere DataStage fields that exceed this limit, the operator issues an error and the step stops
2. The WebSphere DataStage data set cannot contain fields of these types: raw, tagged aggregate, subrecord, and unsigned integer (of any size).

Limitations

Write operations have the following limitations:

- While the names of WebSphere DataStage fields can be of any length, the names of INFORMIX columns cannot exceed 18 characters for versions 7.x and 8.x and 128 characters for version 9.x. If you write WebSphere DataStage fields that exceed this limit, the operator issues an error and the step terminates.
- The WebSphere DataStage data set cannot contain fields of these types: raw, tagged aggregate, subrecord, or unsigned integer (of any size).
- If an INFORMIX write operator tries to write a data set whose fields contain a data type listed above, the write operation terminates and the step containing the operator returns an error. You can convert unsupported data types by means of the modify operator. See "Modify Operator" for information.

Example 2: Appending data to an existing Informix table

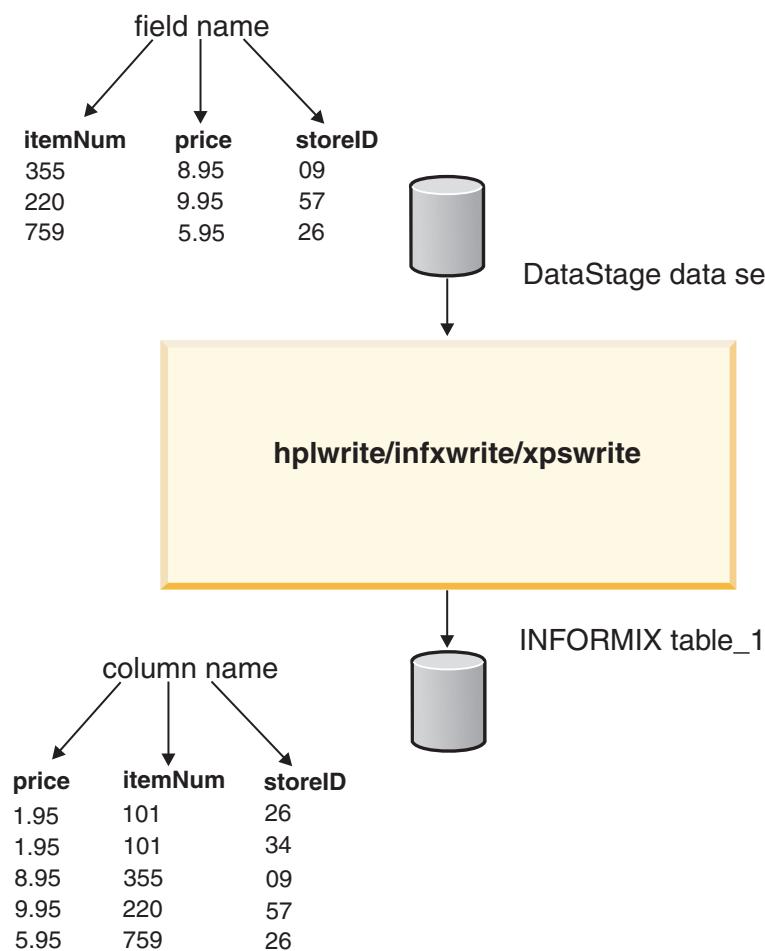
In this example, an Informix write operator appends the contents of a WebSphere DataStage data set to an existing Informix table. The append action is the default action of the Informix write operators.

The record schema of the WebSphere DataStage data set and the row schema of the Informix table correspond to one another, and field and column names are identical. The following table shows the input WebSphere DataStage record schema and output Informix row schema.

Table 180. Schemas for appending WebSphere DataStage records to rows in Informix table

Input WebSphere DataStage record	Output Informix table schema
itemNum:int32; price:decimal[3,2]; storeID:int16;	price DECIMAL, itemNum INTEGER, store IDSMALLINT

The following diagram shows the operator action. Columns that the operator appends to the Informix table are shown in boldface type. The order of fields in the WebSphere DataStage record and columns of the Informix rows are not the same but are written successfully.



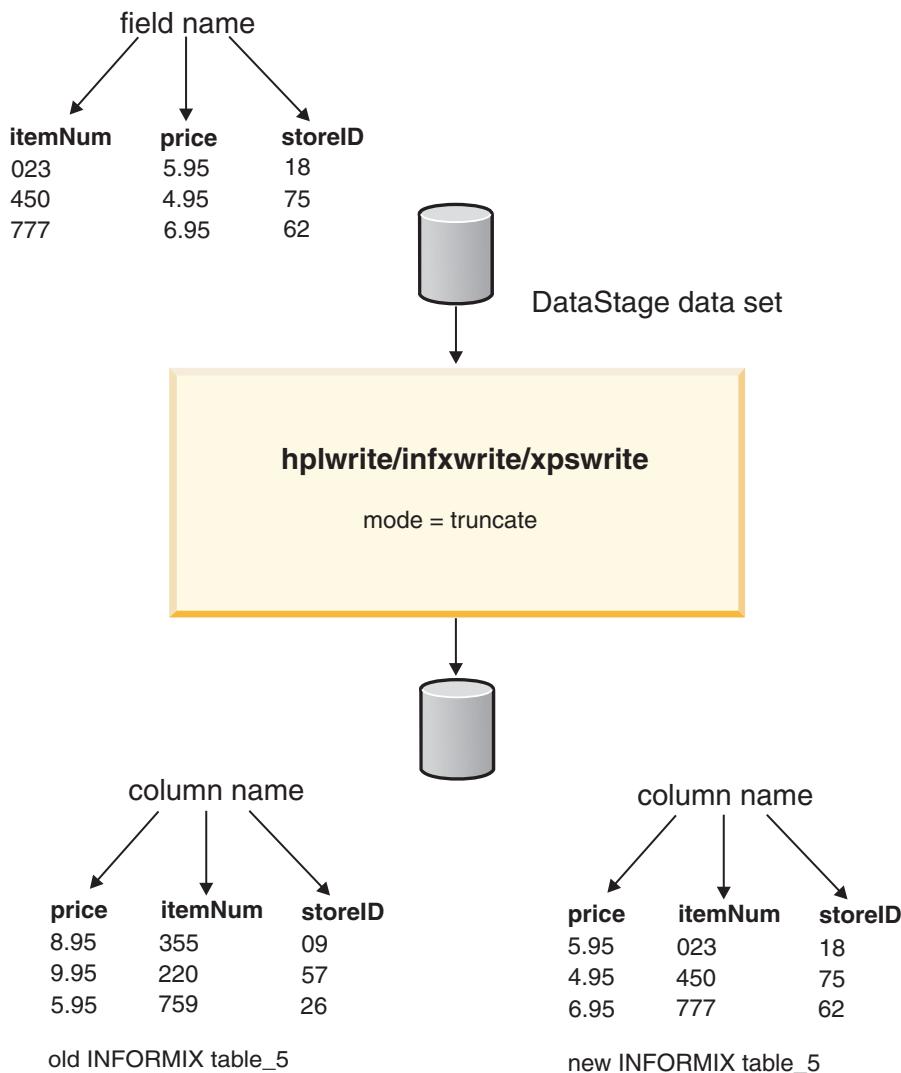
Here is the osh command for this example:

```
$ osh "... write_operator -table table_1
-server IXServer
-database my_db ... "
```

Example 3: writing data to an INFORMIX table in truncate mode

In this example, the Informix write operator writes new data to a table (table_5) whose schema and column name is retained. The operator does so by specifying a write mode of truncate. The operator deletes old data and writes new data under the old column names and data types.

The record schema of the WebSphere DataStage data set and the row schema of the Informix table correspond to one another, and field and column names are identical. The attributes of old and new Informix tables are identical. Table attributes include indexes and fragmentation.



Here is the osh command for this example.

```
$ osh " ... write_operator -table table_5
-server IXserver
-database my_db -mode truncate ... "
```

Example 4: Handling unmatched WebSphere DataStage fields in an Informix write operation

In this example, the records of the WebSphere DataStage data set contain a field that has no equivalent column in the rows of the Informix table. If the operator attempts to write a data set with such an unmatched field, the corresponding step returns an error and stops. The -drop option of the operator removes the unmatched field from processing and writes a warning to the message stream, as follows:

Input field `field_name` dropped because it does not exist in table `table_name`

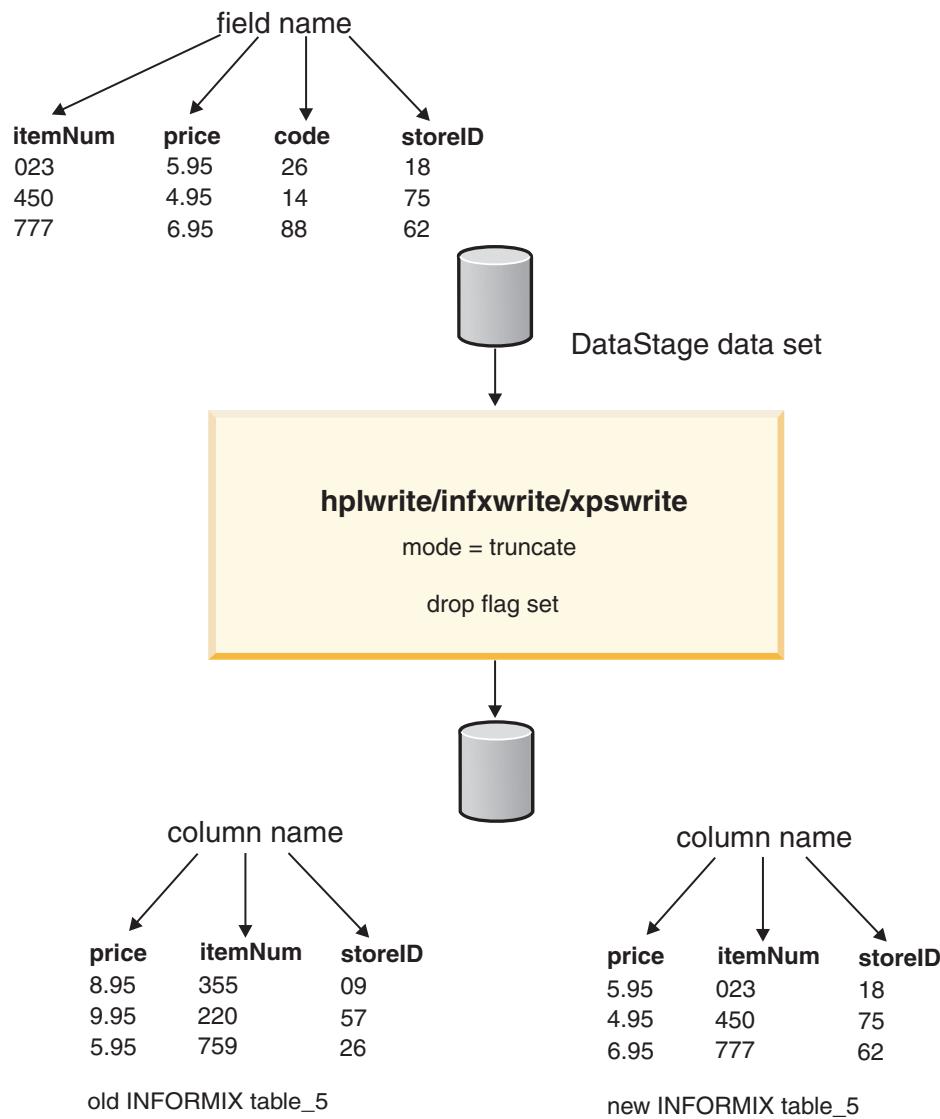
`field_name`

is the name of the WebSphere DataStage filed that is dropped from the operation

`table_name`

is the name of the Informix table to which data is being written

In the example, which operates in truncate write mode, the field named `code` is dropped and a message to that effect is displayed.



Here is the osh command for this example:

```
$ osh " ... write_operator -table table_5
-server IXserver
-dbname my_db -mode truncate ... "
```

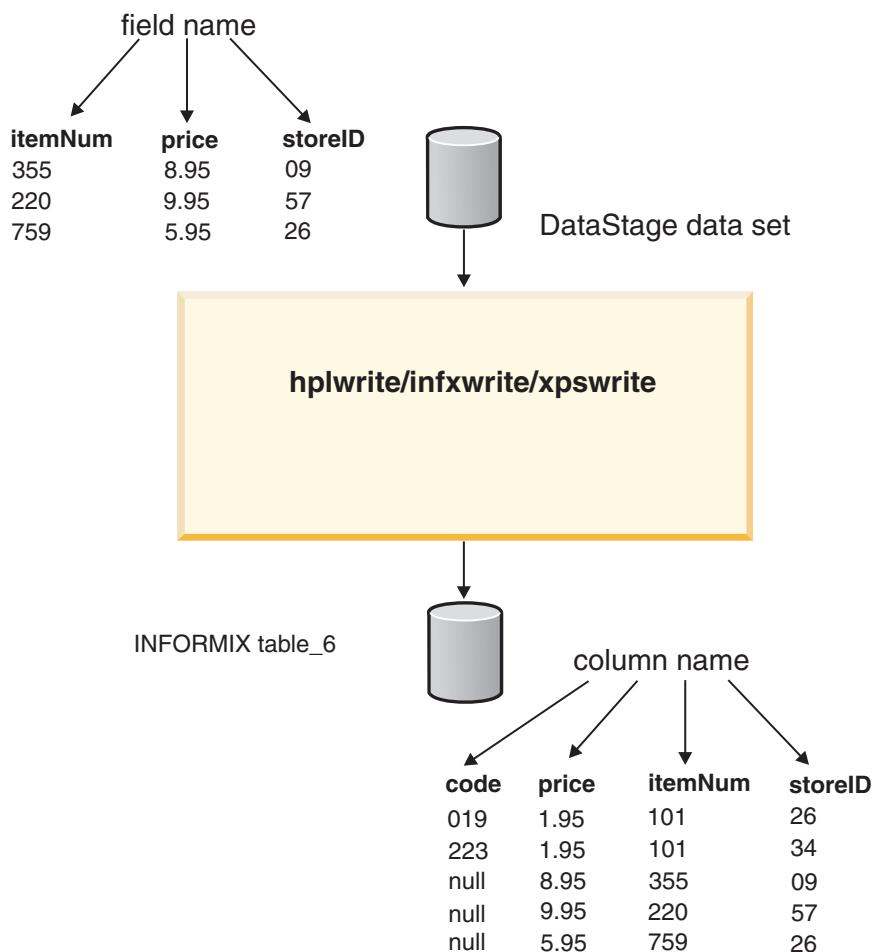
Example 5: Writing to an INFORMIX table with an unmatched column

In this example, the INFORMIX column does not have a corresponding field in the WebSphere DataStage data set, and the operator writes the INFORMIX default to the unmatched column. The operator functions in its default write mode of append.

This table lists the WebSphere DataStage and INFORMIX schemas:

Table 181. Schemas for writing WebSphere DataStage records to Informix tables with Unmatched Column

Input WebSphere DataStage record	Output Informix table row
itemNum :int32; price:decimal[3,2]; storeID:int16;	code SMALLINT, price DECIMAL, itemNum INTEGER, storeID SMALLINT



Here is the osh command for this example:

```
$ osh " ...write_operator -dbname my_db  
-server IXserver  
-table table_6 ... "
```

hpread operator

Before you can use the hpread operator, you must set up the Informix onupload database. Run the Informix iupload utility to create the database. The hpread operator sets up a connection to an Informix database, queries the database, and translates the resulting two-dimensional array that represents the Informix result set to a WebSphere DataStage data set. This operator works with Informix versions 9.4 and 10.0.

Note:

1. The High Performance Loader uses more shared memory, than Informix does in general. If the High Performance Loader is unable to allocate enough shared memory, the hpread might not work. For more information about shared memory limits, contact your system administrator.
2. If your step is processing a few records, you can use the infxread operator rather than hpread. In any case, there is no performance benefit from using hpread for very small data sets. If your data set size is less than about 10 times the number of processing nodes, you must set -dboptions smalldata on the hplwrite operator.

Special operator features

- For small numbers of records, infxread ("The infxread Operator") might perform better than hpread. In any case, there is no performance benefit from using hpread for very small data sets.
- For the High Performance Loader to run (and thus for hpread to run), the INFORMIX onupload database must exist and be set up. This can be assured by running the INFORMIX iupload utility once and then exiting it. An appropriate warning appears if the database is not set up properly.
- The High Performance Loader uses more shared memory, and therefore more semaphores, than INFORMIX does in general. If the HPL is unable to allocate enough shared memory or semaphores, hpread might not work. For more information about shared memory limits, contact your system administrator.

Establishing a remote connection to the hpread operator

You can set up the hpread operator to run on a remote machine without having Informix installed on your local computer.

Troubleshooting configuration

Prerequisite: The remote computer must be cross-mounted with the local computer.

1. Verify that the Informix sqlhosts file on the remote machine has a TCP interface.
2. Copy the Informix etc/sqlhosts file from the remote computer to a directory on the local computer. Set the Informix HPLINFORMIXDIR environment variable to this directory.
For example, if the directory on the local computer is /apt/informix, the sqlhosts file should be in the /apt/informix/directory, and the HPLINFORMIXDIR variable should be set to /apt/informix.
3. Set the INFORMIXSERVER environment variable to the name of the remote Informix server.
4. Add the remote Informix server nodes to your node configuration file in \$APT_ORCHHOME/.../config; and use a nodepool resource constraint to limit the execution of the hpread operator to these nodes. The nodepool for the remote nodes is arbitrarily named InformixServer. The configuration file must contain at least two nodes, one for the local machine and one for the remote machine.
5. The remote variable needs to be set in a startup script which you must create on the local machine. Here is a sample startup.apt file with HPLINFORMIXDIR being set to /usr/informix/9.4, the INFORMIX directory on the remote machine:

```

#!/bin/sh
INFORMIXDIR=/usr/informix/9.4
export INFORMIXDIR
INFORMIXSQLHOSTS=$INFORMIXDIR/etc/sqlhosts
export INFORMIXDIRSQLHOSTS
shift 2
exec $*

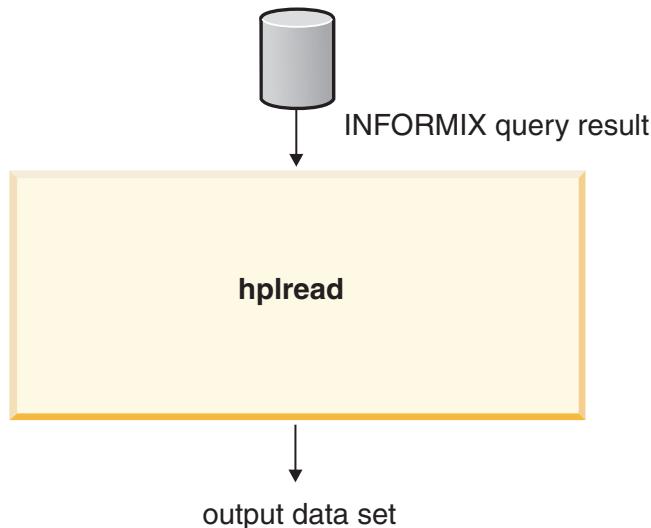
```

- Set the environment variable `APT_STARTUP_SCRIPT` to the full pathname of the `startup.apt` file.

You are now ready to run a job that uses the `hplread` operator to connect to a remote Informix server. If you are unable to connect to the remote server, try making either one or both of the following changes to the `sqlhosts` file on the local computer:

- In the fourth column in the row that corresponds to the remote Informix server name, replace the Informix server name with the Informix server port number in the `/etc/services` file on the remote computer.
- The third column contains the host name of the remote computer. Change this value to the IP address of the remote computer.

Data flow diagram



Properties of the `hplread` operator

Informix `hplread` operator properties is used to read data from an Informix table

Table 182. hplread operator properties

Property	Value
Number of input data sets	0
Number of output data sets	1
Input interface schema	none
Output interface schema	determined by the SQL query
Transfer behavior	none
Execution mode	sequential
Partitioning method	not applicable
Composite operator	yes

Hplread: syntax and options

The osh syntax of the hplread operator follows. You must specify either the -query or the -table option.

Option values that you supply are shown in italic typeface. When a value contains a space or a tab character, you must enclose it in single quotation marks.

```
hplread
  -table table_name [-filter filter] [-selectlist list] | -query sql_query
  [-dbname database_name]
  [-close close_command]
  [-dboptions smalldata] [-open open_command] [-part part] [-server server_name]
```

Table 183. hplread operator options

Option	Use
-close	<p>-close <i>close_command</i></p> <p>Specify an Informix SQL statement to be parsed and executed on all processing nodes after the table selection or query is completed. This parameter is optional.</p>
-dbname	<p>-dbname <i>database_name</i></p> <p>Specify the name of the Informix database to read from.</p>
-dboptions	<p>-dboptions <i>smalldata</i></p> <p>Set this option if the number of records is fewer than 10 * <i>number of processing nodes</i>.</p>
-open	<p>-open <i>open_command</i></p> <p>Specify an Informix SQL statement to be parsed and executed by the database on all processing nodes before the read query is prepared and executed. This parameter is optional</p>
-query	<p>-query <i>sql_query</i></p> <p>Specify a valid SQL query to submit to Informix to read the data.</p> <p>The <i>sql_query</i> variable specifies the processing that Informix performs as data is read into WebSphere DataStage. The query can contain joins, views, synonyms, and so on.</p>
-server	<p>-server <i>server_name</i></p> <p>Set the name of the Informix server that you want to connect to. If no server is specified, the value of the INFORMIXSERVER environment variable is used.</p>
-table	<p>-table <i>table_name</i> [-filter <i>filter</i>] [selectlist <i>list</i>]</p> <p>Specify the name of the Informix table to read from. The table must exist. You can prefix the name of the table with a table owner in the form:</p> <p><i>table_owner.table_name</i></p> <p>The operator reads the entire table unless you limit its scope by using the -filter or -selectlist options, or both.</p> <p>With the -filter suboption, you can specify selection criteria to be used as part of an SQL statement WHERE clause, to specify the rows of the table to include in or exclude from the WebSphere DataStage data set.</p> <p>With the -selectlist suboption, you can specify a list of columns to read, if you do not want all columns to be read..</p>

Example

Refer to "Example 1: Reading All Data from an INFORMIX Table" .

hplwrite operator for Informix

The hplwrite operator is used to write data to an Informix table. The hplwrite operator creates a connection to the Informix database to write data to an WebSphere DataStage data set

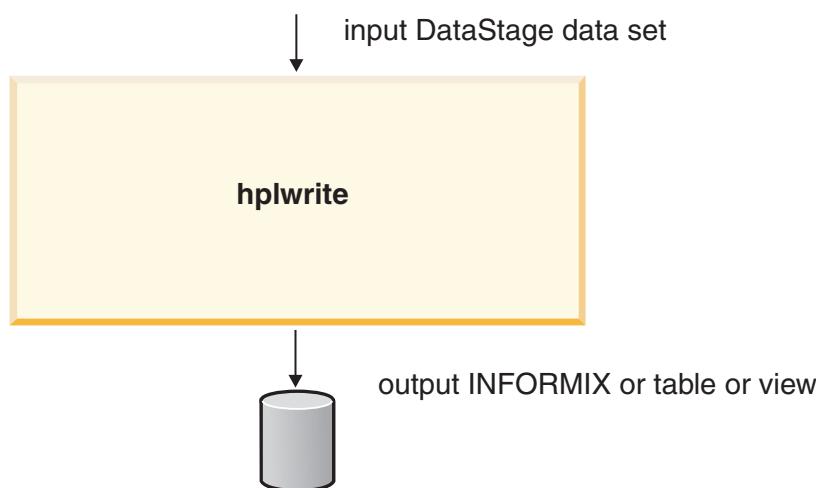
. Before you can use the hplread operator, you must set up the Informix ipload database. Run the Informix ipload utility to create the database. If the destination Informix table does not exist, the hplwrite operator creates a table. This operator runs with INFORMIX Versions 9.4 and 10.0. Before you can use the hplwrite operator, you must set up the Informix onupload database. Run the Informix ipload utility to create the database. The hplwrite operator sets up a connection to an Informix database, queries the database, and translates the resulting two-dimensional array that represents the Informix result set to a WebSphere DataStage data set. This operator works with Informix versions 9.4 and 10.0.

Note: Remember: The High Performance Loader uses more shared memory, than Informix does in general. If the High Performance Loader is unable to allocate enough shared memory, the hplread might not work. For more information about shared memory limits, contact your system administrator.

Special operator features

1. If your step is processing a few records, you might use infxwrite rather than hplwrite. In any case, there is no performance benefit from using hplwrite for very small data sets. If your data set size is less than about 10 times the number of processing nodes, you must set -dboptions smalldata on the hplwrite operator.
2. For the High Performance Loader and hplwrite to run, the INFORMIX onupload database must exist and be set up. You do this by running the INFORMIX ipload utility once and exiting it. An appropriate warning appears if the database is not set up properly.
3. The High Performance Loader uses more shared memory, and therefore more semaphores, than INFORMIX does in general. If the HPL is unable to allocate enough shared memory or semaphores, hplwrite might not work. For more information about shared memory limits, contact your system administrator.

Data flow diagram



Properties of the hplwrite operator

The Informix hplwrite properties are used to write data to an Informix table

Table 184. hplwrite properties

Property	Value
Number of input data sets	1
Number of output data sets	0
Input interface schema	derived from the input data set
Execution mode	sequential
Partitioning method	same
Composite operator	no

hplwrite: syntax and options

The osh syntax of the hplwrite operator are given below. You must specify exactly one occurrence of the -table option; all other options are optional. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
hplwrite
    -table table_name [selectlist list]
    [-close close_command]
    [-dbname database_name]
    [-dboptions smalldata]
    [-drop]
    [-express]
    [-mode append | create | replace | truncate]
    [-open open_command]
    [-server server name]
    [-stringlength length]
```

Table 185. hplwrite operator options

Option	Use
-close	-close <i>close_command</i> Specify an INFORMIX SQL statement to be parsed and executed by INFORMIX on all processing nodes after the table is populated.
-dbname	-dbname <i>database_name</i> Specify the name of the Informix database that contains the table that is specified by the -table parameter.
-dboptions	-dboptions smalldata Set this option if the number of records is fewer than 10 times the number of processing nodes.
-drop	-drop Use this option to drop, with a warning, all input fields that do not correspond to columns of an existing table. If do you not specify the -drop parameter, an unmatched field generates an error and the associated step stops.

Table 185. hplwrite operator options (continued)

Option	Use
-express	<p>-express</p> <p>Attention: Use this parameter only if speed is more important than valid data. Because normal constraints on writing to an Informix table are turned off, invalid data can be written, and existing data can be corrupted. In addition, there are other issues with the use of this parameter:</p> <ul style="list-style-type: none"> • You must either turn off transaction logging before the operation or back up the database after the operation to enable other sources to write to the database again. • Triggers cannot be set on the data that is written. • You cannot write any row larger than the system page size
-mode	<p>-mode append</p> <p>The hplwrite operator appends new records to the table. The database user who writes in this mode must have Resource privileges. This mode is the default mode.</p> <p>-mode create</p> <p>The hplwrite operator creates a new table. The database user who writes in this mode must have Resource privileges. An error is returned if the table already exists.</p> <p>-mode replace</p> <p>The hplwrite operator deletes the existing table and creates a new one. The database user who writes in this mode must have Resource privileges.</p> <p>-mode truncate</p> <p>The hplwrite operator retains the table attributes but discards existing records and appends new ones. The operator runs more slowly in this mode if the user does not have Resource privileges.</p>
-open	<p>-open <i>open_command</i></p> <p>Specify an Informix SQL statement to be parsed and executed by Informix on all processing nodes before table is accessed.</p>
-server	<p>-server <i>server_name</i></p> <p>Specify the name of the Informix server that you want to connect to.</p>
-stringlength	<p>-stringlength <i>length</i></p> <p>Set the default length of variable-length raw fields or string fields. If you do not specify a length, the default length is 32 bytes. You can specify a length up to 255 bytes.</p>
-table	<p>-table <i>table_name</i> [-selectlist <i>selectlist</i>]</p> <p>Specifies the name of the Informix table to write to. See the mode options for constraints on the existence of the table and the required user privileges.</p> <p>The -selectlist parameter specifies an SQL list that determines which fields are written. If you do not supply the list, the hplwrite operator writes all fields to the table.</p>

Examples

Refer to the following sections for examples of INFORMIX write operator use:

- "Example 2: Appending Data to an Existing INFORMIX Table"

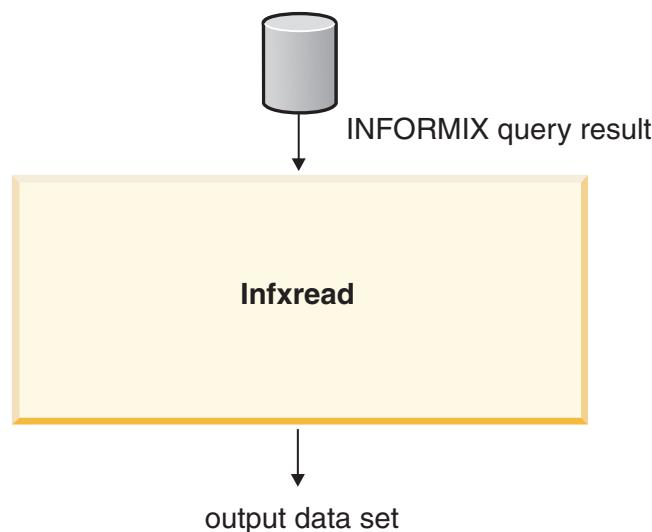
- "Example 3: Writing Data to an INFORMIX Table in Truncate Mode"
- "Example 4: Handling Unmatched WebSphere DataStage Fields in an INFORMIX Write Operation"
- "Example 5: Writing to an INFORMIX Table with an Unmatched Column"

infxread operator

The infxread operator is used to read data from an Informix table. When you have to process fewer records you can use the infxread operator instead of using the hpread operator.

The infxread operator sets up a connection to an Informix database, queries the database, and translates the resulting two-dimensional array that represents the Informix result set to a WebSphere DataStage data set. This operator runs with Informix versions 9.4 and 10.0.

Data Flow Diagram



infxread: properties

The infxread operator properties has to be set in order to read data from an Informix database.

Table 186. infxread operator properties

Property	Value
Number of input data sets	0
Number of output data sets	1
Input interface schema	none
Output interface schema	determined by the SQL query.
Transfer behavior	none
Execution mode	parallel
Partitioning method	not applicable
Preserve-partitioning flag in output data set	clear
Composite operator	no

Infxread: syntax and Options

You must specify either the -query or the -table option.

Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
infxread  
-table table_name  
[-filter filter]  
[-selectlist list]  
|  
-querysql_query  
[-close close_command]  
[-dbname database_name]  
[-open open_command]  
[-part table_name]  
[-server server_name] .
```

Table 187. infxread operator options

Option	Use
-close	<p>-close <i>close_command</i></p> <p>Specify an Informix SQL statement to be parsed and executed on all processing nodes after the table selection or query is completed. This parameter is optional</p>
-dbname	<p>-dbname <i>database_name</i></p> <p>Specify the name of the Informix database to read from. This parameter is optional.</p>
-dboptions	This option is deprecated and is retained only for compatibility with previous WebSphere DataStage releases.
-open	<p>-open <i>open_command</i></p> <p>Specify an Informix SQL statement to be parsed and executed by the database on all processing nodes before the read query is prepared and executed. This parameter is optional.</p>
-part	<p>-part <i>table_name</i></p> <p>If the table is fragmented, specify this option. This option improves performance by creating one instance of the operator per table fragment. If the table is fragmented across nodes, this option creates one instance of the operator per fragment per node. If the table is fragmented and you do not specify -part, the operator functions successfully, if more slowly. You must have Resource privilege to use this option.</p> <p>When this option is used with the -table option, the table name must be the same name as that specified for the -table option.</p>

Table 187. *infxread* operator options (continued)

Option	Use
-query	<p>-query <i>sql_query</i></p> <p>Specify a valid SQL query to be submitted to Informix to read the data. The <i>sql_query</i> variable specifies the processing that Informix performs as data is read into WebSphere DataStage. The query can contain joins, views, synonyms, and so on.</p>
-server	<p>-server <i>server_name</i></p> <p>This option is maintained for compatibility with the <i>xpsread</i> operator. This parameter is optional.</p>
-table	<p>-table <i>table_name</i> [-filter <i>filter</i>] [-selectlist <i>selectlist</i>]</p> <p>Specify the name of the Informix table to read from. The table must exist. You can prefix the table name with a table owner in the form:</p> <p><i>table_owner.table_name</i></p> <p>The operator reads the entire table unless you limit its scope by using the -filter option or the -selectlist option, or both.</p> <p>You can specify selection criteria to use as part of an SQL statement WHERE clause, to specify the rows of the table to include in or exclude from the WebSphere DataStage data set.</p> <p>If you use the -selectlist option, you can specify a list of columns to read, if you do not want all columns to be read.</p>

Example

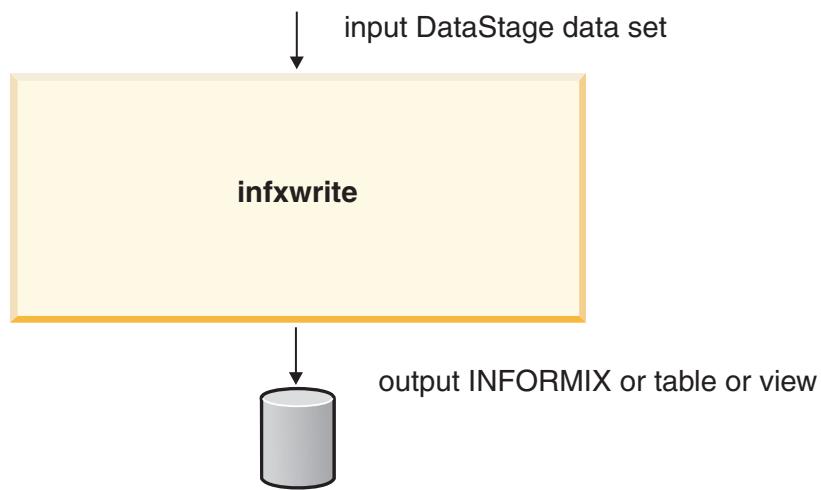
Refer to "Example 1: Reading All Data from an INFORMIX Table".

infxwrite operator

When the number of records to be written to the Informix table is less than the *infxwrite* operator can be used instead of *hplwrite* operator.

The *infxwrite* operator sets up a connection to the Informix database and inserts records into a table. The operator takes a single input data set. The record schema of the data set and the write mode of the operator determine how the records of a data set are inserted into the table.

Data flow diagram



Properties of the infxwrite operator

The properties of the infxwrite operator should be set in order to write data to an Informix database.

Table 188. infxwrite operator properties

Property	Value
Number of input data sets	1
Number of output data sets	0
Input interface schema	derived from the input data set
Execution mode	sequential
Partitioning method	same
Composite operator	no

This operator does not function in parallel. For parallel operation, use the xpswrite or hplwrite operator when running with Informix versions 9.4 or 10.0.

infxwrite: syntax and options

The osh syntax for the infxwrite operator is give below. You must specify exactly one occurrence of the -table option; all other options are optional.

Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
infxwrite
[-tabletable_name
 [-selectlistlist]
 [-closeclose_command]
 [-dbname]
 [-drop-drop] [-mode append | create | replace | truncate]
 [-openopen_command]
 [-serverserver_name]
 [-stringlengthlength]
```

Table 189. *infxwrite* operator options

Option	Use
<code>-close</code>	<p><code>-close close_command</code></p> <p>Specify an Informix SQL statement to be parsed and executed by Informix on all processing nodes after the table is populated.</p>
<code>-dbname</code>	<p><code>-dbname database_name</code></p> <p>Specify the name of the Informix database that contains the table that is specified by table.</p>
<code>-dboptions</code>	<p>This option is deprecated and is retained only for compatibility for previous versions of WebSphere DataStage.</p>
<code>-drop</code>	<p><code>-drop</code></p> <p>Use this option to drop, with a warning, all input fields that do not correspond to columns of an existing table. If do you not specify this option, an unmatched field generates an error and the associated step terminates.</p>
<code>-mode</code>	<p><code>-mode append create replace truncate</code></p> <p>append: <i>infxwrite</i> appends new records to the table. The database user who writes in this mode must have Resource privileges. This is the default mode.</p> <p>create: <i>infxwrite</i> creates a new table; the database user who writes in this mode must have Resource privileges. WebSphere DataStage returns an error if the table already exists.</p> <p>replace: <i>infxwrite</i> deletes the existing table and creates a new one in its place; the database user who writes in this mode must have Resource privileges.</p> <p>truncate: <i>infxwrite</i> retains the table attributes but discards existing records and appends new ones. The operator runs more slowly in this mode if the user does not have Resource privileges.</p>
<code>-open</code>	<p><code>-open open_command</code></p> <p>Specify an Informix SQL statement to be parsed and executed by Informix on all processing nodes before opening the table.</p>
<code>-server</code>	<p><code>-server server_name</code></p> <p>This option is maintained for compatibility with the <code>xpswrite</code> operator.</p>
<code>-stringlength</code>	<p><code>-stringlength length</code></p> <p>Set the default length of variable-length raw fields or string fields. If you do not specify a length, the default length is 32 bytes. You can specify a length up to 255 bytes.</p>

Table 189. *infxwrite* operator options (continued)

Option	Use
-table	<p>-table <i>table_name</i> [-selectlist <i>list</i>]</p> <p>Specifies the name of the Informix table to write to. See the mode options for constraints on the existence of the table and the required user privileges.</p> <p>The -selectlist option specifies an SQL list that determines which fields are written. If you do not supply the list, the <i>infxwrite</i> operator writes the input field records according to your chosen mode specification.</p>

Examples

Refer to the following sections for examples of INFORMIX write operator use:

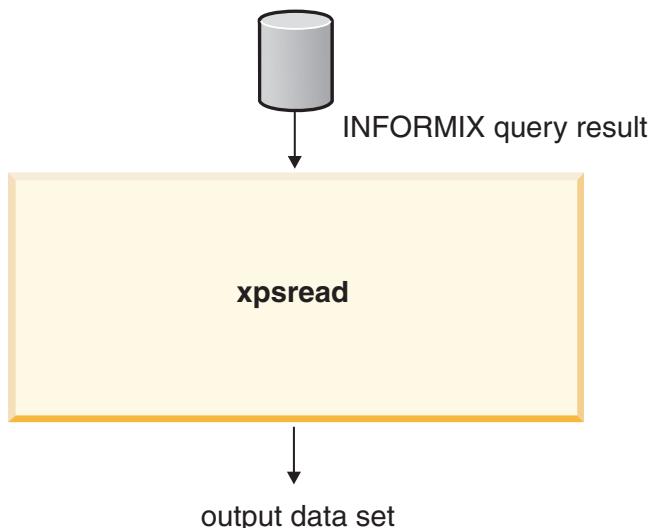
- "Example 2: Appending Data to an Existing INFORMIX Table"
- "Example 3: Writing Data to an INFORMIX Table in Truncate Mode"
- "Example 4: Handling Unmatched WebSphere DataStage Fields in an INFORMIX Write Operation"
- "Example 5: Writing to an INFORMIX Table with an Unmatched Column"

xpsread operator

The *xpsread* operator runs in parallel to other write operators of Informix in versions 9.4 and 10.0.

The *xpsread* operator sets up a connection to an Informix database, queries the database, and translates the resulting two-dimensional array that represents the Informix result set to a WebSphere DataStage data set. This operator works with Informix versions 9.4 and 10.0.

Data flow diagram



Properties of the *xpsread* operator

The properties of the *xpsread* operator should be set in order to read from an Informix database.

Table 190. *xpsread operator options*

Property	Value
Number of input data sets	0
Number of output data sets	1
Input interface schema	none
Output interface schema	determined by the SQL query
Transfer behavior	none
Execution mode	parallel
Partitioning method	not applicable
Preserve-partitioning flag in output data set	clear
Composite operator	yes

Xpsread: syntax and options

The syntax of the xpsread operator is below. You must specify either the -query or the -table option.

Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
xpsread





```

Table 191. *xpsread operator options*

Option	Use
-close	<p>-close close_command</p> <p>Optionally specify an Informix SQL statement to be parsed and executed on all processing nodes after the query runs.</p>
-dbname	<p>-dbname database_name</p> <p>Specify the name of the Informix database to read from.</p>
-dboptions	<p>This option is deprecated and is retained only for compatibility with previous versions of WebSphere DataStage.</p>
-open	<p>-open open_command</p> <p>Optionally specify an Informix SQL statement to be parsed and executed by the database on all processing nodes before the read query is prepared and executed.</p>

Table 191. *xpsread operator options (continued)*

Option	Use
<code>-part</code>	<p><code>-part <i>table_name</i></code></p> <p>If the table is fragmented, you can use this option to create one instance of the operator per table fragment, which can improve performance. If the table is fragmented across nodes, this option creates one instance of the operator per fragment per node. If the table is fragmented and you do not specify this option, the operator functions successfully, but more slowly. You must have Resource privilege to use this option.</p>
<code>-query</code>	<p><code>-query <i>sql_query</i></code></p> <p>Specify a valid SQL query that is submitted to INFORMIX to read the data.</p> <p>The <i>sql_query</i> variable specifies the processing that Informix performs as data is read into WebSphere DataStage. The query can contain joins, views, synonyms, and so on. The <code>-query</code> option is mutually exclusive with the <code>-table</code> option.</p>
<code>-server</code>	<p><code>-server <i>server_name</i></code></p> <p>Specify the name of the Informix server that you want to connect to. If no server is specified, the value of the INFORMIXSERVER environment variable is used.</p>
<code>-table</code>	<p><code>-table <i>table_name</i> [-filter <i>filter</i>] [-selectlist <i>list</i>]</code></p> <p>Specify the name of the INFORMIX table to read from. The table must exist. You can prefix <i>table_name</i> with a table owner in the form:</p> <p><i>table_owner.table_name</i></p> <p>With the <code>-filter</code> suboption, you can optionally specify selection criteria to be used as part of an SQL statement's WHERE clause, to specify the rows of the table to include in or exclude from the WebSphere DataStage data set.</p> <p>If you do not want all columns to be read, use the <code>-selectlist</code> suboption, specify a list of columns to read,</p>

Example

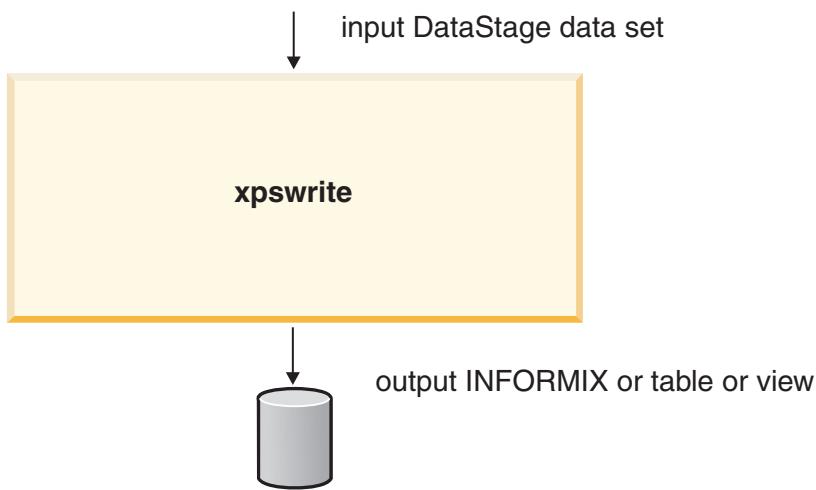
Refer to "Example 1: Reading All Data from an INFORMIX Table" .

xpswrite operator

The xpswrite operator is used to write data to an Informix database. The xpswrite operator can be run in parallel in Informix versions 9.4 and 10.0.

The xpswrite operator sets up a connection to Informix and inserts records into a table. The operator takes a single input data set. The record schema of the data set and the write mode of the operator determine how the records of a data set are inserted into the table. This operator works with Informix version 9.4 and 10.0.

Data flow diagram



Properties of the xpswrite operator

Table 192. xpswrite operator properties

Property	Value
Number of input data sets	1
Number of output data sets	0
Input interface schema	none
Output interface schema	derived from the input record schema of the data set.
Transfer behavior	none
Execution mode	parallel
Partitioning method	same
Preserve-partitioning flag in output data set	not applicable
Composite operator	no

Xpswrite: syntax and options

You must specify exactly one occurrence of the -table option; all other options are optional. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
xpswrite

[selectlistlist]
[-closeclose command]
[-dbnamedatabase_name]
[-drop]
[-mode append | create | replace | truncate]
[-openopen command]
[-serverserver name]
[-stringlengthlength]
```

Table 193. *xpswrite operator options*

Option	Use
-close	<p>-close <i>close_command</i></p> <p>Specify an Informix SQL statement to be parsed and executed by Informix on all processing nodes after the table is populated.</p>
-dbname	<p>-dbname <i>database_name</i></p> <p>Specify the name of the Informix database that contains the table that is specified by table.</p>
-dboptions	This option is deprecated and is retained only for compatibility with previous versions of WebSphere Data Stage.
-drop	<p>-drop</p> <p>Use this option to drop, with a warning, all input fields that do not correspond to columns of an existing table. If you do not specify this option, an unmatched field generates an error and the associated step stops.</p>
-mode	<p>-mode append create replace truncate</p> <p>append: xpswrite appends new records to the table. The database user who writes in this mode must have Resource privileges. This is the default mode.</p> <p>create: xpswrite creates a new table; the database user who writes in this mode must have Resource privileges. WebSphere DataStage returns an error if the table already exists.</p> <p>replace: xpswrite deletes the existing table and creates a new one in its place; the database user who writes in this mode must have Resource privileges.</p> <p>truncate: xpswrite retains the table attributes but discards existing records and appends new ones.</p> <p>The operator runs more slowly in this mode if the user does not have Resource privileges.</p>
-open	<p>-open <i>open_command</i></p> <p>Specify an Informix SQL statement to be parsed and executed by INFORMIX on all processing nodes before opening the table.</p>
-server	<p>-server <i>server_name</i></p> <p>Specify the name of the Informix server you want to connect to.</p>
-stringlength	<p>-stringlength <i>length</i></p> <p>Set the default length of variable-length raw fields or string fields. If you do not specify a length, the default length is 32 bytes. You can specify a length up to 255 bytes.</p>
-table	<p>-table <i>table_name</i> [-selectlist <i>list</i>]</p> <p>Specifies the name of the Informix table to write to. See the mode options for constraints on the existence of the table and the user privileges required.</p> <p>If you do not supply the list, the -selectlist option specifies an SQL list of fields to be written. xpswrite writes all fields to the table.</p>

Examples

Refer to the following for examples of INFORMIX write operator use:

- "Example 2: Appending Data to an Existing INFORMIX Table"
- "Example 3: Writing Data to an INFORMIX Table in Truncate Mode"
- "Example 4: Handling Unmatched WebSphere DataStage Fields in an INFORMIX Write Operation"
- "Example 5: Writing to an INFORMIX Table with an Unmatched Column"

Chapter 19. The Teradata interface library

The Teradata operators access data in Teradata databases.

- The teraread operator sets up a connection to a Teradata database and reads the results of a query into a WebSphere DataStage data set. As the operator requests records, the result set is translated row by row into a WebSphere DataStage data set. .
- The terawrite operator sets up a connection to a Teradata database to write data to it from a WebSphere DataStage data set. As the operator writes records, the WebSphere DataStage data set is translated row by row to a Teradata table.

National language support

WebSphere DataStage's National Language Support (NLS) makes it possible for you to process data in international languages using Unicode character sets. NLS is built on IBM's International Components for Unicode (ICU). For information on National Language Support see *WebSphere DataStage National Language Support Guide* and access the ICU home page:

<http://oss.software.ibm.com/developerworksopensource/icu/project>

The WebSphere DataStage Teradata operators support Unicode character data in usernames, passwords, column names, table names, and database names.

Teradata database character sets

The Teradata database supports a fixed number of character set types for each char or varchar column in a table. Use this query to get the character set for a Teradata column:

```
select 'column_name', chartype from dbc.columns  
      where tablename = 'table_name'
```

The database character set types are:

- Latin: chartype=1. The character set for U.S. and European applications which limit character data to the ASCII or ISO 8859 Latin1 character sets. This is the default.
- Unicode: chartype=2. 16-bit Unicode characters from the ISO 10646 Level 1 character set. This setting supports all of the ICU multi-byte character sets.
- KANJISJIS: chartype=3. For Japanese third-party tools that rely on the string length or physical space allocation of KANJISJIS.
- Graphic: chartype=4. Provided for DB2 compatibility.

Note: The KANJI1: chartype=5 character set is available for Japanese applications that must remain compatible with previous releases; however, this character set will be removed in a subsequent release because it does not support the new string functions and will not support future characters sets. We recommend that you use the set of SQL translation functions provided to convert KANJI1 data to Unicode.

Japanese language support

During the installation of the Teradata server software, you are prompted to specify whether Japanese language support should be enabled. If it is enabled, the Teradata data-dictionary tables use the Kanji1 character set type to support multi-byte code points in database object names such as tables and columns. When not enabled, the Teradata data-dictionary tables use the Latin character set to support single-byte code points for object names.

Specifying a WebSphere DataStage ustring character set

The WebSphere DataStage Teradata interface operators perform character set conversions between the Teradata database characters sets and the multi-byte Unicode ustring field type data. You use the -db_cs option of the Teradata interface operators to specify a ICU character set for mapping strings between the database and the Teradata operators. The default value is Latin1.

Its syntax is:

```
teraread | terawrite -db_cs icu_character_set
```

For example:

```
terawrite -db_cs ISO-8859-5
```

Your database character set specification controls the following conversions:

- SQL statements are mapped to your specified character set before they are sent to the database via the native Teradata APIs.
- If you do not want your SQL statements converted to this character set, set the APT_TERA_NO_SQL_CONVERSION environment variable. This variable forces the SQL statements to be converted to Latin1 instead of the character set specified by the -db_cs option.
- All char and varchar data read from the Teradata database is mapped from your specified character set to the ustring schema data type (UTF-16). If you do not specify the -db_cs option, string data is read into a string schema type without conversion.
- The teraread operator converts a varchar(n) field to ustring[n/min], where min is the minimum size in bytes of the largest codepoint for the character set specified by -db_cs.
- ustring schema type data written to a char or varchar column in the Teradata database is converted to your specified character set.
- When writing a varchar(n) field, the terawrite operator schema type is ustring[n * max] where max is the maximum size in bytes of the largest codepoint for the character set specified by -db_cs.

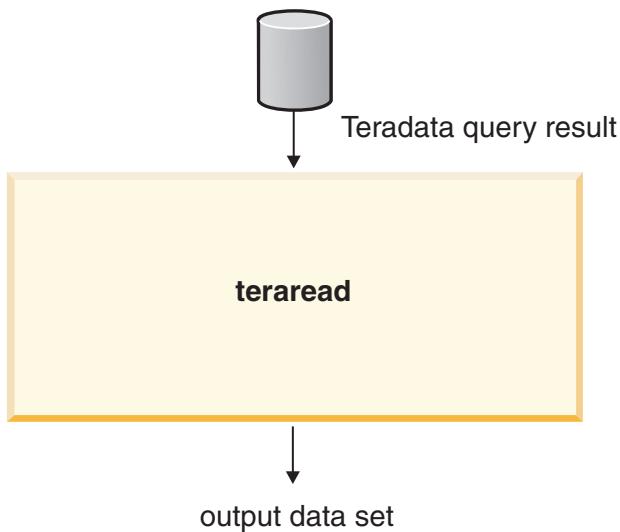
No other environment variables are required to use the Teradata operators. All the required libraries are in /usr/lib which should be on your PATH.

To speed up the start time of the load process slightly, you can set the APT_TERA_NO_PERM_CHECKS environment variable to bypass permission checking on several system tables that need to be readable during the load process.

Teraread operator

The teraread operator sets up a connection to a Teradata database and reads the results of a query into a WebSphere DataStage data set. As the operator requests records, the result set is translated row by row into the WebSphere DataStage data set. The operator converts the data type of Teradata columns to corresponding WebSphere DataStage data types, as listed in Table 19-2.

Data flow diagram



teraread: properties

Table 194. teraread Operator Properties

Property	Value
Number of output data sets	1
Output interface schema	determined by the SQL query.
Execution mode	parallel (default) or sequential

Note: Consider the following points:

- An RDBMS such as Teradata does not guarantee deterministic ordering behavior, unless an SQL query constrains it to do so.
- The teraread operator is not compatible with Parallel CLI, which should not be enabled. In any case, no performance improvements can be obtained by using Parallel CLI with teraread.

By default, WebSphere DataStage reads from the default database of the user who performs the read operation. If this database is not specified in the Teradata profile for the user, the user name is the default database. You can override this behavior by means of the -dbname option. You must specify either the -query or the -table option.

The operator can optionally pass -open and -close option commands to the database. These commands are run once by Teradata on the conductor node either before a query is executed or after the read operation is completed.

Specifying the query

You read the results of a Teradata query by specifying either the -query or the -table option and the corresponding suboptions.

- The -query option defines an SQL query to read Teradata data. WebSphere DataStage evaluates the query and performs the read operation.
- The -table option specifies a table, the rows to include in or exclude from a read operation, and the columns to read. These specifications are then used internally to construct a query.

By default, the operator displays a progress message for every 100,000 records per partition it processes, for example:

```
##I TTER000458 16:00:50(009) <teraread,0> 98300000 records processed.
```

However, you can either change the interval or disable the messages by means of the -progressInterval option.

Column name and data type conversion

A Teradata result set has one or more rows and one or more columns. A Teradata result set is converted to a WebSphere DataStage data set as follows:

- The rows of a Teradata result set correspond to the records of a WebSphere DataStage data set.
- The columns of a Teradata row correspond to the fields of a WebSphere DataStage record.
- The name and data type of a Teradata column determine the name and data type of the corresponding WebSphere DataStage field.
- Both Teradata columns and WebSphere DataStage fields support nulls, and a null contained in a Teradata column is stored as a null in the corresponding WebSphere DataStage field.

WebSphere DataStage gives the same name to its fields as the Teradata column name. However, while WebSphere DataStage field names appear in either upper or lowercase, Teradata column names appear only in uppercase.

The teraread operator automatically converts Teradata data types to WebSphere DataStage data types, as in the following table:

Table 195. teraread Operator Data Type Conversions

Teradata Data Type	WebSphere DataStage Data Type
byte(size)	raw[size]
byteint	int8
char(size)	string[size]
date	date
decimal(m , n)	decimal(m , n)
double precision	dfloat
float	dfloat
graphic(size)	raw[max= size]
Integer	int32
long varchar	string[max= size]
long vargraphic	raw[max= size]
numeric(m , n)	decimal(m , n)
real	dfloat
smallint	int16
time	not supported
timestamp	not supported
varbyte(size)	raw[max= size]
varchar(size)	string[max= size]
vargraphic(size)	raw[max= size]

You can use the modify operator to perform explicit data type conversion after the Teradata data type has been converted to a WebSphere DataStage data type.

teraread restrictions

The teraread operator is a distributed FastExport and is subject to all the restrictions on FastExport. Briefly, these are:

- There is a limit to the number of concurrent FastLoad and FastExport jobs. Each use of the Teradata operators (teraread and terawrite) counts toward this limit.
- Aggregates and most arithmetic operators in the SELECT statement are not allowed.
- The use of the USING modifier is not allowed.
- Non-data access (that is, pseudo-tables like DATE or USER) is not allowed.
- Single-AMP requests are not allowed. These are SELECTs satisfied by an equality term on the primary index or on a unique secondary index.

Teraread: syntax and Options

The syntax of the teraread operator is given below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
teraread
  -table table_name [-filter filter] [-selectlist list]
  |
  -query sqlquery
  [-db_cs character_set]
  -server servername
  -dboptions
    '{-user = username -password = password workdb=work_database
  [-sessionsperplayer = nn] [-requestedsessions = nn]
  [-synctimeout = timeout_in_secs}'
    [-close close_command; close_command]
  [-dbname database_name]
  [-open open_command; open_command]
  [-progressInterval number]
```

You must specify either the -query or the -table option. You must also specify the -server option to supply the server name, and specify the -dboptions option to supply connection information to log on to Teradata.

Table 196. teraread Operator Options

Option	Use
-close	<p>-close <i>close_command</i>; <i>close_command</i></p> <p> Optionally specifies a Teradata command to be run once by Teradata on the conductor node after the query has completed. You can specify multiple SQL statements separated by semi-colons. Teradata will start a separate transaction and commit it for each statement.</p>
-db_cs	<p>-db_cs <i>character_set</i></p> <p> Optionally specify the character set to be used for mapping strings between the database and the Teradata operators. The default value is Latin1. See "Specifying a WebSphere DataStage ustring Character Set" for information on the data affected by this setting.</p>

Table 196. *teraread* Operator Options (continued)

Option	Use
-dbname	<p>-dbname <i>database_name</i></p> <p>By default, the read operation is carried out in the default database of the Teradata user whose profile is used. If no default database is specified in that user's Teradata profile, the user name is the default database. This option overrides the default.</p> <p>If you supply the <i>database_name</i>, the database to which it refers must exist and you must have the necessary privileges.</p>
-dboptions	<p>-dboptions '{-user = <i>username</i> -password = <i>password</i> workdb= <i>work_database</i>[-sessionsperplayer = <i>nn</i>] [-requestedsessions = <i>nn</i>] [-synctimeout=<i>nnn</i>] }'</p> <p>You must specify both the <i>username</i> and <i>password</i> with which you connect to Teradata.</p> <p>If the user does not have CREATE privileges on the default database, the workdb option allows the user to specify an alternate database where the error tables and work table will be created.</p> <p>The value of -sessionsperplayer determines the number of connections each player has to Teradata. Indirectly, it also determines the number of players. The number selected should be such that (sessionsperplayer * number of nodes * number of players per node) equals the total requested sessions. The default is 2.</p> <p>Setting the value of -sessionsperplayer too low on a large system can result in so many players that the step fails due to insufficient resources. In that case, -sessionsperplayer should be increased.</p> <p>The value of the optional -requestedsessions is a number between 1 and the number of vprocs in the database. The default is the maximum number of available sessions.</p> <p>synctimeout specifies the time that the player slave process waits for the control process. The default is 20 seconds.</p>
-open	<p>-open <i>open_command</i>; <i>open_command</i></p> <p>Optionally specifies a Teradata command run once by Teradata on the conductor node before the query is initiated. You can specify multiple SQL statements separated by semi-colons. Teradata will start a separate transaction and commit it for each statement.</p>
-progressInterval	<p>-progressInterval <i>number</i></p> <p>By default, the operator displays a progress message for every 100,000 records per partition it processes. Specify this option either to change the interval or disable the messages. To change the interval, specify a new number of records per partition. To disable the messages, specify 0.</p>

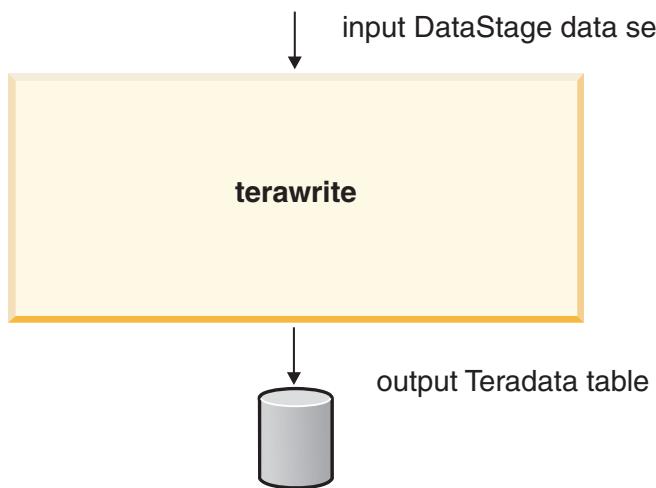
Table 196. teraread Operator Options (continued)

Option	Use
-query	<p>-query <i>sqlquery</i></p> <p>Specifies a valid Teradata SQL query in single quotes that is submitted to Teradata. The query must be a valid Teradata SQL query that the user has the privileges to run. Do not include formatting characters in the query.</p> <p>A number of other restrictions apply; see "teraread Restrictions".</p>
-server	<p>-server <i>servername</i></p> <p>You must specify a Teradata server.</p>
-table	<p>-table <i>table_name</i> [-filter <i>filter</i>] [-selectlist <i>list</i>]</p> <p>Specifies the name of the Teradata table to read from. The table must exist, and the user must have the necessary privileges to read it. The teraread operator reads the entire table, unless you limit its scope by means of the -filter or -selectlist option, or both options.</p> <p>The -filter suboption optionally specifies selection criteria to be used as part of an SQL statement's WHERE clause. Do not include formatting characters in the query. A number of other restrictions apply; see "teraread Restrictions".</p> <p>The -selectlist suboption optionally specifies a list of columns to read. The items of the list must appear in the same order as the columns of the table.</p>

Terawrite Operator

The terawrite operator sets up a connection to a Teradata database to write data to it from an WebSphere DataStage data set. As the operator writes records, the WebSphere DataStage data set is translated row by row to a Teradata table. The operator converts the data type of WebSphere DataStage fields to corresponding Teradata types, as listed in Table 19-5. The mode of the operator determines how the records of a data set are inserted in the table.

Data flow diagram



terawrite: properties

Table 197. terawrite Operator Properties

Property	Value
Number of input data sets	1
Input interface schema	derived from the input data set
Execution mode	parallel (default) or sequential
Partitioning method	same You can override this partitioning method. However, a partitioning method of entire is not allowed.

Note: The terawrite operator is not compatible with Parallel CLI, which should not be enabled. In any case, no performance improvements can be obtained by using Parallel CLI with terawrite.

By default, WebSphere DataStage writes to the default database of the user name used to connect to Teradata. The userid under which the step is running is not the basis of write authorization. If no default database is specified in that user's Teradata profile, the user name is the default database. You can override this behavior by means of the -dbname option. You must specify the table.

By default, the operator displays a progress message for every 100,000 records per partition it processes, for example:

```
##I TTER000458 16:00:50(009) <terawrite,0> 98300000 records processed.
```

The -progressInterval option can be used to change the interval or disable the messages.

Column Name and Data Type Conversion

Here is how WebSphere DataStage data sets are converted to Teradata tables:

- a WebSphere DataStage data set corresponds to a Teradata table.
- The records of the WebSphere DataStage data set correspond to the rows of the Teradata table.
- The fields of the WebSphere DataStage record correspond to the columns of the Teradata row.

- The name and data type of a WebSphere DataStage field are related to those of the corresponding Teradata column. However, WebSphere DataStage field names are case sensitive and Teradata column names are not. Make sure that the field names in the data set are unique, regardless of case.
- Both WebSphere DataStage fields and Teradata columns support nulls, and a WebSphere DataStage field that contains a null is stored as a null in the corresponding Teradata column.

The terawrite operator automatically converts WebSphere DataStage data types to Teradata data types as shown in the following table:

Table 198. terawrite Operator Data Type Conversions

WebSphere DataStage Data Type	Teradata Data Type
date	date
decimal(<i>p</i> , <i>s</i>)	numeric(<i>p</i> , <i>s</i>)
dfloat	double precision
int8	byteint
int16	smallint
int32	integer
int64	unsupported
raw	varbyte(<i>default</i>)
raw[<i>fixed_size</i>]	byte(<i>fixed_size</i>)
raw[<i>max</i> = <i>n</i>]	varbyte(<i>n</i>)
sfloat	unsupported
string	varchar(<i>default_length</i>)
string[<i>fixed_size</i>]	char(<i>fixed_size</i>)
string[<i>max</i>]	varchar(<i>max</i>)
time	not supported
timestamp	not supported
uint8	unsupported
uint16	unsupported
uint32	unsupported

When terawrite tries to write an unsupported data type to a Teradata table, the operation terminates with an error.

You can use the modify operator to perform explicit data type conversions before the write operation is initiated.

Correcting load errors

The terawrite operator presents a summary of the number of rows processed, the number of errors, and the elapsed time. This summary is useful for determining the state of the operator upon completion, since terawrite "succeeds" even if errors in the data were encountered.

If there were errors, the summary gives a limited indication of the number and nature of the errors; in particular, the field name and Teradata error code is given.

Since FastLoad creates error tables whenever it is run (see your FastLoad documentation for details), you might use BTEQ to examine the relevant error table and then correct the records that failed to load. FastLoad does not load duplicate records, however, such records are not loaded into the error tables.

The remdup operator can be used to remove the duplicate records prior to using terawrite.

Write modes

The write mode of the operator determines how the records of the data set are inserted into the destination table. The write mode can have one of the following values:

- append: The terawrite operator appends new rows to the table; the database user who writes in this mode must have TABLE CREATE privileges and INSERT privileges on the database being written to. This is the default mode.
- create: The terawrite operator creates a new table. The database user who writes in this mode must have TABLE CREATE privileges. If a table exists with the same name as the one you want to create, the step that contains terawrite terminates in error.
- replace: The terawrite operator drops the existing table and creates a new one in its place. The database user who writes in this mode must have TABLE CREATE and TABLE DELETE privileges. If a table exists with the same name as the one you want to create, it is overwritten.
- truncate: The terawrite operator retains the table attributes (including the schema) but discards existing records and appends new ones. The database user who writes in this mode must have DELETE and INSERT privileges on that table.

Note: The terawrite operator cannot write to tables that have indexes other than the primary index defined on them. This applies to the append and truncate modes.

Override the default append mode by means of the -mode option.

Writing fields

Fields of the WebSphere DataStage data set are matched by name and data type to columns of the Teradata table but do not have to appear in the same order.

The following rules determine which fields of a WebSphere DataStage data set are written to a Teradata table:

- If the WebSphere DataStage data set contains fields for which there are no matching columns in the Teradata table, the step containing the operator terminates. However, you can remove an unmatched field from processing: either specify the -drop option and the operator drops any unmatched field, or use the modify operator to drop the extra field or fields before the write operation begins.
- If the Teradata table contains a column that does not have a corresponding field in the WebSphere DataStage data set, Teradata writes the column's default value into the field. If no default value is defined for the Teradata column, Teradata writes a null. If the field is not nullable, an error is generated and the step is terminated.

Limitations

Write operations have the following limitations:

- A Teradata row might contain a maximum of 256 columns.
- While the names of WebSphere DataStage fields can be of any length, the names of Teradata columns cannot exceed 30 characters. If you write WebSphere DataStage fields that exceed this limit, use the modify operator to change the WebSphere DataStage field name.

- WebSphere DataStage assumes that the terawrite operator writes to buffers whose maximum size is 32 KB. However, you can override this and enable the use of 64 KB buffers by setting the environment variable APT_TERA_64K_BUFFERS.
- The WebSphere DataStage data set cannot contain fields of the following types:
 - int64
 - Unsigned integer of any size
 - String, fixed- or variable-length, longer than 32 KB
 - Raw, fixed- or variable-length, longer than 32 KB
 - Subrecord
 - Tagged aggregate
 - Vectors

If terawrite tries to write a data set whose fields contain a data type listed above, the write operation is not begun and the step containing the operator fails. You can convert unsupported data types by using the modify operator.

Restrictions

The terawrite operator is a distributed FastLoad and is subject to all the restrictions on FastLoad. In particular, there is a limit to the number of concurrent FastLoad and FastExport jobs. Each use of the teraread and terawrite counts toward this limit.

Terawrite: syntax and options

You must specify the -table option, the -server option to supply the server name, and the -dboptions option to supply connection information to log on to Teradata. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
terawrite
  -table table_name [-selectlist list]
  -server servername
  -db_cs character_set
  -dboptions
    '{-user = username -password = password workdb = work_database [-sessionsperplayer = nn ]
  [-requestedsessions = nn] [-synctimeout = timeout_in_secs]}''
  [-close close_command] [-dbname database_name] [-drop] [-mode] [-open open_command]
  [-primaryindex = 'field1, field2, ... fieldn' ] [-progressInterval number]
  [-stringlength length ]
```

Table 199. terawrite Operator Options

Option	Usage and Meaning
-close	-close <i>close_command</i> Optionally specify a Teradata command to be parsed and executed by Teradata on all processing nodes after the table has been populated.
-db_cs	-db_cs <i>character_set</i> Optionally specify the character set to be used for mapping strings between the database and the Teradata operators. The default value is Latin1. See "Specifying a WebSphere DataStage ustring Character Set" for information on the data affected by this setting.

Table 199. terawrite Operator Options (continued)

Option	Usage and Meaning
-dbname	<p>-dbname <i>database_name</i></p> <p>By default, the write operation is carried out in the default database of the Teradata user whose profile is used. If no default database is specified in that user's Teradata profile, the user name is the default database. If you supply the <i>database_name</i>, the database to which it refers must exist and you must have necessary privileges.</p>
-dboptions	<p>-dboptions '{-user = <i>username</i> , -password = <i>password</i> workdb= <i>work_database</i>[-sessionsperplayer = <i>nn</i>] [-requestedsessions = <i>nn</i>] [-synctimeout=<i>nnn</i>]}'</p> <p>You must specify both the user name and password with which you connect to Teradata.</p> <p>If the user does not have CREATE privileges on the default database, the workdb option allows the user to specify an alternate database where the error tables and work table will be created.</p> <p>The value of -sessionsperplayer determines the number of connections each player has to Teradata. Indirectly, it also determines the number of players. The number selected should be such that (sessionsperplayer * number of nodes * number of players per node) equals the total requested sessions. The default is 2.</p> <p>Setting the value of -sessionsperplayer too low on a large system can result in so many players that the step fails due to insufficient resources. In that case, the value for -sessionsperplayer should be increased.</p> <p>The value of the optional -requestedsessions is a number between 1 and the number of vprocs in the database.</p> <p>synctimeout specifies the time that the player slave process waits for the control process. The default is 20 seconds.</p>
-drop	<p>-drop</p> <p>This optional flag causes the operator to silently drop all unmatched input fields.</p>

Table 199. terawrite Operator Options (continued)

Option	Usage and Meaning
<code>-mode</code>	<p><code>-mode append create replace truncate</code></p> <p>append: Specify this option and terawrite appends new records to the table. The database user must have TABLE CREATE privileges and INSERT privileges on the table being written to. This mode is the default.</p> <p>create: Specify this option and terawrite creates a new table. The database user must have TABLE CREATE privileges. If a table exists of the same name as the one you want to create, the step that contains terawrite terminates in error.</p> <p>replace: Specify this option and terawrite drops the existing table and creates a new one in its place; the database user must have TABLE CREATE and TABLE DELETE privileges. If a table exists of the same name as the one you want to create, it is overwritten.</p> <p>truncate: Specify this option and terawrite retains the table attributes, including the schema, but discards existing records and appends new ones. The database user must have DELETE and INSERT privileges on the table.</p>
<code>-open</code>	<p><code>-open open_command</code></p> <p>Optionally specify a Teradata command to be parsed and executed by Teradata on all processing nodes before the table is populated.</p>
<code>-primaryindex</code>	<p><code>-primaryindex = ' field 1, field 2, ... fieldn '</code></p> <p>Optionally specify a comma-separated list of field names that become the primary index for tables. Format the list according to Teradata standards and enclose it in single quotes.</p> <p>For performance reasons, the WebSphere DataStage data set should not be sorted on the primary index. The primary index should not be a smallint, or a field with a small number of values, or a high proportion of null values. If no primary index is specified, the first field is used. All the considerations noted above apply to this case as well.</p>
<code>-progressInterval</code>	<p><code>-progressInterval number</code></p> <p>By default, the operator displays a progress message for every 100,000 records per partition it processes. Specify this option either to change the interval or to disable the messages. To change the interval, specify a new number of records per partition. To disable the messages, specify 0.</p>
<code>-server</code>	<p><code>-serverservername</code></p> <p>You must specify the name of a Teradata server.</p>

Table 199. terawrite Operator Options (continued)

Option	Usage and Meaning
-stringlength	<p>-stringlength <i>length</i></p> <p>Optionally specify the maximum length of variable-length raw or string fields. The default length is 32 bytes. The upper bound is slightly less than 32 KB.</p>
-table	<p>-table <i>tablename</i> [-selectlist <i>list</i>]</p> <p>Specify the name of the table to write to. The table name must be a valid Teradata table name.</p> <p>-selectlist optionally specifies a list that determines which fields are written. If you do not supply the list, terawrite writes to all fields. Do not include formatting characters in the list.</p>

Chapter 20. The Sybase interface library

The Sybase operators enable sharing of data between a Sybase database and WebSphere DataStage.

These topics describe the operators underlying the WebSphere DataStage Sybase Enterprise stage. It is intended for users who are familiar with the OSH language utilized by the WebSphere DataStage parallel engine.

WebSphere DataStage can communicate with Sybase ASE (ASE) and Sybase IQ (IQ), Relational Database Management Systems in four different modes . The Stages work uniformly across on both ASE and IQ databases and for all purposes, Sybase refers to both Sybase ASE and Sybase IQ databases. Some specific cases where there are functional and behavioral scope changes in ASE and IQ are described in the document.

- The sybasereade operator is used for reading from one or more tables of a Sybase database.
- The sybasewrite operator is used to load into a single table of a Sybase database.
- The sybaseupsert operator inserts into and updates and deletes one or more tables of a Sybase database.
- The sybaselookup operator is used to efficiently join large tables of a Sybase database, with small data sets.

Accessing Sybase from WebSphere DataStage

To access Sybase from WebSphere DataStage, follow the Sybase configuration process.

Sybase client configuration

The standard installation of the Sybase Database Server automatically installs the Sybase open clinet, which the operators use to communicate with the Sybase database. WebSphere DataStage communicates with the Sybase database using the CT® library. To support this:

- Set your SYBASE_OCS Environment variable to the name of the Open Client Directory in the Client Installation. This can be obtained from the server login environment. (For example, for Sybase 12.5 it is OCS-12_5).
- Set your SYBASE environment variable to your Sybase installation directory.
- If the database is Sybase ASE, set your SYBASE_ASE environment variable (for example, for ASE 12.5 server it is ASE-12_5). If the database is Sybase IQ server, set your ASDIR environment variable (for example, for IQ 12.5 server it is ASIQ-12_5)
- Set up the Interface file, adding the details about the database server (database name, host machine name or IP address and port number)
- Add SYBASE/bin to your PATH and &SYBASE/\$ SYBASE_OCS/lib to your LIBPATH, LD_LIBRARY_PATH, or SHLIB_PATH.
- Have login privileges to Sybase using a valid Sybase user name and a corresponding password. These must be recognized by Sybase before you attempt to access it.

Note: \$SYBASE/\$SYBASE_OCS/bin must appear first in your PATH. This is to ensure that \$SYBASE/\$SYBASE_OCS/bin/isql being executed always, when user executes "isql" command.

National Language Support

WebSphere DataStage's National Language Support (NLS) makes it possible for you to process data in international languages using Unicode character sets. WebSphere DataStage uses International Components for Unicode (ICU) libraries to support NLS functionality. For information on National Language Support, see *WebSphere DataStage NLS Guide* and access the ICU home page:

<http://oss.software.ibm.com/developerworksopensource/icu/project>

The WebSphere DataStage's Sybase Parallel Enterprise Stage support Unicode character data in schema, table, and index names; in user names and passwords; column names; table and column aliases; SQL*Net service names; SQL statements; and file-name and directory paths.

The stage has one option which optionally control character mapping:

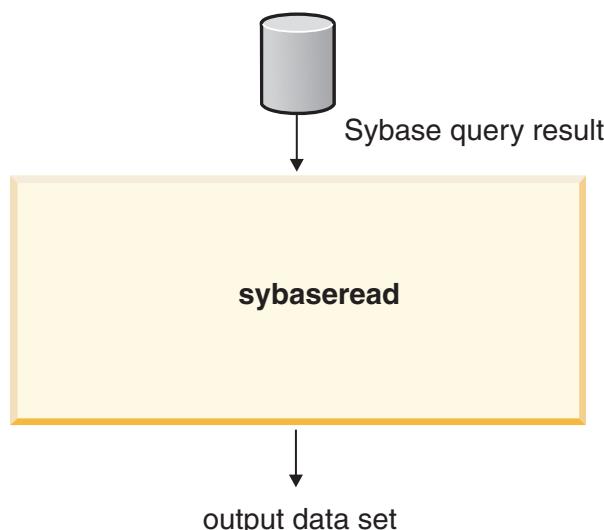
- `-db_cs character_set`

Specifies an ICU character set to map between Sybase char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to Sybase.

The `asesybasereade` and `sybasereade` Operators

The `asesybasereade` and `sybasereade` operators are used for reading from one or more tables of a Sybase database (Sybase ASE/Sybase IQ).

Data flow diagram



`asesybasereade` and `sybasereade`: properties

Table 200. `asesybasereade` and `sybasereade` Properties

Property	Value
Number of input data sets	0
Number of output data sets	1
Input interface schema	None
Output interface schema	Determined by the sql query
Transfer behavior	None

Table 200. *asesybasereade* and *sybasereade* Properties (continued)

Property	Value
Partitioning method	Not applicable
Execution mode	Sequential
Collection method	Not applicable
Preserve-partitioning flag in output data set	Clear
Composite operator	No

Operator Action

The chief characteristics of the *asesybasereade* and *sybasereade* operators are:

- You can direct it to run in specific node pools.
- It translates the query's result set (a two-dimensional array) row by row to a WebSphere DataStage data set.
- Its output is a WebSphere DataStage data set, that you can use as input to a subsequent WebSphere DataStage Stage.
- Its translation includes the conversion of Sybase data types to WebSphere DataStage data types.
- The size of Sybase rows can be greater than that of WebSphere DataStage records.
- The operator specifies either a Sybase table to read or a SQL query to carry out.
- It optionally specifies commands to be run before the read operation is performed and after it has completed the operation.
- You can perform a join operation between WebSphere DataStage data set and Sybase (there might be one or more tables) data.

Where *asesybasereade* and *sybasereade* Run

The *asesybasereade* and *sybasereade* operators run sequentially. The *asesybasereade* operator fetches all the records sequentially even in a partitioned table.

Column name conversion

A Sybase result set is defined by a collection of rows and columns. The *sybasereade* and *asesybasereade* operators translate the query's result set (a two-dimensional array) to a WebSphere DataStage data set. Here is how a Sybase query result set is converted to a WebSphere DataStage data set:

- The rows of the Sybase result set correspond to the records of a WebSphere DataStage data set.
- The columns of the Sybase row correspond to the fields of a WebSphere DataStage record; the name and data type of the Sybase column correspond to the name and data type of a WebSphere DataStage field.
- Names are translated exactly except when the Sybase column name contains a character that WebSphere DataStage does not support. In that case, two underscore characters replace the unsupported character.
- Both Sybase columns and WebSphere DataStage fields support nulls, and a null contained in a Sybase column is stored as key word NULL in the corresponding WebSphere DataStage field.

Data type conversion

The *sybasereade* operator converts Sybase data types to WebSphere DataStage data types, as in the following table:

Table 201. Mapping of SYBASE data types to WebSphere DataStage data types

SYBASE data type	Corresponding WebSphere DataStage data type
tinyint	int8
smallint	int16
int	int32
integer	int32
numeric(p,s)	decimal[p,s]
decimal(p,s)	decimal[p,s]
dec(p,s)	decimal[ps,s]
float	sfloat
double precision	dfloat
real	sfloat
smallmoney	decimal[10,4]
money	decimal[15,4]
smalldatetime	timestamp
datetime	timestamp
date	date
time	time
char(n)	string[n], a fixed-length string with length = n
varchar(n)	string[max=n], a variable-length string with maximum length = n
nchar(n)	or ustring[n], a fixed-length string with length = n, only for ASE
nvarchar(n)	ustring[max=n], a variable-length string with maximum length = n, only for ASE
binary(n)	raw(n)
varbinary(n)	raw[max=n]
bit	int8
unsigned int	uint32
money	decimal[15,4]

Data Types that are not listed in the table above generate an error.

Targeting the read operation

When reading a Sybase table you can either specify the table name that allows WebSphere DataStage to generate a default query that reads the total records of the table, or you can explicitly specify the query.

Specifying the Sybase table name

If you choose the -table option, WebSphere DataStage issues the following SQL SELECT statement to read the table:

```
select [selectlist]
      from table_name
      and (filter);
```

You can specify optional parameters to narrow the read operation. They are as follows:

- The selectlist specifies the columns of the table to be read; by default, WebSphere DataStage reads all columns.
- The filter specifies the rows of the table to exclude from the read operation by default, WebSphere DataStage reads all rows.
- You can optionally specify an -open and -close command. These commands are executed by the sybase before the table is opened and after it is closed.

Specifying a SQL SELECT statement

- If you choose the -query option, you pass a SQL query to the operator. The query specifies the table and the processing that you want to perform on the table as it is read into WebSphere DataStage. The SQL statement can contain joins, views, database links, synonyms, and so on. However, the following restrictions apply to -query:
 - The query might not contain bind variables.
 - If you want to include a filter or select list, you must specify them as part of the query.
 - The query runs sequentially.
 - You can specify optional open and close commands. Sybase runs these commands immediately before reading the data from the table and after reading the data from the table.

Join Operations

You can perform a join operation between WebSphere DataStage data sets and Sybase data. First invoke the asesybasereade or sybasereade operator and then invoke either the lookup operator (see page [Lookup Operator](#)) or a join operator (see "The Join Library"). Alternatively, you can use the asesybaselookup or sybaselookup operator.

Asesybasereade and sybasereade: syntax and Options

The syntax for asesybasereade and sybasereade follow. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

The syntax for asesybasereade is identical to the syntax for sybasereade:

```

-server          -- sybase server; exactly one occurrence required
    server      -- string
-table           -- table to load; optional
    tablename   -- string
-db_name         -- use database; optional
    db_name     -- string
-user            -- sybase username; exactly one occurrence required
    username   -- string
-password        -- sybase password; exactly one occurrence required
    password   -- string
-filter          -- conjunction to WHERE clause that specifies which rows are read from
    the table ; used with -table option ; optional
    filter      -- string; default=all rows in table are read
-selectlist      -- SQL select list that specifies which column in table are read ; used
    with -table option ; optional
    list        -- string; default=all columns in table are read
-query           -- SQL query to be used to read from table ; optional
    query      -- string
-db_cs           -- specifies database character set; optional
    db_cs      -- string
-open            -- Command run before opening a table ; optional
    opencommand -- string
-close           -- Command run after completion of processing a table ;
optional
    closecommand -- string
-server          -- sybase server; exactly one occurrence required
    server      -- string

```

```

-table          -- table to load; optional
  tablename    -- string
-db_name       -- use database; optional
  db_name      -- string
-user          -- sybase username; exactly one occurrence required
  username     -- string
-password       -- sybase password; exactly one occurrence required
  password     -- string
-filter         -- conjunction to WHERE clause that specifies which row
are read from the table ; used with -table option ; optional
  filter        -- string; default=all rows in table are read
-selectlist    -- SQL select list that specifies which column in table
are read ; used with -table option ; optional
  list          -- string; default=all columns in table are read
-query         -- SQL query to be used to read from table ; optional
  query        -- string
-db_cs         -- specifies database character set; optional
  db_cs        -- string
-open          -- Command run before opening a table ; optional
  opencommand   -- string
-close         -- Command run after completion of processing a table ;
optional
  closecommand  -- string

```

You might specify either the *-query* or the *-table* option.

Table 202. asesybasereade and sybasereade Options

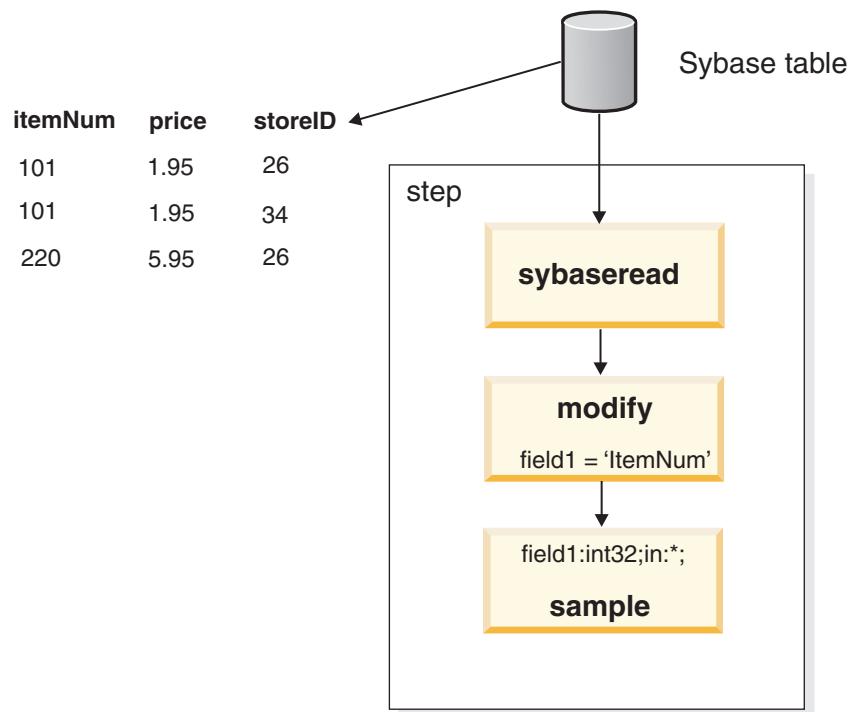
Option	Value
-server	server Specify the name of the server that you want to connect to.
-table	table Specify the name of the table from which you want to read data. This option is mutually exclusive with the <i>-query</i> option. You must have SELECT privileges on the table. The table name might contain a view name.
-db_name	-db_name Specify the database name to be used for all database connections. This option is required.
-user	username Specify the user name used to connect to the data source.
-password	password Specify the password used to connect to the data source.
-filter	filter The <i>-filter</i> suboption optionally specifies a conjunction, enclosed in single quotes, to the WHERE clause of the SELECT statement to specify the rows of the table to include or exclude from reading into DataStage.
-selectlist	list The suboption <i>-selectlist</i> optionally specifies a SQL select list, enclosed in single quotes, that can be used to determine which fields are read.

Table 202. *asesybasereade* and *sybasereade* Options (continued)

Option	Value
-open	opencommand Optionally specify a SQL statement to be executed before the insert array is processed. The statement is executed only once on the conductor node.
-close	closecommand Optionally specify a SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statement is executed only once on the conductor node.
-query	query Specify a SQL query to read from one or more tables. The -query option is mutually exclusive with the -table option.
-db_cs	db_cs Optionally specify the ICU code page, which represents the database character set in use. The default is ISO-8859-1.

Sybasereade example 1: Reading a Sybase Table and Modifying a Field Name

The following figure shows a Sybase table used as input to a WebSphere DataStage Stage:



The Sybase table contains three columns, the data types in the columns are converted by the operator as follows:

- itemNum of type integer is converted to int32
- price of type NUMERIC [6,2] is converted to decimal [6,2]
- storeID of type integer is converted to int32

The schema of the WebSphere DataStage data set created from the table is also shown in this figure. Note that the WebSphere DataStage field names are the same as the column names of the Sybase table.

However, the operator to which the data set is passed has an input interface schema containing the 32-bit integer field field1, while the data set created from the Sybase table does not contain a field of the same name. For this reason, the modify operator must be placed between the read operator and sample operator to translate the name of the field, itemNum, to the name field1

Here is the osh syntax for this example:

```
$ osh "sybasereade -table table_1
-server server_name
-user user1
-password user1
| modify '$modifySpec' | ... "
$ modifySpec="field1 = itemNum;"
modify
('field1 = itemNum,;')
```

The asesybasewrite and sybasewrite Operators

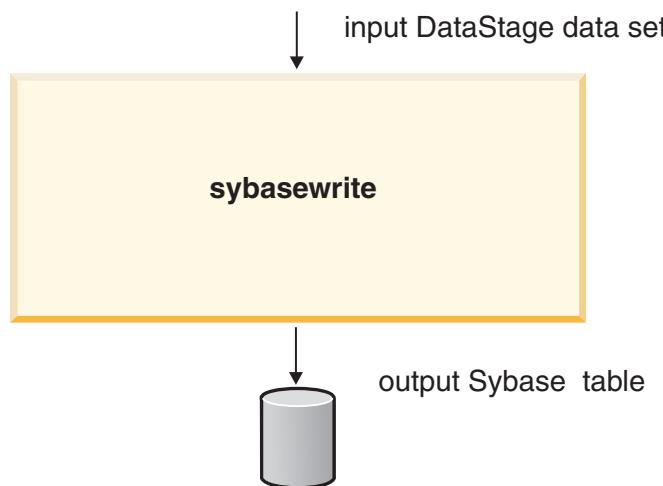
The asesybasewrite and sybasewrite operators set up a connection to Sybase ASE and Sybase IQ respectively, and insert records into a table. The operators take a single input data set. The write mode of the operators determines how the records of a data set are inserted into the table.

Writing to a Multibyte Database

Specify chars and varchars in bytes, with two bytes for each character. This example specifies 10 characters:

```
create table orch_data (col_a varchar (20));
```

Data flow diagram



asesybasewrite and sybasewrite: properties

Table 203. asesybasewrite and sybasewrite Properties

Property	Value
Number of input data sets	1
Number of output data sets	0
Input interface schema	Derived from the input data set
Output interface schema	None
Transfer behavior	None
Execution mode	Sequential default or parallel
Partitioning method	Not applicable
Collection method	Any
Preserve-partitioning flag	Default clear
Composite operator	No

Operator Action

Here are the chief characteristics of the asesybasewrite and sybasewrite operators:

- Translation includes the conversion of WebSphere DataStage data types to Sybase data types.
- The operator appends records to an existing table, unless you set another mode of writing
- When you write to an existing table, the input data set schema must be compatible with the table's schema.
- Each instance of a parallel write operator running on a processing node writes its partition of the data set to the Sybase table. You can optionally specify Sybase commands to be parsed and executed on all processing nodes before the write operation runs or after it completes.
- The **asesybasewrite** operator uses bcp to load data into a table. Bcp can run in fast or slow mode. If any triggers or indexes have been defined on table to write to, bcp automatically runs in slow mode, and you do not have to set any specific database properties. Otherwise, bcp runs in fast mode. However, bcp cannot run in fast mode unless you set the database property **Select into/bulkcopy** to **True**. To set this property, run the following commands by logging in as a system administrator using the iSQL utility.

```
use mastergosp_dboption <database name>, "select into/bulkcopy", truegouse <database name>gocheckpointgo
```

Note: To send bad records down the reject link, you must set the environment variable APT_IMPEXP_ALLOW_ZERO_LENGTH_FIXED_NULL.

Where asesybasewrite and sybasewrite Run

The default execution mode of the asesybasewrite and sybasewrite operators is parallel. The default is that the number of processing nodes is based on the configuration file. However, if the environment variable APT_CONFIG_FILE is set, the number of players is set to the number of nodes.

To direct the asesybasewrite and sybasewrite operators to run sequentially, specify the [seq] framework argument.

You can optionally set the resource pool or a single node on which the operators run.

Data conventions on write operations to Sybase

Sybase columns are named identically to WebSphere DataStage fields, with these restrictions:

- Sybase column names are limited to 30 characters in ASE and 128 characters in IQ. If a WebSphere DataStage field name is longer, you can do one of the following:
 - Choose the -truncate or truncate_length options to configure the operator to truncate WebSphere DataStage field names to the max. length as the data source column name. If you choose truncate_length then you can specify the number of characters to be truncated, and should be less than the max., length the data source supports.
 - Variable length data types -string, raw in WebSphere DataStage is directly mapped to varchar and varbinary respectively. The default size in Sybase is 1 byte. If the operator tries to load data, which is more than 1 byte, an error message is logged. The -stringlength option overrides the default size of 1 byte.
 - Use the modify operator to modify the WebSphere DataStage field name.
- A WebSphere DataStage data set written to Sybase might not contain fields of certain types. If it does, an error occurs and the corresponding step terminates. However WebSphere DataStage offers operators that modify certain data types to ones Sybase accepts, as shown in the Table.

Data type conversion

Table 204. Mapping of WebSphere DataStage data types to SYBASE data types

WebSphere DataStage Data Type	SYBASE Data Type
uint8, int8	tinyint
int16	smallint
int32	int
int32	integer
decimal[p,s]	numeric(p,s)
decimal[p,s]	decimal(p,s)
decimal[ps,s]	dec(p,s)
sffloat	float
dfloat	double precision
sffloat	real
decimal[10,4]	smallmoney
decimal[15,4]	money
timestamp	smalldatetime
timestamp	datetime
date	date
time	time
string[n], a fixed-length string with length = n	char(n)
string[max=n], a variable-length string with maximum length = n	varchar(n)
or ustring[n], a fixed-length string with length = n	nchar(n)
ustring[max=n], a variable-length string with maximum length = n	nvarchar(n)
raw(n)	binary(n)
raw[max=n]	varbinary(n)

Table 204. Mapping of WebSphere DataStage data types to SYBASE data types (continued)

WebSphere DataStage Data Type	SYBASE Data Type
int8	bit
uint32	unsigned int

Write Modes

The write mode of the `asesybasewrite` or `sybasewrite` operator determines how the records of the data set are inserted into the destination table. The write mode can have one of the following values:

- **append:** This is the default mode. The table must exist and the record schema of the data set must be compatible with the table. The write operator appends new rows to the table. The schema of the existing table determines the input interface of the operator.
- **create:** The operator creates a new table. If a table exists with the same name as the one you want to create, the step that contains the operator terminates with an error. The schema of the WebSphere DataStage data set determines the schema of the new table. The table is created with simple default properties. To create a table that is partitioned, indexed, in a non-default table space, or in some other nonstandard way, you can use the `-createtestmt` option with your own create table statement.
- **replace:** The operator drops the existing table and creates a new one in its place. If a table exists of the same name as the one you want to create, it is overwritten. The schema of the WebSphere DataStage data set determines the schema of the new table.
- **truncate:** The operator retains the table attributes but discards existing records and appends new ones. The schema of the existing table determines the input interface of the operator. Each mode requires the specific user privileges shown in the table below:

Note: If a previous write operation fails, you can retry your application specifying a write mode of `replace` to delete any information in the output table that might have been written by the previous attempt to run your program.

Table 205. Sybase privileges for the write operator

Write Mode	Required Privileges
Append	INSERT on existing table
Create	TABLE CREATE
Replace	INSERT and TABLE CREATE on existing table
Truncate	INSERT on existing table

Matched and unmatched fields

The schema of the Sybase table determines the operator's interface schema. Once the operator determines this, it applies the following rules to determine which data set fields are written to the table:

- Fields of the input data set are matched by name with fields in the input interface schema. WebSphere DataStage performs default data type conversions to match the input data set fields with the input interface schema.
- You can also use the `modify` operator to perform explicit data type conversions.
- If the input data set contains fields that do not have matching components in the table, the operator generates an error and terminates the step.
- WebSphere DataStage does not add new columns to an existing table if the data set contains fields that are not defined in the table. Note that you can use either the `sybasewrite -drop` option to drop extra fields from the data set. Columns in the Sybase table that do not have corresponding fields in the

input data set are set to their default value, if one is specified in the Sybase table. If no default value is defined for the Sybase column and it supports nulls, it is set to null. Otherwise, WebSphere DataStage issues an error and terminates the step.

- WebSphere DataStage data sets support nullable fields. If you write a data set to an existing table and a field contains a null, the Sybase column must also support nulls. If not, WebSphere DataStage issues an error message and terminates the step. However, you can use the modify operator to convert a null in an input field to another value.

asesybasewrite and sybasewrite: syntax and Options

Syntax for the asesybasewrite and sybasewrite operators is given below. Option values you supply are shown in *italics*. When your value contains a space or a tab character, you must enclose it in single quotes. Exactly one occurrence of the *-dboptions* option and the *-table* option are required.

Syntax for asesybasewrite:

```

-server          -- sybase server; exactly one occurrence required
    server      -- string
-table           -- table to load; exactly one occurrence required
    tablename   -- string
-db_name         -- use database; optional
    db_name     -- string
-user            -- sybase username; exactly one occurrence required
    username    -- string
-password        -- sybase password; exactly one occurrence required
    password    -- string
-truncate        -- trucate field name to sybase max field name length; optional
-truncateLength -- used with -truncate option to specify a truncation length; optional
    truncateLength -- integer; 1 or larger; default=1
-maxRejectRecords -- used with -reject option to specify maximum records that can be rejected;
    optional maximum reject records
        -- integer; 1 or larger; default=10
-mode            -- mode: one of create, replace, truncate, or append.; optional
    create/replace/append/truncate
        -- string; value one of create, replace, append, truncate; default=append
-createstmt      -- create table statement; applies only to create and replace modes; optional
    statement    -- string
-db_cs           -- specifies database character set; optional
    db_cs        -- string
-string_length   -- specifies default data type length; optional
    string_length -- integer
-drop            -- drop input unmatched fields ; optional
-reject          -- mention the reject link; optional
-open            -- Command run before opening a table ; optional
    opencommand  -- string
-close           -- Command run after completion of processing a table ; optional
    closecommand -- string

```

Table 206. asesybasewrite options

Option	Value
-db_name	db_name Specify the data source to be used for all database connections. If you do not specify one, the default database is used.
-user	username Specify the user name used to connect to the data source. This option might not be required depending on the data source.

Table 206. asesybasewrite options (continued)

Option	Value
-password	<p><code>password</code></p> <p>Specify the password used to connect to the data source. This option might not be required depending on the data source.</p>
-server	<p><code>server</code></p> <p>Specify the server name to be used for all database connections.</p>
-table	<p><code>table</code></p> <p>Specify the table to write to. May be fully qualified.</p>
-mode	<p><code>append create replace truncate</code></p> <p>Specify the mode for the write operator as one of the following:</p> <ul style="list-style-type: none"> • <code>append</code>: new records are appended into an existing table. • <code>create</code>: the operator creates a new table. If a table exists with the same name as the one you want to create, the step that contains the operator terminates with an error. The schema of the WebSphere DataStage data set determines the schema of the new table. The table is created with simple default properties. To create a table that is partitioned, indexed, in a non-default table space, or in some other nonstandard way, you can use the <code>-createtestmt</code> option with your own create table statement. • <code>replace</code>: The operator drops the existing table and creates a new one in its place. The schema of the WebSphere DataStage data set determines the schema of the new table. • <code>truncate</code>: All records from an existing table are deleted before loading new records.
-createtestmt	<p><code>statement</code></p> <p>Optionally specify the create statement to be used for creating the table when <code>-mode create</code> is specified.</p>
-drop	If this option is set unmatched fields of the WebSphere DataStage data set will be dropped. An unmatched field is a field for which there is no identically named field in the datasource table.
-truncate	If this option is set column names are truncated to the maximum size allowed by the Sybase driver.
-truncateLength	<p><code>n</code></p> <p>Specify the length to truncate column names to.</p>
maxRejectRecords	<p><code>maximum reject records</code></p> <p>Specify the maximum number of records that can be sent down the reject link. The default number is 10. You can specify the number one or higher.</p>
-open	<p><code>opencommand</code></p> <p>Optionally specify a SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.</p>
-close	<p><code>closecommand</code></p> <p>Optionally specify a SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statements are executed only once on the conductor node.</p>
-string_length	<p><code>string_length</code></p> <p>Optionally specify the length of a string. The value is an integer.</p>
-reject	Optionally specify the reject link.

Table 206. asesybasewrite options (continued)

Option	Value
-db_cs	db_cs Optional specify the ICU code page, which represents the database character set in use. The default is ISO-8859-1.

Syntax for sybasewrite:

```

-server          -- sybase server; exactly one occurrence required
    server      -- string
-table           -- table to load; exactly one occurrence required
    tablename   -- string
-db_name         -- use database; optional
    db_name     -- string
-user            -- sybase username; exactly one occurrence required
    username    -- string
-password        -- sybase password; exactly one occurrence required
    password    -- string
-truncate        -- trucate field name to sybase max field name length; optional
-manual          -- specify the manual loading of data with script; optional
-dumplocation    -- control file for load table stmt; optional
    Control and data file dump location
    -- string
-truncateLength -- used with -truncate option to specify a truncation length; optional
    truncateLength -- integer; 1 or larger; default=1
-mode            -- mode: one of create, replace, truncate, or append.; optional
    create/replace/append/truncate
    -- string; value one of create, replace, append, truncate; default=append
-db_cs           -- specifies database character set; optional
    db_cs       -- string
-string_length   -- specifies default data type length; optional
    string_length -- integer
-drop            -- drop input unmatched fields ; optional
-open            -- Command run before opening a table ; optional
    opencommand -- string
-close           -- Command run after completion of processing a table ; optional
    closecommand -- string

```

Table 207. sybasewrite options

Option	Value
-db_name	db_name Specify the data source to be used for all database connections. This option is mandatory.
-user	username Specify the user name used to connect to the data source. This option might not be required depending on the data source.
-password	password Specify the password used to connect to the data source. This option might not be required depending on the data source.
-server	server Specify the server name used to connect to the database. This option is optional.
-table	table Specify the table to write to. May be fully qualified.

Table 207. sybasewrite options (continued)

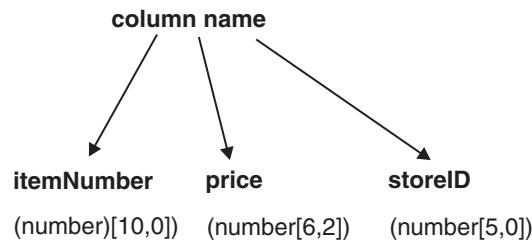
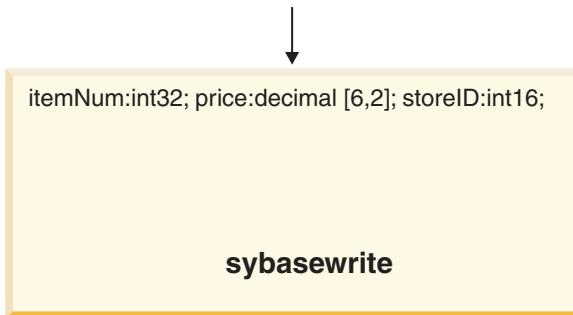
Option	Value
-mode	<p>append/create/replace/truncate</p> <p>Specify the mode for the write operator as one of the following:</p> <ul style="list-style-type: none"> • append: new records are appended into an existing table. • create: the operator creates a new table. If a table exists with the same name as the one you want to create, the step that contains the operator terminates with an error. The schema of the DataStage data set determines the schema of the new table. The table is created with simple default properties. To create a table that is partitioned, indexed, in a non-default table space, or in some other nonstandard way, you can use the -createtmt option with your own create table statement. • replace: The operator drops the existing table and creates a new one in its place. The schema of the DataStage data set determines the schema of the new table. • truncate: All records from an existing table are deleted before loading new records.
-drop	If this option is set unmatched fields of the DataStage data set will be dropped. An unmatched field is a field for which there is no identically named field in the datasource table.
-truncate	If this option is set column names are truncated to the maximum size allowed by the Sybase driver.
-truncateLength	n Specify the length to truncate column names to.
-open	opencommand Optionally specify a SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.
-close	closecommand Optionally specify a SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statements are executed only once on the conductor node.
-db_cs	db_cs Optionally specify the ICU code page, which represents the database character set in use. The default is ISO-8859-1.
-manual	Optionally specify whether you wish to manually load data.
-dumplocation	string Optionally specify the control file and dump location.
-string_length	string_length Optionally specify the length of a string. The value is an integer.

Example 1: Writing to an Existing Sybase Table

When an existing Sybase table is written to:

- The column names and data types of the Sybase table determine the input interface schema of the write operator.
- This input interface schema then determines the fields of the input data set that is written to the table.

For example, the following figure shows the sybasewrite operator writing to an existing table:



The record schema of the WebSphere DataStage data set and the row schema of the Sybase table correspond to one another, and field and column names are identical. Here are the input WebSphere DataStage record schema and output Sybase row schema:

Table 208. comparison of record schema and row schema

Input WebSphere DataStage Record	Output Sybase Table
itemNum:int32;	price Integer
price:decimal [3,2];	itemNum Decimal [3,2]
storeID:int16	storeID Small int

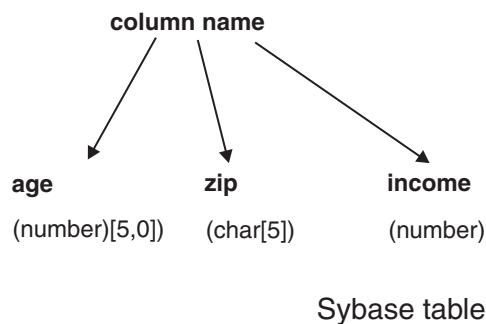
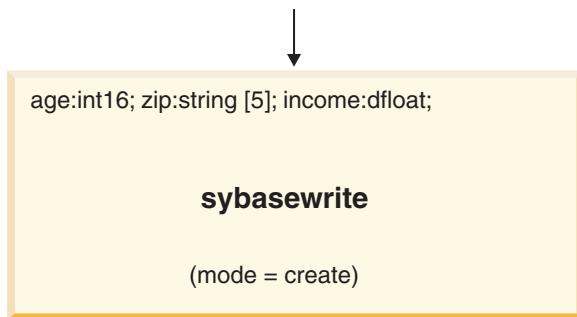
Here is the osh syntax for this example:

```
$ osh " ... op1 | sybasewrite -tablename 'table_2'
-db_name datasource name -user = user101 -password passwd
```

Note: that since the write mode defaults to append, the mode option does not appear in the command.

Example 2: Creating a Sybase Table

To create a table, specify a write mode of either create or replace. The next figure is a conceptual diagram of the create operation:



Here is the osh syntax for this operator:

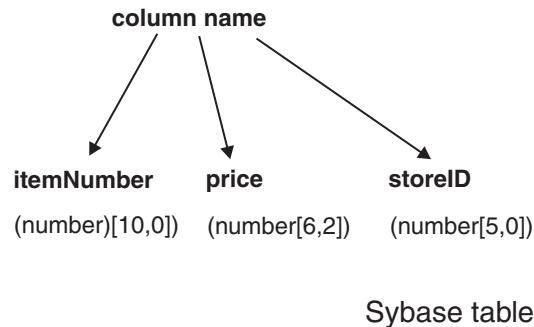
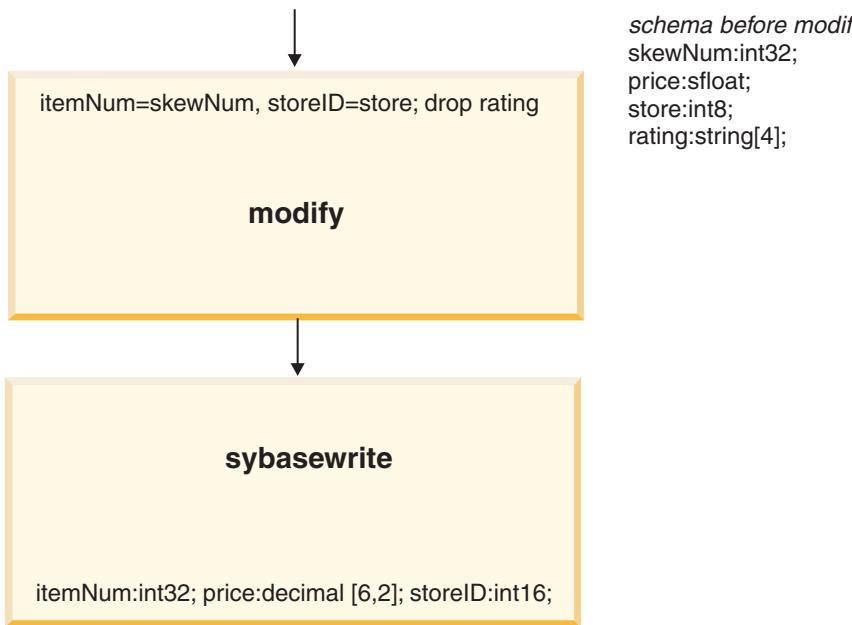
```
$ osh "... sybasewrite -table table_2
-mode create
-dboptions {'user = user101, password = userPword'} ..."
```

The sybasewrite operator creates the table, giving the Sybase columns the same names as the fields of the input WebSphere DataStage data set and converting the WebSphere DataStage data types to Sybase data types.

Example 3: Writing to a Sybase Table Using the modify Operator

The modify operator allows you to drop unwanted fields from the write operation and to translate the name or data type of a field of the input data set to match the input interface schema of the operator.

The next example uses the modify operator:



Sybase table

In this example, you use the modify operator to:

- Translate field names of the input data set to the names of corresponding fields of the operator's input interface schema, that is skewNum to itemNum and store to storeID.
- Drop the unmatched rating field, so that no error occurs.

Note: WebSphere DataStage performs automatic type conversion of store, promoting its int8 data type in the input data set to int16 in the sybasewrite input interface.

Here is the osh syntax for this operator:

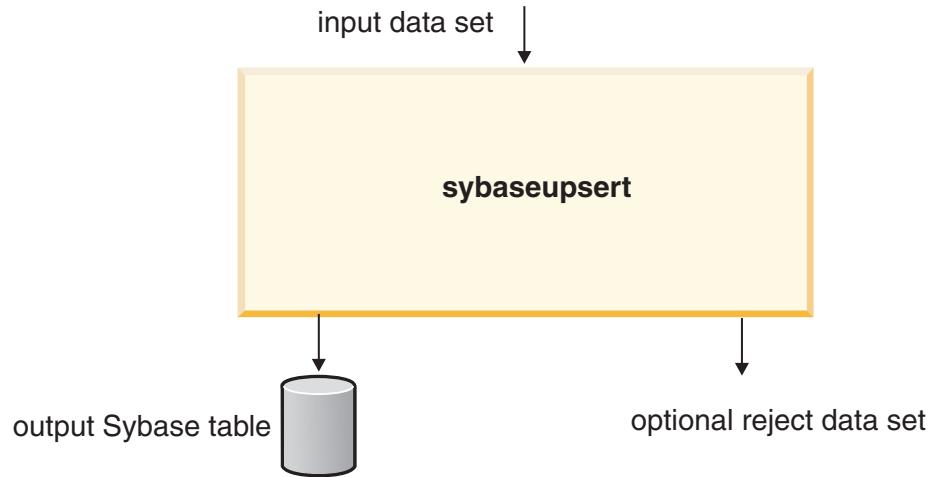
```
$ modifySpec="itemNum = skewNum, storeID = store;drop rating"
$ osh "... op1 | modify '$modifySpec'
| sybasewrite -table table_2
-dboptions {'user = user101, password =
UserPword'"}"
```

The **asesybaseupsert** and **sybaseupsert** Operators

The **asesybaseupsert** and **sybaseupsert** operators insert and update Sybase ASE and Sybase IQ tables respectively, with data contained in a DataStage data set. You provide the insert and update SQL statements using the **-insert** and **-update** options.

These operators receive a single data set as input and write its output to a Sybase table. You can request an optional output data set that contains the records that fail to be inserted or updated.

Data flow diagram



asesybaseupsert and sybaseupsert: properties

Table 209. asesybaseupsert and sybaseupsert Properties

Property	Value
Number of input data sets	1
Number of output data sets by default	None; 1 when you select the -reject option
Input interface schema	Derived from your insert and update statements
Transfer behavior	Rejected update records are transferred to an output data set when you select the -reject option
Execution mode	Parallel by default, or sequential
Partitioning method	Same You can override this partitioning method; however, a partitioning method of entire cannot be used.
Collection method	Any
Combinable operator	Yes

Operator Action

Here are the main characteristics of asesybaseupsert and sybaseupsert:

- If an -insert statement is included, the insert is executed first. Any records that fail to be inserted because of a unique-constraint violation are then used in the execution of the update statement.
- WebSphere DataStage uses host-array processing by default to enhance the performance of insert array processing. Each insert array is executed with a single SQL statement. Update records are processed individually.
- You use the -insertArraySize option to specify the size of the insert array. For example:
`-insertArraySize 250`

The default length of the insert array is 500. To direct WebSphere DataStage to process your insert statements individually, set the insert array size to 1:

```
-insertArraySize 1
```

- Your record fields can be variable-length strings. You can specify a maximum length or use the default maximum length of 80 characters. This example specifies a maximum length of 50 characters:

```
record (field1: string [max=50])
```

The maximum length in this example, by default is, 80 characters:

```
record (field1: string)
```

- When an insert statement is included and host array processing is specified, a WebSphere DataStage update field must also be a WebSphere DataStage insert field.
- The operators convert all values to strings before passing them to Sybase. The following WebSphere DataStage data types are supported:

- - int8, uint8, int16, uint16, int32, uint32, int64, and uint64

- - dfloat and sfloat

- - decimal

- - strings of fixed and variable length

- - timestamp

- - date

- By default, the operators produce no output data set. By using the -reject option, you can specify an optional output data set containing the records that fail to be inserted or updated. It's syntax is:

```
-reject filename
```

Asesybaseupsert and sybaseupsert: syntax and Options

The syntax for asesybaseupsert and sybaseupsert operators is shown below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

Syntax for asesybaseupsert:

```
-server          -- sybase server; exactly one occurrence required
    server      -- string
-db_name         -- use database; optional
    db_name     -- string
-user           -- sybase username; exactly one occurrence required
    username   -- string
-password        -- sybase password; exactly one occurrence required
    password   -- string
-update          -- Specify the update statement to be executed ; optional
    UpdateStatement -- string
-insert          -- Specify the Insert statement to be executed ; optional
    InsertStatement -- string
-delete          -- Specify the delete statement to be executed ; optional
    DeleteStatement -- string
-upsertmode      -- Upsert Mode of the operator ; optional
    insert_update/update_insert_delete_insert
                    -- string; value one of insert_update, update_insert, delete_insert
-reject          -- Records that fail to be update or inserted are written to a reject data set;
    optional
-db_cs           -- specifies database character set; optional
    db_cs       -- string
-insertarraysize -- array size; optional
    arraySize   -- integer; 1 or larger; default=1
-rowCommitInterval -- row commit interval; optional
    rowCommitInterval -- integer; 1 or larger; default=1
-open            -- Command run before opening a table ; optional
    opencommand -- string
-close           -- Command run after completion of processing a table ;optional
    closecommand -- string
```

Specify an ICU character set to map between Sybase char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to Sybase. The default character set is UTF-8, which is compatible with your osh jobs that contain 7-bit US-ASCII data. :

Table 210. sybaseupsert options

Options	value
-db_name	-db_name <i>database_name</i> Specify the data source to be used for all database connections. If no database is specified the default is used.
-server	-server <i>servername</i> Specify the server name to be used for all database connections. This option is mandatory.
-user	-user <i>user_name</i> Specify the user name used to connect to the data source.
-password	-password <i>password</i> Specify the password used to connect to the data source.
Statement options	The user must specify at least one of the following three options and no more than two. An error is generated if the user does not specify a statement option or specifies more than two.
-update	-update <i>update_statement</i> Optionally specify the update or delete statement to be executed.
-insert	-insert <i>insert_statement</i> Optionally specify the insert statement to be executed.
-delete	-delete <i>delete_statement</i> Optionally specify the delete statement to be executed.
-mode	-mode insert_update update_insert delete_insert Specify the upsert mode to be used when two statement options are specified. If only one statement option is specified the upsert mode will be ignored. <ul style="list-style-type: none"> • insert_update - The insert statement is executed first. If the insert fails due to a duplicate key violation (that is, record exists), the update statement is executed. This is the default upsert mode. • update_insert - The update statement is executed first. If the update fails because the record doesn't exist, the insert statement is executed. • delete_insert - The delete statement is executed first. Then the insert statement is executed.
-reject	-reject If this option is set, records that fail to be updated or inserted are written to a reject data set. You must designate an output data set for this purpose. If this option is not specified an error is generated if records fail to update or insert.

Table 210. *sybaseupsert options (continued)*

Options	value
-open	-open <i>open_command</i> Optionally specify a SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.
-close	-close <i>close_command</i> Optionally specify a SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statements are executed only once on the conductor node.
-insertarraysize	<i>n</i> Optionally specify the size of the insert/update array. The default size is 2000 records.

Example

This example updates a Sybase table that has two columns: acct_id and acct_balance, where acct_id is the primary key

Two of the records cannot be inserted because of unique key constraints; instead, they are used to update existing records. One record is transferred to the reject data set because its acct_id generates an -error

Summarized below is the state of the Sybase table before the dataflow is run, the contents of the input file, and the action WebSphere DataStage performs for each record in the input file.

Table 211. *Sybase table*

Table before dataflow	Input file contents	WebSphere DataStage action
Acct_id acct_balance	873092 67.23	Update
073587 45.64	865544 8569.23	Insert
873092 2001.89	566678 2008.56	Update
675066 3523.62	678888 7888.23	Insert
566678 89.72	073587 82.56	Update
	995666 75.72	Insert

The osh code for the example is:

```
$ osh "import -schema record (acct_id: string [6]; acct_balance: dfloat;)
-file input.txt |
hash -key acct_id |
tsort -key acct_id |
sybaseupsert -db_name dsn -user apt -password test
-insert 'insert into accounts
values (orchestrate.acct_id,
        orchestrate.acct_balance)'
-update 'update accounts
set acct_balance = orchestrate.acct_balance
where acct_id = orchestrate.acct_id'
-reject '/user/home/reject/reject.ds'"
```

Table 212. Table after dataflow

acct_id	acct_balance
073587	82.56
873092	67.23
675066	3523.62
566678	2008.56
865544	8569.23
678888	7888.23
995666	75.72

The **asesybaselookup** and **sybaselookup** Operators

With **asesybaselookup** and **sybaselookup** operators, you can perform a join between one or more Sybase ASE/IQ tables and a WebSphere DataStage data set. The resulting output data is a WebSphere DataStage data set containing WebSphere DataStage and Sybase data.

You perform this join by specifying either a SQL SELECT statement, or by specifying one or more Sybase tables and one or more key fields on which to do the lookup.

These operators are particularly useful for sparse lookups, that is, where the WebSphere DataStage data set you are matching is much smaller than the Sybase table. If you expect to match 90% of your data, using the **sybasereade** and **lookup** operators is probably more efficient.

Because these operators can do lookups against more than one Sybase table, it is useful for joining multiple Sybase tables in one query.

The **-statement** option command corresponds to a SQL statement of this form:

```
select a, b, c from data.testtbl  
where  
Orchestrate.b = data.testtbl.c  
and  
Orchestrate.name = "Smith"
```

The **asesybaselookup** and **sybaselookup** operators replace each **Orchestrate.fieldname** with a field value, submit the statement containing the value to Sybase, and output a combination of Sybase and WebSphere DataStage data.

Alternatively, you can use the **-key/-table** options interface to specify one or more key fields and one or more Sybase tables. The following osh options specify two keys and a single table:

```
-key a -key b -table data.testtbl
```

You get the same result as you would by specifying:

```
select * from data.testtbl  
where  
Orchestrate.a = data.testtbl.a  
and  
Orchestrate.b = data.testtbl.b
```

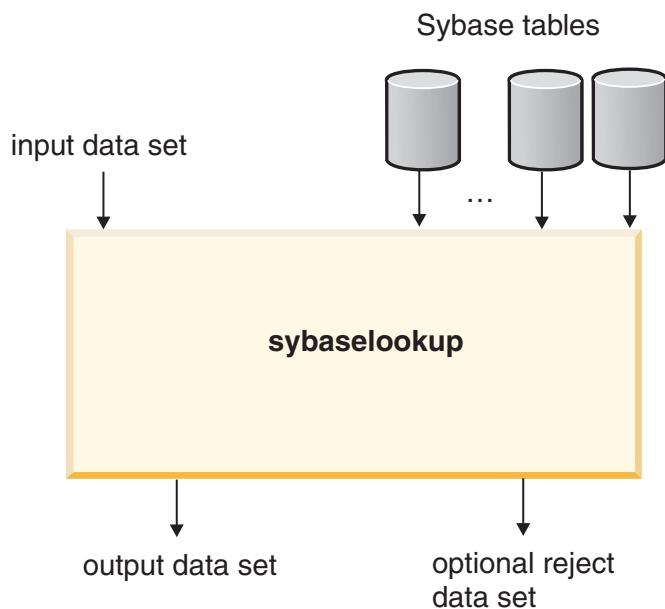
The resulting WebSphere DataStage output data set includes the WebSphere DataStage records and the corresponding rows from each referenced Sybase table. When a Sybase table has a column name that is the same as a WebSphere DataStage data set field name, the Sybase column is renamed using the following syntax:

APT_integer_fieldname

An example is APT_0_lname. The *integer* component is incremented when duplicate names are encountered in additional tables.

Note: If the Sybase ASE/IQ table is not indexed on the lookup keys, the performance of the asesybaselookup or sybaselookup operator is likely to be poor.

Data flow diagram



asesybaselookup and sybaselookup: properties

Table 213. asesybaselookup and sybaselookup Properties

Property	Value
Number of input data sets	1
Number of output data sets	1; 2 if you include the <code>-ifNotFound</code> reject option
Input interface schema	determined by the query
Output interface schema	determined by the sql query
Transfer behavior	transfers all fields from input to output
Execution mode	sequential or parallel (default)
Preserve-partitioning flag in output data set	Clear
Composite operator	no

asesybaselookup and sybaselookup: syntax and Options

The syntax for the asesybaselookup and sybaselookup operators are given below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

Syntax for asesybaselookup:

```

-server          -- sybase server; exactly one occurrence required
    server      -- String
-table           -- one of the lookup tables; any number of occurrences
    table       -- string
-key             -- specifies a lookup key: it is a name of a key for a dataset field and a
corresponding column of the database or a comma separated pair of key ids for a dataset field and
a database table ; any number of occurrences
    key         -- string
-db_name         -- use database; optional
    db_name     -- string
-user             -- sybase username; exactly one occurrence required
    username   -- string
-password          -- sybase password; exactly one occurrence required
    password   -- string
-query            -- SQL query to be used to read from table ; optional
    query      -- string
-db_cs            -- specifies database character set; optional
    db_cs      -- string
-selectlist        -- SQL select list that specifies which column in table are read ; used
with -table option ; optional
    list        -- string; default=all columns in table are read
-filter           -- conjunction to WHERE clause that specifies which rows are read from the
table ; used with -table option ; optional
    filter      -- string; default=all rows in table are read
-transfer_adapter -- Transfer adapter string for table fields ; optional
    transfer_adapter -- string
-open              -- Command run before opening a table ; optional
    open        -- string
-close             -- Command run after completion of processing table; optional
    close       -- string
-fetcharraysize   -- array size; optional
    arraySize   -- integer; 1 or larger; default=1
-ifNotFound        -- action to take for lookup failures; optional
    action      -- string; value one of fail, drop, reject, continue; default=fail

```

You must specify either the -query option or one or more -table options with one or more -key fields.

Table 214. asesybaselookup Options

Option	Value
-server	server Specify the server to connect to.
-db_name	db_name Specify the data source to be used for all database connections. This option is required.
-user	username Specify the user name used to connect to the data source. This option might not be required depending on the data source.
-password	password Specify the password used to connect to the data source. This option might not be required depending on the data source.
-table	table Specify a table and key fields to be used to generate a lookup query. This option is mutually exclusive with the -query option.

Table 214. *asesybaselookup Options (continued)*

Option	Value
-filter	filter Specify the rows of the table to exclude from the read operation. This predicate will be appended to the where clause of the SQL statement to be executed.
-selectlist	string Specify the list of column names that will appear in the select clause of the SQL statement to be executed.
-transfer_adapter	transfer_adapter Optionally specify the transfer adapter for table fields.
-ifNotFound	fail drop reject continue Specify an action to be taken when a lookup fails. Can be one of the following: <ul style="list-style-type: none">• fail: stop job execution• drop: drop failed record from the output data set• reject: put records that are not found into a reject data set. You must designate an output data set for this option• continue: leave all records in the output data set (outer join)
-query	query Specify a lookup SQL query to be executed. This option is mutually exclusive with the -table option.
-open	opencommand Optionally specify a SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.
-close	closecommand Optionally specify a SQL statement to be executed after the insertarray is processed. You cannot commit work using this option. The statement is executed only once on the conductor node.
-fetcharraysize	n Specify the number of rows to retrieve during each fetch operation. Default is 1.
-db_cs	db_cs Optionally specify the ICU code page, which represents the database character set in use. The default is ISO-8859-1.

Syntax for sybaselookup:

```

-server          -- sybase server; exactly one occurrence required
    server      -- string
-table           -- one of the lookup tables; any number of occurrences
    table       -- string
-key            -- specifies a lookup key: it is a name of a key for a dataset field and a

```

```

corresponding column of the database or a comma separated pair of key ids for a dataset field and a
database table ; any number of occurrences
    key          -- string
-db_name      -- use database; optional
    db_name     -- string
-user        -- sybase username; exactly one occurrence required
    username   -- string
-password     -- sybase password; exactly one occurrence required
    password   -- string
-query       -- SQL query to be used to read from table ; optional
    query     -- string
-db_cs        -- specifies database character set; optional
    db_cs      -- string
-selectlist   -- SQL select list that specifies which column in table are read ; used
with -table option ; optional
    list       -- string; default=all columns in table are read
-filter      -- conjunction to WHERE clause that specifies which rows are read from the
table ; used with -table option ; optional
    filter     -- string; default=all rows in table are read
-transfer_adapter
    transfer_adapter -- Transfer adapter string for table fields ; optional
    -- string
-open        -- Command run before opening a table ; optional
    opencommand -- string
-close        -- Command run after completion of processing table; optional
    closecommand -- string
-fetcharraysize
    arraySize   -- array size; optional
    -- integer; 1 or larger; default=1
-ifNotFound  -- action to take for lookup failures; optional
    action on failure -- string; value one of fail, drop, reject, continue; default=fail

```

You must specify either the -query option or one or more -table options with one or more -key fields.

Table 215. sybaselookup Options

Option	Value
-db_name	<p>-db_name <i>database_name</i></p> <p>Specify the data source to be used for all database connections. If you do not specify a database, the default is used.</p>
-user	<p>-user <i>user_name</i></p> <p>Specify the user name used to connect to the data source. This option might not be required depending on the data source</p>
-password	<p>-password <i>password</i></p> <p>Specify the password used to connect to the data source. This option might not be required depending on the data source</p>

Table 215. *sybaselookup Options (continued)*

Option	Value
-table	<p>-table <i>table_name</i></p> <p>Specify a table and key fields to be used to generate a lookup query. This option is mutually exclusive with the -query option. The -table option has 3 suboptions:</p> <ul style="list-style-type: none"> • -filter <i>where_predicate</i> Specify the rows of the table to exclude from the read operation. This predicate will be appended to the where clause of the SQL statement to be executed. • -selectlist <i>list</i> Specify the list of column names that will appear in the select clause of the SQL statement to be executed • -key <i>field</i> Specify a lookup key. A lookup key is a field in the table that will be used to join against a field of the same name in the WebSphere DataStage data set. The -key option can be specified more than once to specify more than one key field.
-ifNotFound	<p>-ifNotFound fail drop reject continue</p> <p>Specify an action to be taken when a lookup fails. Can be one of the following:fail: stop job execution</p> <ul style="list-style-type: none"> • drop: drop failed record from the output data set • reject: put records that are not found into a reject data set. You must designate an output data set for this option • continue: leave all records in the output data set (outer join)
-query	<p>-query <i>sql_query</i></p> <p>Specify a lookup query to be executed. This option is mutually exclusive with the -table option</p>
-open	<p>-open <i>open_command</i></p> <p>Optionally specify a SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.</p>
-close	<p>-close <i>close_command</i></p> <p>Optionally specify a SQL statement to be executed after the insertarray is processed. You cannot commit work using this option. The statement is executed only once on the conductor node.</p>
-fetcharraysize	<p>-fetcharraysize <i>n</i></p> <p>Specify the number of rows to retrieve during each fetch operation. Default is 1.</p>
-db_cs	<p>-db_cs <i>character_set</i></p> <p>Optionally specify the ICU code page, which represents the database character set in use. The default is ISO-8859-1. This option has the following sub option:</p>

Table 215. sybaselookup Options (continued)

Option	Value
-use_strings	<p>-use_strings</p> <p>If this option is set strings (instead of ustrings) will be generated in the DataStage schema.</p>

Example

Suppose you want to connect to the APT81 server as user user101, with the password test. You want to perform a lookup between a WebSphere DataStage data set and a table called target, on the key fields lname, fname, and DOB. You can configure the lookup operator in either of two ways to accomplish this.

Here is the osh command using the -table and -key options:

```
$ osh " sybaselookup - }  
-key lname -key fname -key DOB  
< data1.ds > data2.ds "
```

Here is the equivalent osh command using the -query option:

```
$ osh " sybaselookup  
-query 'select * from target  
where lname = orchestrate.lname  
and fname = orchestrate.fname  
and DOB = orchestrate.DOB'  
< data1.ds > data2.ds "
```

WebSphere DataStage prints the *lname*, *fname*, and *DOB* column names and values from the WebSphere DataStage input data set and also the *lname*, *fname*, and *DOB* column names and values from the Sybase table.

If a column name in the Sybase table has the same name as a WebSphere DataStage output data set schema fieldname, the printed output shows the column in the Sybase table renamed using this format:

APT_integer_fieldname

For example, lname might be renamed to *APT_0_lname*.

Chapter 21. The SQL Server interface library

WebSphere DataStage can share data with external SQL Server data sources.

Four WebSphere DataStage operators provide this access:

- The sqlsrvread operator reads records from an external data source table and places them in a WebSphere DataStage data set.
- The sqlsrvwrite operator sets up a connection to SQL Server and inserts records into a table. The operator takes a single input data set. The write mode of the operator determines how the records of a data set are inserted into the table.
- The sqlsrvrupsert operator lets you insert into or update a table with data contained in a WebSphere DataStage data set. You can match records based on field names and then update or insert those records.
- The sqlsrvrlookup operator lets you perform a join between a table and a WebSphere DataStage data set, with the resulting data output as a WebSphere DataStage data set.

Accessing SQL Server from WebSphere DataStage

This section assumes that WebSphere DataStage has been configured to access SQL Server.

UNIX

- Ensure the shared library path includes \$dshome/..branded_odbclib and set the ODBCINI environment variable to \$dshome/.odbc.ini.
- Start SQL Server.
- Add \$APT_ORCHHOME/branded_odbclib to your PATH and \$APT_ORCHHOME/branded_odbclib to your LIBPATH, LD_LIBRARY_PATH, or SHLIB_PATH.
- Check you can access SQL server using a valid user name and password.

Windows

- Create a DSN for the SQL Server database using the ODBC Datasource Administrator. Select Microsoft® SQL Server driver for a datasource which is to be used with sqlsrvwrite. Select the Data Direct SQL Server Wire driver datasource for use with sqlsrvread, sqlsrvrlookup or sqlsrvrupsert.
- Check you can access SQL server using a valid user name and password.

National Language Support

WebSphere DataStage's National Language Support (NLS) makes it possible for you to process data in international languages using Unicode character sets. NLS is built on IBM's International Components for Unicode (ICU). For more information, see *WebSphere DataStage NLS Guide* and access the ICU home page:

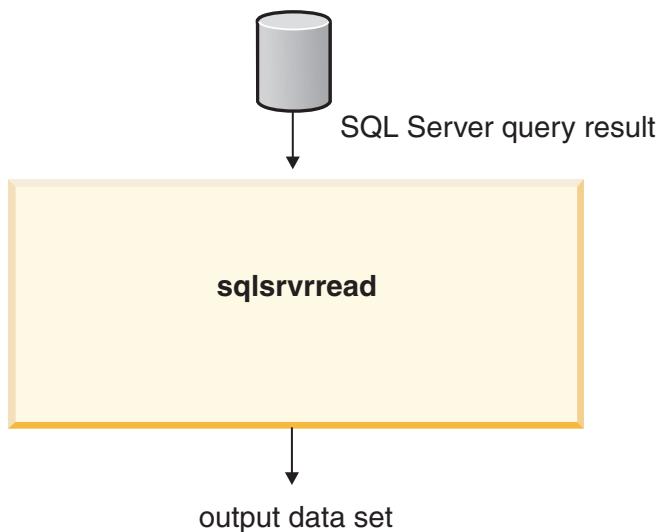
<http://oss.software.ibm.com/developerworks/opensource/icu/project>

The WebSphere DataStage SQL Server operators support Unicode character data in schema, table, and index names, user names and passwords, column names, table and column aliases, SQL statements, and file-name and directory paths.

The sqlsrvrread operator

The sqlsrvrread operator sets up a connection to a SQL Server table, uses an SQL query to request rows (records) from the table, and outputs the result as A WebSphere DataStage data set.

Data flow diagram



sqlsrvrread: properties

Table 216. *sqlsrvrread Operator Properties*

Property	Value
Number of input data sets	0
Number of output data sets	1
Input interface schema	None
Output interface schema	Determined by the SQL query
Transfer behavior	None
Execution mode	Sequential
Partitioning method	Not applicable
Collection method	Not applicable
Preserve-partitioning flag in output data set	Clear
Composite operator	No

Operator action

Below are the chief characteristics of the sqlsrvrread operator:

- You can direct it to run in specific node pools.
- It translates the query's result set (a two-dimensional array) row by row to a WebSphere DataStage data set.
- Its output is a WebSphere DataStage data set that you can use as input to a subsequent WebSphere DataStage operator.
- Its translation includes the conversion of SQL Server data types to WebSphere DataStage data types.

- The size of SQL Server rows can be greater than that of WebSphere DataStage records.
- The operator specifies whether to read a SQL Server table or to carry out an SQL query to carry out.
- It optionally specifies commands to be run before the read operation is performed and after it has completed the operation.
- You can perform a join operation between WebSphere DataStage data set and SQL Server (there might be one or more tables) data.

Where the `sqlsvrread` operator runs

The `sqlsvrread` operator runs sequentially.

Column name conversion

A SQL Server result set is defined by a collection of rows and columns. The `sqlsvrread` operator translates the query's result set (a two-dimensional array) to a WebSphere DataStage data set. Here is how a SQL Server query result set is converted to a data set:

- The schema of the SQL Server result set should match the schema of a WebSphere DataStage data set.
- The columns of a SQL Server row correspond to the fields of a WebSphere DataStage record. The name and data type of a SQL Server column correspond to the name and data type of a WebSphere DataStage field.
- Names are translated exactly except when the SQL Server column name contains a character that WebSphere DataStage does not support. In that case, the unsupported character is replaced by two underscore characters.
- Both SQL Server columns and WebSphere DataStage fields support nulls. A null contained in a SQL Server column is stored as a keyword "NULL" in the corresponding WebSphere DataStage field.

Data type conversion

The `sqlsvrread` operator converts SQL Server data types to WebSphere DataStage data types, as in the following table:

Table 217. Mapping of SQL Server datatypes to WebSphere DataStage datatypes

SQL Server Data Type	WebSphere DataStage Data Type
CHAR	string[n]
VARCHAR	string[max=n]
NCHAR	wstring(n)
NVARCHAR	wstring(max=n)
DECIMAL	decimal(p,s)
NUMERIC	decimal(p,s)
SMALLINT	int16
INTEGER	int32
REAL	sfloat
FLOAT	dfloat
BIT	uint8 (0 or 1)
TINYINT	uint8
BIGINT	int64
BINARY	raw(n)
VARBINARY	raw(max=n)

Table 217. Mapping of SQL Server datatypes to WebSphere DataStage datatypes (continued)

SQL Server Data Type	WebSphere DataStage Data Type
DATETIME	timestamp
SMALLDATETIME	timestamp
TIMESTAMP	raw(8)
UNIQUEIDENTIFIER	string[36]
SMALLMONEY	decimal[10,4]
MONEY	decimal[19,4]

Note: Datatypes that are not listed in this table generate an error.

SQL Server record size

The size of a WebSphere DataStage record is limited to 32 KB. However, SQL Server records can be longer than 32 KB. If you attempt to read a record longer than 32 KB, WebSphere DataStage returns an error and terminates your application.

Targeting the read operation

When reading a SQL Server table, you can either specify the table name that allows WebSphere DataStage to generate a default query that reads all the records of the table or you can explicitly specify the query.

Specifying the SQL Server table name

If you choose the -table option, WebSphere DataStage issues the following SQL SELECT statement to read the table:

```
select [selectlist]
      from table_name
      and (filter);
```

You can specify optional parameters to narrow the read operation. They are as follows:

- The selectlist parameter specifies the columns of the table to be read. By default, WebSphere DataStage reads all columns.
- The filter parameter specifies the rows of the table to exclude from the read operation. By default, WebSphere DataStage reads of all rows.

You can optionally specify an -open and -close option command. These commands are executed by sqlsvrread on SQL Server before the table is opened and after it is closed.

Specifying an SQL SELECT statement

If you choose the -query option, you pass an SQL query to the operator. The query selects from the table as it is read into WebSphere DataStage. The SQL statement can contain joins, views, database links, synonyms, and so on. However, the following restrictions apply to -query:

- The query might not contain bind variables.
- If you want to include a filter or select list, you must specify them as part of the query.
- The query runs sequentially.
- You can specify optional open and close commands. SQL Server runs these commands just before reading the data from the table and after reading the data from the table.

Join operations

You can perform a join operation between WebSphere DataStage data sets and SQL Server data. First invoke the sqlsrvrread operator and then invoke either the lookup operator (see [Lookup Operator](#)) or a join operator (see "The Join Library.")

Sqlsrvrread: syntax and options

The syntax for sqlsrvrread follows. The option values that you provide are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
sqlsrvrread
-query sql_query
-data_source data_source_name
-user user_name
-password password
-tableName table_name [-filter filter] [-selectlist list]
[-close close_command]
[-db_cs character_set [-use_strings]]
[-open open_command]
-fetcharraysize n
-isolation_level read_uncommitted | read_committed | repeatable_read | serializable
```

You must specify either the -query or -table option. You must also specify the -data_source, user, and password.

Table 218. sqlsrvrread operator options

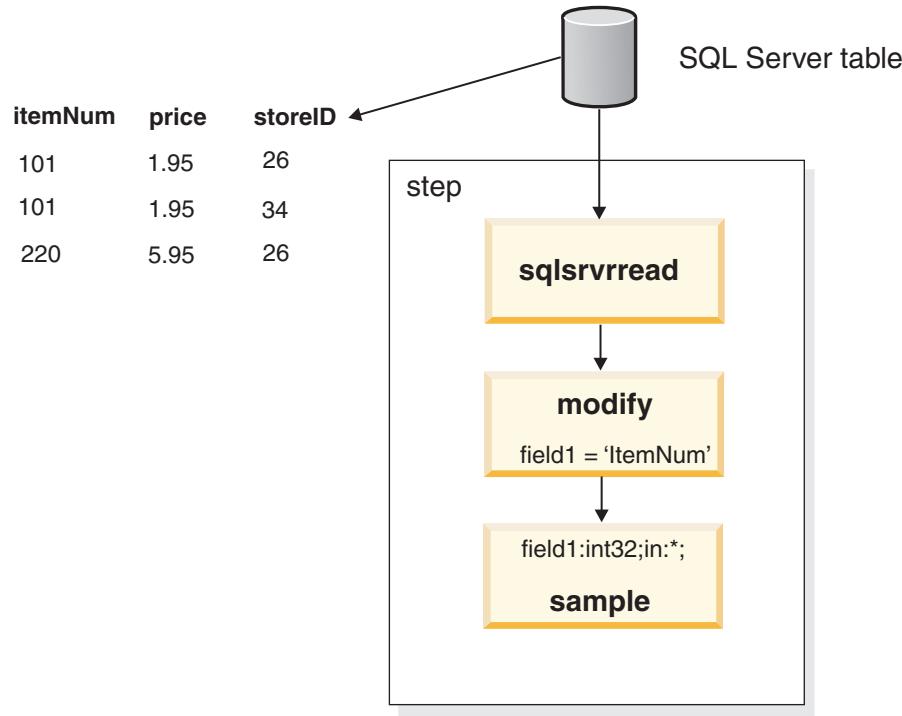
Option	Value
-data_source	<i>-data_source data_source_name</i> Specify the data source to be used for all database connections. This option is required.
-user	<i>-user user_name</i> Specify the user name used to connect to the data source. This option might or might not be required depending on the data source.
-password	<i>-password password</i> Specify the password used to connect to the data source. This option might or might not be required depending on the data source.
-tableName	<i>-tableName table_name</i> Specify the table to read. May be fully qualified. The -table option is mutually exclusive with the -query option. This option has 2 suboptions: <ul style="list-style-type: none">• -filter <i>where_predicate</i>. Optionally specify the rows of the table to exclude from the read operation. This predicate will be appended to the where clause of the SQL statement to be executed.• -selectlist <i>select_predicate</i>. Optionally specify the list of column names that will appear in the select clause of the SQL statement to be executed.

Table 218. *sqlsvrread* operator options (continued)

Option	Value
-open	<p>-open <i>open_command</i></p> <p>Optionally specify an SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.</p>
-close	<p>-close <i>close_command</i></p> <p>Optionally specify an SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statements are executed only once on the conductor node.</p>
-query	<p>-query <i>sql_query</i></p> <p>Specify an SQL query to read from one or more tables. The -query option is mutually exclusive with the -table option.</p>
-fetcharraysize	<p>-fetcharraysize <i>n</i></p> <p>Specify the number of rows to retrieve during each fetch operation. Default is 1.</p>
-isolation_level	<p>-isolation_level read_uncommitted read_committed repeatable_read serializable</p> <p>Optionally specify the isolation level for accessing data. The default isolation level is decided by the database or possibly specified in the data source.</p>
-db_cs	<p>-db_cs <i>character_set</i></p> <p>Optionally specify the ICU code page which represents the database character set in use. The default is ISO-8859-1. This option has the following suboption:</p> <ul style="list-style-type: none"> • -use_strings. If this option is set, strings will be generated in the WebSphere DataStage schema.

Sqlsrvrread example 1: Reading a SQL Server table and modifying a field name

The following figure shows a SQL Server table used as input to a WebSphere DataStage operator:



The SQL Server table contains three columns. The operator converts data in those columns as follows:

- itemNum of type INTEGER is converted to int32.
- price of type DECIMAL[6,2] is converted to decimal[6,2].
- storeID of type SMALLINT is converted to int16.

The schema of the WebSphere DataStage data set created from the table is also shown in this figure. Note that the WebSphere DataStage field names are the same as the column names of the SQL Server table.

However, the operator to which the data set is passed has an input interface schema containing the 32-bit integer field field1, while the data set created from the SQL Server table does not contain a field with the same name. For this reason, the modify operator must be placed between sqlsrvrread and sample operator to translate the name of the field, itemNum, to the name field1.

Here is the osh syntax for this example:

```
$ osh "sqlsrvrread -tablename 'table_1'  
-data_source datasource1  
-user user1  
-password user1  
| modify '$modifySpec' | ... "  
$ modifySpec="field1 = itemNum;"  
modify  
('field1 = itemNum,;')
```

The sqlsrvrwrite operator

The sqlsrvrwrite operator sets up a connection to SQL Server and inserts records into a table. The operator takes a single input data set. The write mode of the operator determines how the records of a data set are inserted into the table.

Writing to a multibyte database

Specifying chars and varchars

Specify chars and varchars in bytes, with two bytes for each character. This example specifies 10 characters:

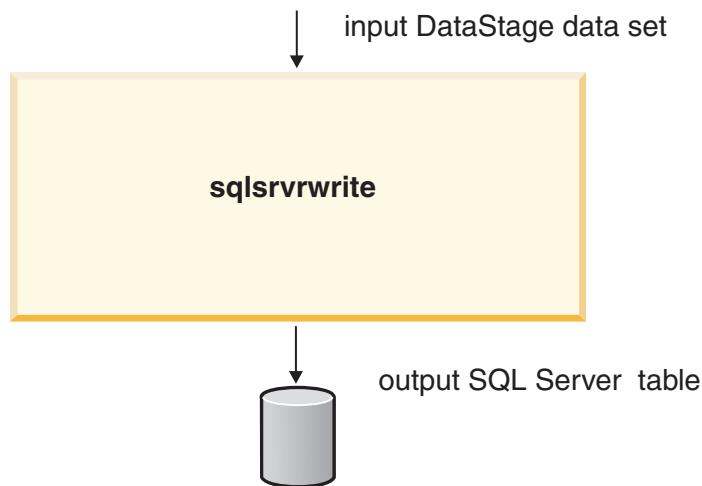
```
create table orch_data(col_a varchar(20));
```

Specifying nchar and nvarchar2 column size

Specify nchar and nvarchar columns in characters. For example, this example specifies 10 characters:

```
create table orch_data(col_a nvarchar(10));
```

Data flow diagram



sqlsrvrwrite: properties

Table 219. *sqlsrvrwrite Operator Properties*

Property	Value
Number of input data sets	1
Number of output data sets	0
Input interface schema	Derived from the input data set
Output interface schema	None
Transfer behavior	None
Execution mode	Sequential default or parallel
Partitioning method	Not applicable
Collection method	Any
Preserve-partitioning flag	Default clear

Table 219. *sqlsvrwrite* Operator Properties (continued)

Property	Value
Composite operator	No

Operator action

Here are the chief characteristics of the *sqlsvrwrite* operator:

- Translation includes the conversion of WebSphere DataStage data types to SQL Server data types.
- The operator appends records to an existing table, unless you set another writing mode.
- When you write to an existing table, the input data set schema must be compatible with the table's schema.
- Each instance of a parallel write operator running on a processing node writes its partition of the data set to the SQL Server table. You can optionally specify the SQL Server commands to be parsed and executed on all processing nodes before the write operation runs or after it completes.

Where the *sqlsvrwrite* operator runs

The default execution mode of the *sqlsvrwrite* operator is parallel. The default is that the number of processing nodes is based on the configuration file.

To direct the operator to run sequentially specify the [seq] framework argument.

You can optionally set the resource pool or a single node on which the operator runs.

Data conventions on write operations to SQL Server

SQL Server columns are named identically to WebSphere DataStage fields, with the following restrictions:

- SQL Server column names are limited to 128 characters. If a WebSphere DataStage field name is longer, you can do one of the following:
 - Choose the -truncate or truncate_length options to configure the operator to truncate WebSphere DataStage field names to the maximum length as the datasource column name. If you choose truncate_length, then you can specify the number of characters to be truncated. The number of characters should be less than the maximum length supported by SQL Server.
 - Use the modify operator to modify the WebSphere DataStage field name.
- There are types of data that cause an error if a WebSphere DataStage data set written to a SQL Server database contains them. If this happens the corresponding step terminates. However, WebSphere DataStage offers operators that modify certain data types to ones that SQL Server accepts, as shown in the table below.

Table 220. Mapping of WebSphere DataStage data types to SQLSRVR data types

WebSphere DataStage Data Type	SQL Server Data Type
string[n]	CHAR
string[max=n]	VARCHAR
ustring(n)	NCHAR
ustring(max=n)	NVARCHAR
decimal(p,s)	DECIMAL
decimal(p,s)	NUMERIC
int16	SMALLINT

Table 220. Mapping of WebSphere DataStage data types to SQLSRVR data types (continued)

WebSphere DataStage Data Type	SQL Server Data Type
int32	INTEGER
sfloat	REAL
dfloat	FLOAT
uint8 (0 or 1)	BIT
uint8	TINYINT
int64	BIGINT
raw(<i>n</i>)	BINARY
raw(max= <i>n</i>)	VARBINARY
timestamp[<i>p</i>]	DATETIME
timestamp[<i>p</i>]	SMALLDATETIME
string[36]	UNIQUEIDENTIFIER
raw(8)	TIMESTAMP
decimal[10,4]	SMALLMONEY
decimal[19,4]	MONEY

Write modes

The write mode of the operator determines how the records of the data set are inserted into the destination table. The write mode can have one of the following values:

- append: This is the default mode. The table must exist and the record schema of the data set must be compatible with the table. The write operator appends new rows to the table. The schema of the existing table determines the input interface of the operator.
- create: This operator creates a new table. If a table exists with the same name as the one you want to create, the step that contains the operator terminates with an error. The schema of the new table is determined by the schema of the WebSphere DataStage data set. The table is created with simple default properties. To create a table that is partitioned, indexed, in a non-default table space, or in some other non-standard way, you can use the -createtmt option with your own create table statement.
- replace: This operator drops the existing table and creates a new one in its place. If a table exists with the same name as the one you want to create, the existing table is overwritten. The schema of the new table is determined by the schema of the WebSphere DataStage data set.
- truncate: This operator retains the table attributes but discards existing records and appends new ones. The schema of the existing table determines the input interface of the operator. Each mode requires the specific user privileges shown in the table below:

Note: If a previous write operation fails, you can retry your job specifying a write mode of replace to delete any information in the output table that might have been written by the previous attempt to run your program.

Table 221. Required SQL Server privileges for write operators

Write Mode	Required Privileges
Append	INSERT on existing table
Create	TABLE CREATE
Replace	INSERT and TABLE CREATE on existing table
Truncate	INSERT on existing table

Matched and unmatched fields

The schema of the SQL Server table determines the operator's interface schema. Once the operator determines this, it applies the following rules to determine which data set fields are written to the table:

- Fields of the input data set are matched by name with fields in the input interface schema. WebSphere DataStage performs default data type conversions to match the input data set fields with the input interface schema.
- You can also use the modify operator to perform explicit data type conversions.
- If the input data set contains fields that do not have matching components in the table, the operator generates an error and terminates the step.
- WebSphere DataStage does not add new columns to an existing table if the data set contains fields that are not defined in the table. Note that you can use either the sqlsrvrwrite -drop option to drop extra fields from the data set. Columns in the SQL Server table that do not have corresponding fields in the input data set are set to their default value if one is specified in the SQL Server table. If no default value is defined for the SQL Server column and it supports nulls, it is set to null. Otherwise, WebSphere DataStage issues an error and terminates the step.
- WebSphere DataStage data sets support nullable fields. If you write a data set to an existing table and a field contains a null, the SQL Server column must also support nulls. If not, WebSphere DataStage issues an error message and terminates the step. However, you can use the modify operator to convert a null in an input field to another value.

Sqlsrvrwrite: syntax and options

Syntax for the sqlsrvrwrite operator is given below. Option values you supply are shown in italics. When your value contains a space or a tab character, you must enclose it in single quotes. Exactly one occurrence of the -tableName option is required.

```
sqlsrvrwrite


|                                                                                                                    |  |
|--------------------------------------------------------------------------------------------------------------------|--|
| -tableName <i>table_name</i>                                                                                       |  |
| -data_source <i>data_source_name</i>                                                                               |  |
| -user <i>user_name</i>                                                                                             |  |
| -password <i>password</i>                                                                                          |  |
| -mode append   create   replace   truncate                                                                         |  |
| [-createstmt <i>create_statement</i> ]                                                                             |  |
| [-drop]                                                                                                            |  |
| [-close <i>close_command</i> ]                                                                                     |  |
| [-open <i>open_command</i> ]                                                                                       |  |
| [-truncate]                                                                                                        |  |
| [-truncateLength <i>n</i> ]                                                                                        |  |
| [-db_cs <i>character_set</i> ]                                                                                     |  |
| [-batchsize <i>n</i> ]                                                                                             |  |
| [-isolation_level <i>read_uncommitted</i>   <i>read_committed</i>   <i>repeatable_read</i>   <i>serializable</i> ] |  |


:
```

Table 222. sqlsrvrwrite Operator options

Options	Value
-data_source	<p>-data_source <i>data_source_name</i></p> <p>Specify the data source to be used for all database connections. This option is required.</p>
-user	<p>-user <i>user_name</i></p> <p>Specify the user name used to connect to the data source. This option might or might not be required depending on the data source.</p>

Table 222. *sqlsvrwrite* Operator options (continued)

Options	Value
-password	<p>-password <i>password</i></p> <p>Specify the password used to connect to the data source. This option might or might not be required depending on the data source.</p>
-tableName	<p>-tableName <i>table_name</i></p> <p>Specify the table to which to write. May be fully qualified.</p>
-mode	<p>-mode append create replace truncate</p> <p>Specify the mode for the write operator as one of the following:</p> <ul style="list-style-type: none"> append: New records are appended into an existing table. create: A new table is created. If a table exists with the same name as the one you want to create, the step that contains the operator terminates with an error. The schema of the new table is determined by the schema of the WebSphere DataStage data set. The table is created with simple default properties. To create a table that is partitioned, indexed, in a non-default table space, or in some other non-standard way, you can use the -createstmt option with your own create table statement. replace: This operator drops the existing table and creates a new one in its place. The schema of the new table is determined by the schema of the WebSphere DataStage data set. truncate: All records from an existing table are deleted before loading new records.
-createstmt	<p>-createstmt <i>create_statement</i></p> <p>Optionally specify the create statement to be used for creating the table when -mode create is specified.</p>
-drop	If this option is set, unmatched fields of the WebSphere DataStage the data set will be dropped. An unmatched field is a field for which there is no identically named field in the datasource table.
-truncate	If this option is set, column names are truncated to the maximum size allowed by the SQL SERVER driver.
-truncateLength	<p>-truncateLength <i>n</i></p> <p>Specify the length to which to truncate column names.</p>
-open	-open <i>open_command</i>
-close	<p>-close <i>close_command</i></p> <p>Optionally specify an SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statements are executed only once on the conductor node.</p>

Table 222. sqlsrvrwrite Operator options (continued)

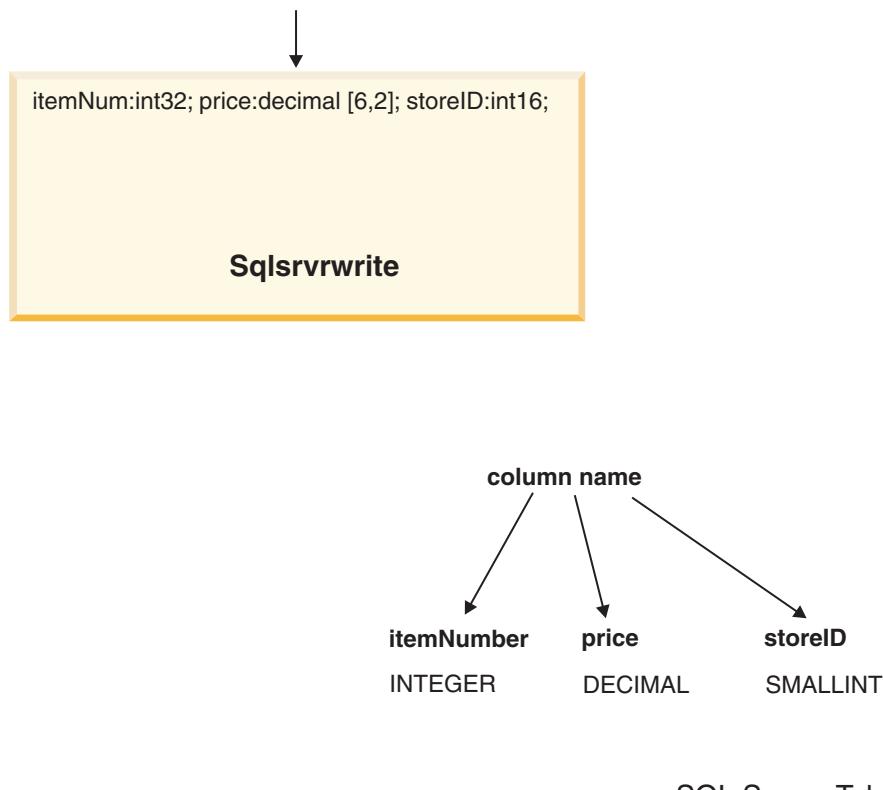
Options	Value
-batchsize	-batchsize <i>n</i> Optionally specify the number of records that should be committed before starting a new transaction. The default value is 1000 records.
-isolation_level	-isolation_level read_uncommitted read_committed repeatable_read serializable Optionally specify the isolation level for accessing data. The default isolation level is decided by the database or possibly specified in the datasource.
-db_cs	-db_cs <i>character_set</i> Optionally specify the ICU code page which represents the database character set in use. The default is ISO-8859-1.

Example 1: Writing to an existing SQL Server table

When an existing SQL Server table is written to:

- The column names and data types of the SQL Server table determine the input interface schema of the write operator.
- This input interface schema then determines the fields of the input data set that is written to the table.

For example, the following figure shows the sqlsrvrwrite operator writing to an existing table:



The record schema of the WebSphere DataStage data set and the row schema of the SQL Server table correspond to one another, and field and column names are identical. Here are the input WebSphere DataStage record schema and output SQL Server row schema:

I

Table 223. Schemas for Example 1

WebSphere DataStage	SQL Server
itemNum:int32;	itemNum INTEGER
price:decimal[6,2];	price DECIMAL[6,2]
storeID:int16	storeID SMALLINT

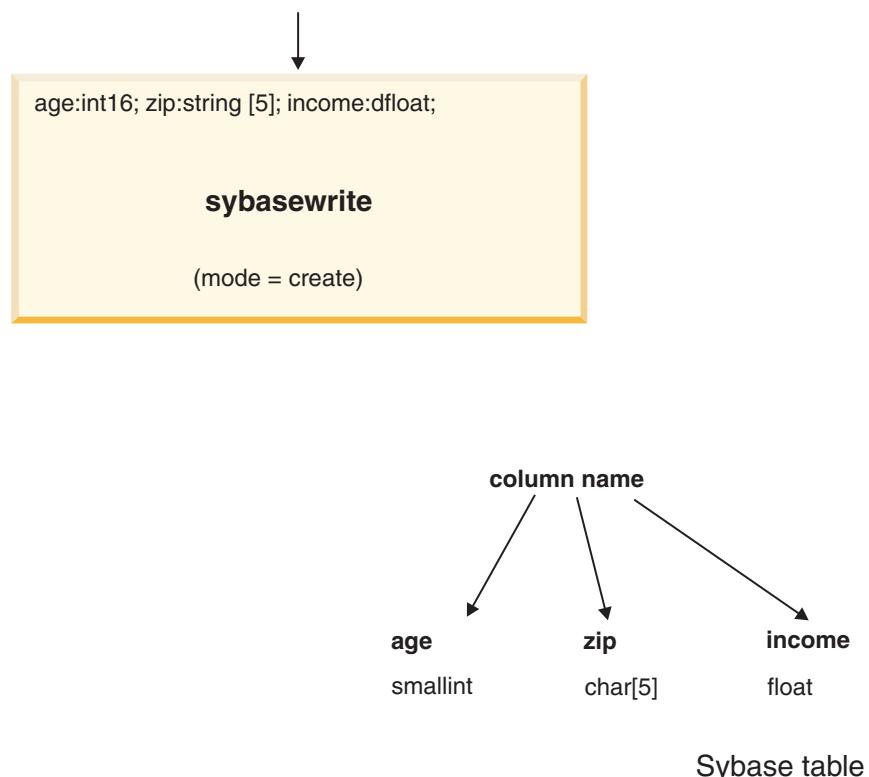
Here is the osh syntax for this example:

```
$ osh " ... op1 | sqlsrvrwrite -tablename 'table_2'  
-data_source datasourcename -user = user101 -password passwrd
```

Note: Since the write mode defaults to append, the mode option does not appear in the command.

Example 2: Creating a SQL Server table

To create a table, specify a write mode of either create or replace. The next figure is a conceptual diagram of the create operation:



Here is the osh syntax for this operator:

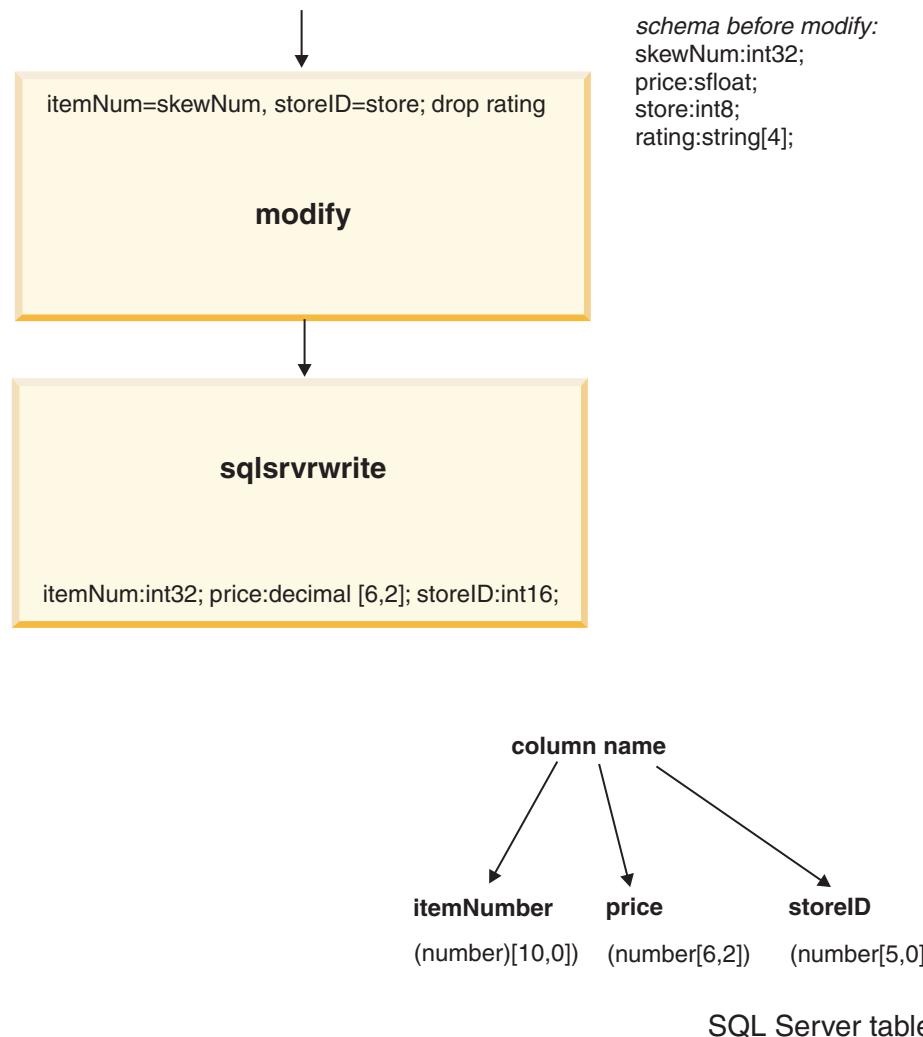
```
$ osh "... sqlsrvrwrite -table table_2  
-mode create  
-doptions {'user = user101, password = userPword'} ..."
```

The sqlsvrwrite operator creates the table, giving the SQL Server columns the same names as the fields of the input WebSphere DataStage data set and converting the WebSphere DataStage data types to SQL Server data types.

Example 3: Writing to a SQL Server table using the modify operator

The modify operator allows you to drop unwanted fields from the write operation and to translate the name or data type of a field of the input data set to match the input interface schema of the operator.

The next example uses the modify operator:



In this example, you use the modify operator to:

- Translate field names of the input data set to the names of corresponding fields of the operator's input interface schema, that is, skewNum to itemNum and store to storeID.
- Drop the unmatched rating field so that no error occurs.

Note that WebSphere DataStage performs automatic type conversion of store, promoting its int8 data type in the input data set to int16 in the sqlsvrwrite input interface.

Here is the osh syntax for this operator:

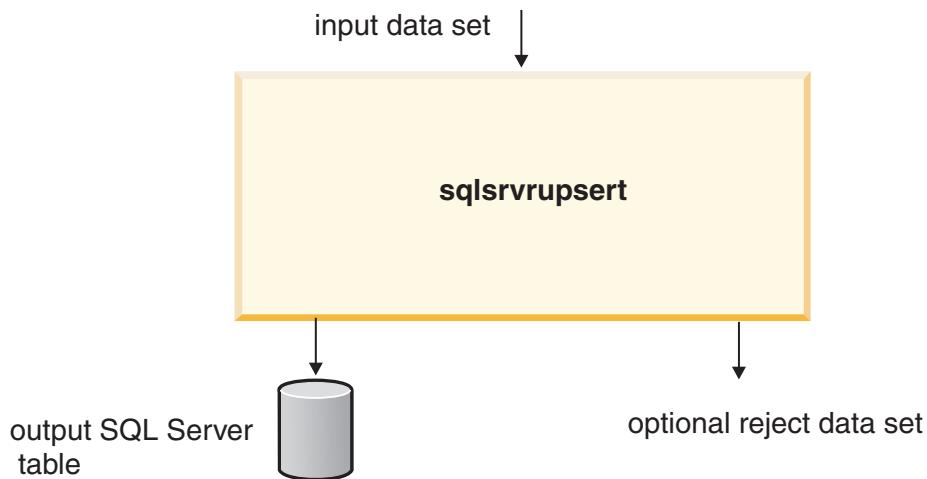
```
$ modifySpec="itemNum = skewNum, storeID = store;drop rating"
$ osh "... op1 | modify '$modifySpec'
| sqlsrvrwrite -table table_2
-user username -password passwd"
```

The sqlsrvrupsert operator

The sqlsrvrupsert operator inserts and updates SQL Server table records with data contained in a WebSphere DataStage data set. You provide the insert and update SQL statements using the -insert and -update options. By default, sqlsrvrupsert uses SQL Server host-array processing to optimize the performance of inserting records.

This operator receives a single data set as input and writes its output to a SQL Server table. You can request an optional output data set that contains the records that fail to be inserted or updated.

Data flow diagram



sqlsrvrupsert: properties

Table 224. sqlsrvrupsert Properties

Property	Value
Number of input data sets	1
Number of output data sets by default	None; 1 when you select the -reject option
Input interface schema	Derived from your insert and update statements
Transfer behavior	Rejected update records are transferred to an output data set when you select the -reject option.
Execution mode	Parallel by default, or sequential
Partitioning method	Same. You can override this partitioning method. However, a partitioning method of entire cannot be used.
Collection method	Any
Combinable operator	Yes

Operator action

Here are the main characteristics of sqlsrvrupsert:

- If a -insert statement is included, the insert is executed first. Any records that fail to be inserted because of a unique-constraint violation are then used in the execution of the update statement.
- WebSphere DataStage uses host-array processing by default to enhance the performance of insert array processing. Each insert array is executed with a single SQL statement. Update records are processed individually.
- You use the -insertArraySize option to specify the size of the insert array. For example, insertArraySize 250.
- The default length of the insert array is 500. To direct WebSphere DataStage to process your insert statements individually, set the insert array size to 1: -insertArraySize 1.
- Your record fields can be variable-length strings. You can specify a maximum length or use the default maximum length of 80 characters.
 - This example specifies a maximum length of 50 characters:
`record(field1:string[max=50])`
 - The maximum length in this example is, by default, 80 characters:
`record(field1:string)`
- When an insert statement is included and host array processing is specified, A WebSphere DataStage update field must also be A WebSphere DataStage insert field.
- The sqlsrvrupsert operator converts all values to strings before passing them to SQL Server. The following WebSphere DataStage data types are supported:
 - int8, uint8, int16, uint16, int32, uint32, int64, and uint64
 - dfloat and sfloat
 - decimal
 - strings of fixed and variable length
 - timestamp
 - date
- By default, sqlsrvrupsert produces no output data set. By using the -reject option, you can specify an optional output data set containing the records that fail to be inserted or updated. Its syntax is:
 - -reject *filename*

Sqlsrvrupsert: syntax and options

The syntax for sqlsrvrupsert is shown below. Option values you supply are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
sqlsrvrupsert
-data_source data_source_name
-user username
-password password
-mode insert_update | update_insert | delete_insert
-update update_statement |
-insert insert_statement |
-delete delete_statement
[-insertArraySize n]
[-reject]
[-open open_command]
[-close close_command]
[-rowCommitInterval n]
```

Specify an ICU character set to map between the SQL Server char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to SQL Server. The default character set is UTF-8 which is compatible with your osh jobs that contain 7-bit US-ASCII data.

Table 225. *sqlsvrupsert* Operator Options

Options	Value
-data_source	<p>-data_source <i>data_source_name</i></p> <p>Specify the data source to be used for all database connections. This option is required.</p>
-user	<p>-user <i>user_name</i></p> <p>Specify the user name used to connect to the data source. This option might or might not be required depending on the data source.</p>
-password	<p>-password <i>password</i></p> <p>Specify the password used to connect to the data source. This option might or might not be required depending on the data source.</p>
Statement options	The user must specify at least one of the options specified in the next three rows of this table. No more than two statements should be specified. An error is generated if the user does not specify a statement option or specifies more than two.
-update	<p>-update <i>update_statement</i></p> <p>Optionally specify the update or delete statement to be executed.</p>
-insert	<p>-insert <i>insert_statement</i></p> <p>Optionally specify the insert statement to be executed.</p>
-delete	<p>-delete <i>delete_statement</i></p> <p>Optionally specify the delete statement to be executed.</p>
-mode	<p>-mode <i>insert_update</i> <i>update_insert</i> <i>delete_insert</i></p> <p>Specify the upsert mode to be used when two statement options are specified. If only one statement option is specified, then the upsert mode will be ignored.</p> <p><i>insert_update</i>: The insert statement is executed first. If the insert fails due to a duplicate key violation (that is, if the record exists), the update statement is executed. This is the default upsert mode.</p> <p><i>update_insert</i>: The update statement is executed first. If the update fails because the record doesn't exist, the insert statement is executed.</p> <p><i>delete_insert</i>: The delete statement is executed first. Then the insert statement is executed.</p>
-reject	If this option is set, records that fail to be updated or inserted are written to a reject data set. You must designate an output data set for this purpose. If this option is not specified an error is generated if records fail to update or insert.

Table 225. *sqlsvrupsert* Operator Options (continued)

Options	Value
-open	-open <i>open_command</i> Optionally specify an SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.
-close	-close <i>close_command</i> Optionally specify an SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statements are executed only once on the conductor node.
-insertarraysize	-insertarraysize <i>n</i> Optionally specify the size of the insert/update array. The default size is 2000 records.
-rowCommitInterval	-rowCommitInterval <i>n</i> Optionally specify the number of records that should be committed before starting a new transaction. This option can only be specified if arraysizes = 1. Otherwise rowCommitInterval = arraysizes. This is because of the rollback logic/retry logic that occurs when an array execute fails.

Example

This example updates a SQL Server table that has two columns: acct_id and acct_balance, where acct_id is the primary key.

Two of the records cannot be inserted because of unique key constraints. Instead, they are used to update existing records. One record is transferred to the reject data set because its acct_id generates an error.

Summarized below are the state of the SQL Server table before the dataflow is run, the contents of the input file, and the action WebSphere DataStage performs for each record in the input file.

Table 226. Example Data

Table before dataflow	Input file contents	WebSphere DataStage Action		
acct_id	acct_balance	acct_id	acct_balance	
073587	45.64	873092	67.23	Update
873092	2001.89	865544	8569.23	Insert
675066	3523.62	566678	2008.56	Update
566678	89.72	678888	7888.23	Insert
		073587	82.56	Update
		995666	75.72	Insert

Table 227. Table after Upsert

acct_id	acct_balance
073587	82.56

Table 227. Table after Upsert (continued)

acct_id	acct_balance
873092	67.23
675066	3523.62
566678	2008.56
865544	8569.23
678888	7888.23
995666	75.72

The osh syntax for this example is:

```
$ osh "import -schema record(acct_id:string[6]; acct_balance:dfloat;)
-file input.txt |
hash -key acct_id |
tsort -key acct_id |
sqlsvrupsert -data_source dsn -user apt -password test
-insert 'insert into accounts
values(Orchestrate.acct_id,
orchestrate.acct_balance)'
-update 'update accounts
set acct_balance = Orchestrate.acct_balance
where acct_id = Orchestrate.acct_id'
-reject '/user/home/reject/reject.ds'"
```

The sqlsvrlookup operator

With the sqlsvrlookup operator, you can perform a join between one or more SQL Server tables and a WebSphere DataStage data set. The resulting output data is a WebSphere DataStage data set containing WebSphere DataStage and SQL Server data.

You perform this join by specifying either an SQL SELECT statement, or by specifying one or more SQL Server tables and one or more key fields on which to do the lookup.

This operator is particularly useful for sparse lookups, that is, where the WebSphere DataStage data set you are matching is much smaller than the SQL Server table. If you expect to match 90% of your data, using the sqlsvrread and lookup operators is probably more efficient.

Because sqlsvrlookup can do lookups against more than one eSQL Server table, it is useful for joining multiple SQL Server tables in one query.

The -statement option command corresponds to an SQL statement of this form:

```
select a,b,c from data.testtbl
where
Orchestrate.b = data.testtbl.c
and
Orchestrate.name = "Smith"
```

The operator replaces each Orchestrate.fieldname with a field value, submits the statement containing the value to SQL Server, and outputs a combination of SQL Server and SQL Server data.

Alternatively, you can use the -key/-table options interface to specify one or more key fields and one or more SQL Server tables. The following osh options specify two keys and a single table:

```
-key a -key b -table data.testtbl
```

You get the same result as you would by specifying:

```

select * from data.testtbl
where
Orchestrate.a = data.testtbl.a
and
Orchestrate.b = data.testtbl.b

```

The resulting WebSphere DataStage output data set includes the WebSphere DataStage records and the corresponding rows from each referenced SQL Server table. When a SQL Server table has a column name that is the same as a WebSphere DataStage data set field name, the SQL Server column is renamed using the following syntax:

`APT_integer_fieldname`

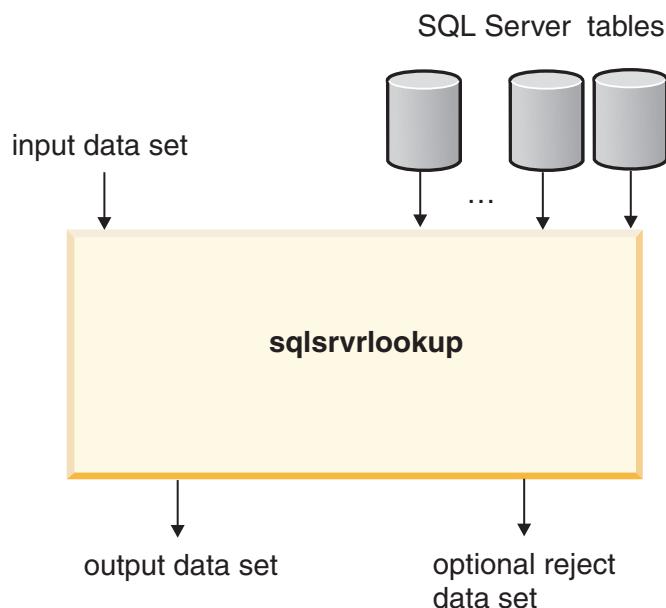
For example:

`APT_0_1name.`

The integer component is incremented when duplicate names are encountered in additional tables.

Note: If the SQL Server table is not indexed on the lookup keys, this operator's performance is likely to be poor.

Data flow diagram



sqlsrvrlookup: properties

Table 228. `sqlsrvrlookup` Operator Properties

Property	value
Number of input data sets	1
Number of output data sets	1; 2 if you include the <code>-ifNotFound</code> reject option
Input interface schema	determined by the query
Output interface schema	Determined by the sql query
Transfer behavior	transfers all fields from input to output
Execution mode	sequential or parallel (default)
Partitioning method	Not applicable

Table 228. *sqlsvrlookup* Operator Properties (continued)

Property	value
Collection method	Not applicable
Preserve-partitioning flag in output data set	Clear
Composite operator	no

Sqlsrvrlookup: syntax and options

The syntax for the sqlsrvrlookup operator is given below. The option values that you provide are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
sqlsvrlookup
-data_source data_source_name
-user username
-password password
-tableName table_name [-key field [-key field ...]
[-tableName table_name -key field [-key field ...]]
| -query sql_query
-ifNotFound fail | drop | reject | continue
[-open open_command]
[-close close_command]
[-fetcharraysize n]
[-db_cs character_set]
```

You must specify the -query option or one or more -table options with one or more -key fields.

The sqlsrvrlookup operator is a parallel operator by default. The options are as follows:

Table 229. *sqlsvrlookup* Operator Options

Options	Value
-data_source	-data_source <i>data_source_name</i> Specify the data source to be used for all database connections. This option is required.
-user	-user <i>user_name</i> Specify the user name used to connect to the data source. This option might or might not be required depending on the data source.
-password	-password <i>password</i> Specify the password used to connect to the data source. This option might or might not be required depending on the data source.

Table 229. *sqlsvrlookup* Operator Options (continued)

Options	Value
-tableName	<p>-tableName <i>table_name</i></p> <p>Specify a table and key fields to be used to generate a lookup query. This option is mutually exclusive with the -query option. The -table option has 2 suboptions:</p> <ul style="list-style-type: none"> • -filter <i>where_predicate</i>. Specify the rows of the table to exclude from the read operation. This predicate will be appended to the where clause of the SQL statement to be executed. • -selectlist <i>select_predicate</i>. Specify the list of column names that will appear in the select clause of the SQL statement to be executed.
-key	<p>-key <i>field</i></p> <p>Specify a lookup key. A lookup key is a field in the table that will be joined to a field of the same name in the WebSphere DataStage data set. The -key option can be specified more than once to specify more than one key field.</p>
-ifNotFound	<p>-ifNotFound fail drop reject continue</p> <p>Specify an action to be taken when a lookup fails. Can be one of the following:</p> <ul style="list-style-type: none"> • fail: stop job execution • drop: drop failed record from the output data set • reject: put records that are not found into a reject data set. You must designate an output data set for this option • continue: leave all records in the output data set (outer join)
-query	<p>-query <i>sql_query</i></p> <p>Specify a lookup query to be executed. This option is mutually exclusive with the -table option.</p>
-open	<p>-open <i>open_command</i></p> <p>Optionally specify an SQL statement to be executed before the insert array is processed. The statements are executed only once on the conductor node.</p>
-close	<p>-close <i>close_command</i></p> <p>Optionally specify an SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statement is executed only once on the conductor node.</p>
-fetcharraysize	<p>-fetcharraysize <i>n</i></p> <p>Specify the number of rows to retrieve during each fetch operation. Default is 1.</p>

Table 229. *sqlsvrlookup Operator Options (continued)*

Options	Value
-db_cs	<p>-db_cs <i>character_set</i></p> <p>Optionally specify the ICU code page which represents the database character set in use. The default is ISO-8859-1. This option has the following suboption:</p> <ul style="list-style-type: none"> • -use_strings. If this option is set strings (instead of ustrings) will be generated in the WebSphere DataStage schema.

Example

Suppose you want to connect to the APT81 server as user user101, with the password test. You want to perform a lookup between A WebSphere DataStage data set and a table called target, on the key fields lname, fname, and DOB. You can configure sqlsvrlookup in either of two ways to accomplish this.

- Here is the osh command using the -table and -key options:

```
$ osh " sqlsvrlookup - }
-key lname -key fname -key DOB
< data1.ds > data2.ds "
```

- Here is the equivalent osh command using the -query option:

```
$ osh " sqlsvrlookup
-query 'select * from target
where lname = Orchestrate.lname
and fname = Orchestrate.fname
and DOB = Orchestrate.DOB'
< data1.ds > data2.ds "
```

WebSphere DataStage prints the lname, fname, DOB column names, and values from the WebSphere DataStage input data set as well as the lname, fname, DOB column names, and values from the SQL Server table.

If a column name in the SQL Server table has the same name as A WebSphere DataStage output data set schema fieldname, the printed output shows the column in the SQL Server table renamed using this format:

APT_integer_fieldname

For example, lname might be renamed to:

APT_0_lname.

Chapter 22. The iWay interface library

WebSphere DataStage can read or lookup data on a data source that is accessed through an iWay server.

Two WebSphere DataStage operators provide this access:

- The `iwayread` operator reads records from an external data source table and places them in a WebSphere DataStage dataset.
- The `iwaylookup` operator lets you perform a join between a table and a WebSphere DataStage dataset or between a number of tables, with the resulting data output as a WebSphere DataStage dataset.

iWay middleware can be used to access a variety of data sources; its function is to insulate users from the complexity of accessing certain data sources, and is often used to access data from legacy databases on mainframe systems.

The operators connect to the iWay server using the iWay 5.3 API. They connect to and support all platforms and data sources supported by this API.

WebSphere DataStage does not provide facilities for writing or updating data sources through iWay.

Because iWay can potentially access a number of different data sources, fine tuning for reading a particular data source is done on the iWay connector rather than the WebSphere DataStage stage.

Accessing iWay from WebSphere DataStage

In order to use the iWay operators to connect to an iWay server you must have the iWay client (that is, the iWay connector) installed on the WebSphere DataStage server machine. If you have a cluster system, the iWay client should be installed on the conductor node.

National Language Support

WebSphere DataStage's National Language Support (NLS) makes it possible for you to process data in international languages using Unicode character sets. NLS is built on IBM's International Components for Unicode (ICU).

The WebSphere DataStage iWay operators support Unicode character data in schema, table, and index names, user names and passwords, column names, table and column aliases, SQL statements, and file-name and directory paths.

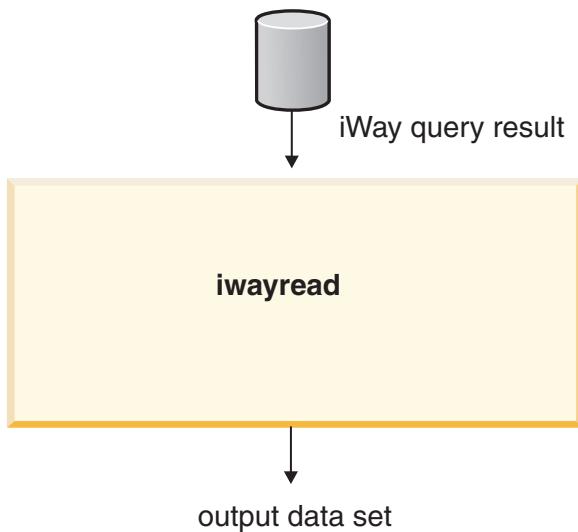
You can specify the character set that the data source you are accessing through iWay uses. All statements are converted to the data source character set before executing.

You can configure NLS at the iWay server level as well, using the iWay web console. See *WebSphere DataStage Connectivity Guide for iWay Servers* for more details.

The `iwayread` operator

The `iwayread` operator sets up a connection to an iWay server and uses an SQL query to request rows (records) from a table in a data source accessed via the iWay server. It outputs the result as a WebSphere DataStage data set.

Data flow diagram



iwayread: properties

Table 230. *iwayread Operator Properties*

Property	Value
Number of input data sets	0
Number of output data sets	1
Input interface schema	None
Output interface schema	outRec:*
Transfer behavior	The table columns are mapped to the WebSphere DataStage underlying data types and the output schema is generated for the data set.
Execution mode	Sequential
Partitioning method	Not applicable
Collection method	Not applicable
Preserve-partitioning flag in output data set	Not applicable
Composite operator	No

Operator action

Below are the chief characteristics of the iwayread operator:

- You can direct it to run in specific node pools.
- Its output is a WebSphere DataStage data set that you can use as input to a subsequent WebSphere DataStage operator.
- Its translation includes the conversion of iWay data types to WebSphere DataStage data types.

Data type conversion

When reading, the iWay Enterprise stage automatically converts iWay data types to WebSphere DataStage data types as shown in the following table:

Table 231. Data Type Conversion

WebSphere DataStage Data Type	iWay Data Type
int32	Integer
sfloat	Single Float
dfloat	Double Float
decimal <i>(m,n)</i>	Decimal <i>(m,n)</i>
string <i>[n]</i>	Alphanumeric (length= <i>n</i>)
raw	Binary
date	Date
string	Text
Not supported	Graphic (DBCS)
time	Time
timestamp	Timestamp

iwayread: syntax and options

The syntax for iwayread follows. The option values that you provide are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
iwayread
  -query sql_query [-query_type stmt_type]
  | -table table_name [-filter where_predicate]
    | [-selectlist select_predicate]
  [-timeout timeout]
  [-server iway_server]
  [-user user_name]
  [-password password]
  [-close close_command [-close_type stmt_type] [-close_timeout timeout]]
  [-open open_command [-open_type stmt_type] [-open_timeout timeout]]
  [-db_cs character_set [-use_strings]]
  [-data_password table_password]
  [-eda_params name=value, ...]
```

You must specify either the -query or -table option.

Table 232. iwayread operator options and its values

Option	Value
-server	<p>-server <i>server_name</i></p> <p>Specify the iWay server to be used for all data source connections. If this is not specified, the default server is used.</p>
-user	<p>-user <i>user_name</i></p> <p>Specify the user name used to connect to the iWay server. This is not required if the iWay server has security mode off.</p>
-password	<p>-password <i>password</i></p> <p>Specify the password used to connect to the iWay server. This is not required if the iWay server has security mode off.</p>

Table 232. iwayread operator options and its values (continued)

Option	Value
-query	<p>-query <i>sql_query</i></p> <p>Specify an SQL query to read from one or more tables. The -query option is mutually exclusive with the -table option. It has one suboption:</p> <ul style="list-style-type: none"> • -query_type <i>stmt_type</i>. Optionally specify the type of statement. Can be one of: SQL - a SQL query StoredProc - indicates that the statement specified is a FOCUS procedure CMD - a command The default is SQL.
-table	<p>-table <i>table_name</i></p> <p>Specify the table to read. May be fully qualified. The -table option is mutually exclusive with the -query option. This option has two suboptions:</p> <ul style="list-style-type: none"> • -filter <i>where_predicate</i>. Optionally specify the rows of the table to exclude from the read operation. This predicate will be appended to the where clause of the SQL statement to be executed. • -selectlist <i>select_predicate</i>. Optionally specify the list of column names that will appear in the select clause of the SQL statement to be executed.
-timeout	<p>-timeout <i>timeout</i></p> <p>Optionally specifies a timeout value (in seconds) for the statement specified witha -table or -query option. The default value is 0, which causes WebSphere DataStage to wait indefinitely for the statement to execute.</p>
-open	<p>-open <i>open_command</i></p> <p>Optionally specify an SQL statement to be executed before the read is carried out. The statements are executed only once on the conductor node. This option has two suboptions:</p> <ul style="list-style-type: none"> • -open_type <i>stmt_type</i>. Optionally specify the type of argument being supplied with -open. Can be one of: SQL - a SQL query StoredProc - indicates that the statement specified is a FOCUS procedure CMD - a command The default is SQL. • -open_timeout <i>timeout</i>. Optionally specify the timeout value (in seconds) for the statement specified with the -open option. The default is 0, which means WebSphere DataStage will wait indefinitely for the statement to execute.

Table 232. iwayread operator options and its values (continued)

Option	Value
-close	<p>-close <i>close_command</i></p> <p>Optionally specify an SQL statement to be executed after the read is carried out. You cannot commit work using this option. The statements are executed only once on the conductor node. This option has two suboptions:</p> <ul style="list-style-type: none"> • -close_type <i>stmt_type</i>. Optionally specify the type of argument being supplied with -close. Can be one of: SQL - a SQL query StoredProc - indicates that the statement specified is a FOCUS procedure CMD - a command The default is SQL. • -close_timeout <i>timeout</i>. Optionally specify the timeout value (in seconds) for the statement specified with the -close option. The default is 0, which means WebSphere DataStage will wait indefinitely for the statement to execute.
-db_cs	<p>-db_cs <i>code_page</i></p> <p>Optionally specify the ICU code page that represents the character set the database you are accessing through iWay is using. The default is ISO-8859-1. This option has a single suboption:</p> <ul style="list-style-type: none"> • -use_strings. Set this to have WebSphere DataStage generate strings instead of ustrings in the schema.
-data_password	<p>-data_password <i>data_password</i></p> <p>Optionally specify a table level password for accessing the required table.</p>
-eda_params	<p>-eda_params <i>name=value,... name=value</i></p> <p>Optionally specify values for the iWay environment variables as a list of semi-colon separated <i>name=value</i> pairs.</p>

Example: Reading a table via iWay

The example script connects to an iWay server called CUSU with username=vivek and password=vivek. It reads from the table stockfd accessed via iWay, and writes the records to the data set stock.ds:

```
$ osh "iwayread -server CUSU -user vivek -password vivek -table stockcd >| stock.ds"
```

The iwaylookup operator

With the iwaylookup operator, you can perform a join between tables accessed through iWay and a WebSphere DataStage data set. The resulting output data is a WebSphere DataStage data set containing WebSphere DataStage data and data from the tables accessed through iWay.

You perform this join by specifying either an SQL SELECT statement, or by specifying one or more SQL Server tables and one or more key fields on which to do the lookup.

This operator is particularly useful for sparse lookups, that is, where the WebSphere DataStage data set you are matching is much smaller than the table. If you expect to match 90% of your data, using the iwayread operator in conjunction with the lookup operator is probably more efficient.

Because iwaylookup can do lookups against more than one table accessed through iWay, it is useful for joining multiple tables in one query.

The -statement option command corresponds to an SQL statement of this form:

```
select a,b,c from data.testtbl  
where  
Orchestrate.b = data.testtbl.c  
Orchestrate.name = "Smith"
```

The operator replaces each `Orchestrate.fieldname` with a field value, submits the statement containing the value to the database accessed through iWay, and outputs a combination of the two tables.

Alternatively, you can use the -key/-table options interface to specify one or more key fields and one or more tables. The following osh options specify two keys and a single table:

```
-key a -key b -table data.testtbl
```

You get the same result as you would by specifying:

```
select * from data.testtbl  
where  
Orchestrate.a = data.testtbl.a  
and  
Orchestrate.b = data.testtbl.b
```

The resulting WebSphere DataStage output data set includes the WebSphere DataStage records and the corresponding rows from each referenced table. When a table has a column name that is the same as a WebSphere DataStage data set field name, the column is renamed using the following syntax:

`APT_integer_fieldname`

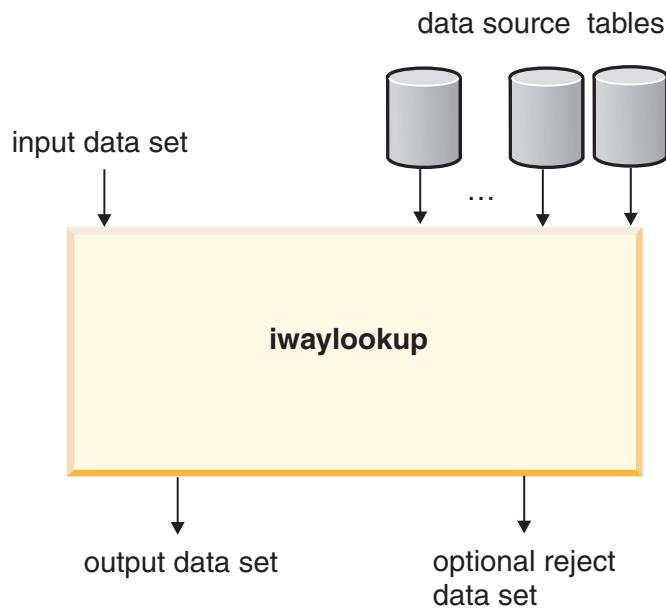
For example:

`APT_0_name.`

The integer component is incremented when duplicate names are encountered in additional tables.

Note: If the table is not indexed on the lookup keys, this operator's performance is likely to be poor.

Data flow diagram



iwaylookup: properties

Table 233. iwaylookup Operator Properties

Property	value
Number of input data sets	1
Number of output data sets	1; 2 if you include the -ifNotFound reject option
Input interface schema	key0:data_type;...keyN:data_type;inRec:*
Output interface schema (output data set)	outRec:/* with lookup fields missing from the input set concatenated
Output interface schema (reject data set)	rejectRec:*
Transfer behavior	transfers all fields from input to output
Execution mode	sequential or parallel (default)
Partitioning method	Not applicable
Collection method	Not applicable
Preserve-partitioning flag in output data set	Propagate
Composite operator	no

iwaylookup: syntax and options

The syntax for iwaylookup follows. The option values that you provide are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotes.

```
iwaylookup
  -query sql_query [-query_type stmt_type]
  |
  -table table_name [-filter where_predicate]
    [-selectlist select_predicate]
  [-timeout timeout]
  [-server iway_server]
  [-user user_name]
  [-password password]
```

```

[-close close_command [-close_type stmt_type] [-close_timeout timeout]]
[-open open_command [-open_type stmt_type] [-open_timeout timeout]]
[-db_cs character_set [-use_strings]]
[-data_password table_password]
[-eda_params name=value, ...]

```

You must specify either the -query or -table option.

The sqlsvrlookup operator is a parallel operator by default. The options are as follows:

Table 234. iwayread operator options and its values

Option	Value
-server	<p>-server <i>server_name</i></p> <p>Specify the iWay server to be used for all data source connections. If this is not specified, the default server is used.</p>
-user	<p>-user <i>user_name</i></p> <p>Specify the user name used to connect to the iWay server. This is not required if the iWay server has security mode off.</p>
-password	<p>-password <i>password</i></p> <p>Specify the password used to connect to the iWay server. This is not required if the iWay server has security mode off.</p>
-query	<p>-query <i>sql_query</i></p> <p>Specify an SQL query to read from one or more tables. The -query option is mutually exclusive with the -table option. It has one suboption:</p> <ul style="list-style-type: none"> • -query_type <i>stmt_type</i>. Optionally specify the type of statement. Can be one of: <ul style="list-style-type: none"> SQL - a SQL query StoredProc - indicates that the statement specified is a FOCUS procedure CMD - a command The default is SQL.
-table	<p>-table <i>table_name</i></p> <p>Specify the table to read. May be fully qualified. The -table option is mutually exclusive with the -query option. This option has two suboptions:</p> <ul style="list-style-type: none"> • -filter <i>where_predicate</i>. Optionally specify the rows of the table to exclude from the read operation. This predicate will be appended to the where clause of the SQL statement to be executed. • -selectlist <i>select_predicate</i>. Optionally specify the list of column names that will appear in the select clause of the SQL statement to be executed.

Table 234. iwayread operator options and its values (continued)

Option	Value
-IfNotFound	<p>-IfNotFound fail drop reject continue</p> <p>Specify an action to take if a lookup fails. This is one of the following:</p> <ul style="list-style-type: none"> • fail. Stop job execution. • drop. Drop failed record from output data set. • reject. Put records that are not found in the lookup table into a reject data set. (You must designate an output data set for this option.) • continue. leave all records in the output data set (that is, perform an outer join).
-timeout	<p>-timeout <i>timeout</i></p> <p>Optionally specifies a timeout value (in seconds) for the statement specified with a -table or -query option. The default value is 0, which causes WebSphere DataStage to wait indefinitely for the statement to execute.</p>
-open	<p>-open <i>open_command</i></p> <p>Optionally specify an SQL statement to be executed before the read is carried out. The statements are executed only once on the conductor node. This option has two suboptions:</p> <ul style="list-style-type: none"> • -open_type <i>stmt_type</i>. Optionally specify the type of argument being supplied with -open. Can be one of: SQL - a SQL query StoredProc - indicates that the statement specified is a FOCUS procedure CMD - a command The default is SQL. • -open_timeout <i>timeout</i>. Optionally specify the timeout value (in seconds) for the statement specified with the -open option. The default is 0, which means WebSphere DataStage will wait indefinitely for the statement to execute.
-close	<p>-close <i>close_command</i></p> <p>Optionally specify an SQL statement to be executed after the read is carried out. You cannot commit work using this option. The statements are executed only once on the conductor node. This option has two suboptions:</p> <ul style="list-style-type: none"> • -close_type <i>stmt_type</i>. Optionally specify the type of argument being supplied with -close. Can be one of: SQL - a SQL query StoredProc - indicates that the statement specified is a FOCUS procedure CMD - a command The default is SQL. • -close_timeout <i>timeout</i>. Optionally specify the timeout value (in seconds) for the statement specified with the -close option. The default is 0, which means WebSphere DataStage will wait indefinitely for the statement to execute.

Table 234. iwayread operator options and its values (continued)

Option	Value
-db_cs	<p>-db_cs <i>code_page</i></p> <p>Optionally specify the ICU code page that represents the character set the database you are accessing through iWay is using. The default is ISO-8859-1. This option has a single suboption:</p> <ul style="list-style-type: none"> • -use_strings. Set this to have WebSphere DataStage generate strings instead of ustrings in the schema.
-transfer_adapter	<p>-transfer_adapter <i>transfer_string</i></p> <p>Optionally specify new column names for the target data set (generally used when the table and data set to be renamed have the same column names).</p>
-data_password	<p>-data_password <i>data_password</i></p> <p>Optionally specify a table level password for accessing the required table.</p>
-eda_params	<p>-eda_params <i>name=value,... name=value</i></p> <p>Optionally specify values for the iWay environment variables as a list of semi-colon separated <i>name=value</i> pairs.</p>

Example: looking up a table via iWay

The example script connects to the iWay server CUSU using the username vivek and password vivek and performs a join between the table stockcf and the input data set stock.ds. It uses stock_symbol as the key.

This is the osh:

```
$ osh "iwaylookup -server CUSU -user vivek -password vivek -table stockcf -key stock_symbol < stocks.ds
>| output.ds"
```

The equivalent SQL for this operation is:

```
sewlect * from stockcf where stockcf.stock_symbol =input_dataset.stock_symbol
```

Chapter 23. The Netezza Interface Library

Netezza Enterprise stage is a database stage. You can use this stage to write bulk data to Netezza Performance Server 8000 (NPS). The Netezza Enterprise stage uses the Netezza write operator (nzwrite) to write data to Netezza Performance Server.

The Netezza write operator sets up a connection to an external data source and inserts records into a table. The operator takes a single input data set. The write mode determines how the records of a data set are inserted into the table. To understand the syntax and work with the Netezza write operator, you should be familiar with the Orchestrate (OSH) language.

Netezza write operator

The Netezza write operator writes data to Netezza Performance Server. Netezza does not provide a read operator and hence the data retrieved from the source database is saved in a WebSphere DataStage data set. The Netezza enterprise stage reads data from the data set and writes it to the Netezza Performance Server database.

Netezza data load methods

The Netezza write operator employs two alternate methods to load data to Netezza Performance Server. You can write data to a Netezza Performance Server database either by using a nzload load utility or by writing data to an external table before writing it to the database.

nzload method

You can use this load method if the data in the source database is consistent; that is, it implements a single character set for the entire database. Also the input schema for the nzload must be the same as that of the target table in the Netezza Performance Server database. The prerequisite to use the nzload method is that, nzclient utilities and ODBC functionality must be installed in the DataStage server.

The nzload utility provided by Netezza to load data into Netezza Performance Server. The retrieved data from the source database is fed to the nzload utility to load the destination database on the Netezza Performance Server.

External table method

If the data source that you want to read contains default values for table columns and uses variable format for data encoding such as UTF-8. You can write the data to an external table before loading it into the Netezza Performance Server database.

If you select the External table method, nzwrite creates an external table in the Netezza Performance Server. Data from the DataStage data set is written into the external table. From the external table, data is streamed to the destination table of the Netezza Performance Server. By default, Netezza Enterprise stage uses this load method.

Write modes

You can specify a write mode to determine how the records of the DataStage data set are inserted into the destination table in the Netezza Performance Server database.

append

Appends new rows to the specified table. To use this mode, you must have TABLE CREATE and

INSERT privileges on the database that is being written to. Also the table must exist and the record schema of the data set must be compatible with the table. This mode is the default mode.

create

Creates a new table in the database. To use this mode, you must have TABLE CREATE privileges. If a table already exists with the same name as the one that you want to create, the step that contains this mode ends in error. The table is created with simple default properties. To create a table that is partitioned, indexed, in a non-default table space, or to create a customized table, you must use the -createstmt option with your own create table statement.

replace

Drops the existing table and creates a new one in its place. To use this mode, you must have TABLE CREATE and TABLE DELETE privileges. If another table exists with the same name as the one that you want to create, the existing table is overwritten.

truncate

Retains all the attributes of a table (including its schema), but discards the existing records and appends new records into the table. To use this mode, you must have DELETE and INSERT privileges on that table.

Limitations of write operation

During the write operation, the rows in the source database are mapped and streamed into the destination database. The columns of the source database are not mapped individually. So you cannot interchange the order of the columns while writing to the destination database. The Netezza write operator expects the column format in the destination database to be identical to the schema of the source.

Character set limitations

Netezza Performance Server supports only ASCII table names and column names. The following character sets are supported:

- UTF-8 for NCHAR and NVARCHAR data types
- LATIN9 for CHAR and VARCHAR data types

Bad input records

Bad input records are the records that might be corrupted in the source database. If the Netezza write operator encounters such records during the write operation, the write operation will end in an error.

To ignore the corrupted records and continue the write operation, you can specify the number of times to ignore the corrupt records before the write operation stops. The Netezza write operator ignores the bad records and continues to write data into the Netezza Performance Server until the number of bad input records equals the number specified.

Error logs

You can view the error logs to identify errors that occur during any database operations, and view information about the success or failure of these operations. By default the log files are created under tmp in the root directory.

While writing data to the Netezza Performance Server by using the nzload method, log files such as nzlog and nzbad files are created in the /tmp directory on the client computer. The nzlog file is upended for every nzload command that loads the same table into the same database.

The nzbad files contains the records that caused the errors. The system overwrites this file each time you invoke the nzload command for the same table and database name. The following names are used for the log files:

- /tmp/*database name.table name.nzlog*

- /tmp/*database name.table name.nzbad*

While writing data to the Netezza Performance Server by using the External Table method, the log files are created in the /tmp directory in the Netezza Performance Server. The following names are used for the log files:

- /tmp/*external table name.log*
- /tmp/*external table name.bad*

Syntax for nzwrite operation

Syntax for the Netezza write operator is given below. The option values that you provide are shown in *italics*. When the specified value contains a space or a tab character, you must enclose it in single quotes. Following are the required options.

-dbserver <Server Name>

Specify the name of the Netezza server to connect to.

-data_source <data source name>

Specify the data source to be used for all database connections.

-table <tablename>

Specify the table name in the database to write data to. The table name can be fully qualified.

-loadmethod <ET/nzload>

Specify a load method.

-mode <append | create | replace | truncate>

Specify a write mode:

append

Appends new records into an existing table.

create

Creates a table.

replace

Drops the existing table and create a new table in its place. The schema of the new table is determined by the schema of the Orchestrate data set.

truncate

Deletes all the records from an existing table before new records are written.

Refer “Write modes” on page 783 for more information about these modes.

Following are the conditional Netezza write operator options:

-user <user name>

Specify the user name used to connect to the data source. This parameter is required if the data source requires a user name.

-password <password>

Specify the password used to connect to the data source. This parameter is required if the data source requires a password.

-dbname <database name>

Specify the name of the database to which to write. This option is applicable only when you are writing data using nzload method.

Following are the optional Netezza write operator options:

-delim <delimiter character>

Specify the delimiter character. The default delimiter is @ (at sign) character. You can use any ASCII character except blank and hyphen (-). The hyphen is the default delimiter for the date/time/timestamp parameter.

-open <open_command>

Specify an SQL statement to run after the insert array is processed. The statements are run only once on the conductor node.

-close <close_command>

Specify an SQL statement to run before the insert array is processed. You cannot commit work by using this option. The statements are executed only once on the conductor node.

-createstmt <create_statement>

Specify the CREATE SQL statement for creating the table when - mode create is specified. Use this command when you want to override the default write statements of the Netezza write operator.

-truncate

If you set this option, column names are truncated to the maximum size allowed by the ODBC driver.

-truncatelen <n>

You can specify the length of the column names to which they should be truncated when the data is written into the destination table. This option can be used only with the -truncate option.

-drop

If you set this option, the Netezza write operator drops unmatched input fields of the DataStage data set. An unmatched field is a field for which there is no identically named field in the destination table.

Example: syntax to replace a table

The syntax for nzwrite is as follows. The option values that you provide are shown in italic.

nzwrite

-loadmethod *nzload*

-mode *replace*

-server *NZSQL*

-dbname *tempdb*

-dbserver *netezza1*

-user *user1*

-password *password*

-table *test1*

Chapter 24. The Classic Federation interface library

Classic Federation enterprise stage contains operators that perform the database operations.

classicfedread

Reads records from an external data source table and places them in a WebSphere DataStage data set. To do this, the stage uses the classicfedread operator. You can perform a join operation between WebSphere DataStage data sets and external data. Use the classicfedread operator followed by the lookup operator or the join operator.

classicfedwrite

Sets up a connection to an external data source and inserts records into a table. The operator takes a single input data set. The write mode determines how the records of a data set are inserted into the table.

classicfedupsert

Inserts data into an external data source table or updates an external data source table with data in a WebSphere DataStage data set. You can match records that are based on field names and then update or insert those records.

classicfedlookup

Performs a join operation between an external data source table and a WebSphere DataStage data set, with the resulting data output as a WebSphere DataStage data set.

Accessing the federated database from WebSphere DataStage

DataStage must be configured in order to access federated database by using the federated database configuration process.

Access the federated database from WebSphere DataStage when the Classic Federation Enterprise stage is on a distributed DataStage server and the parallel engine shares the same federation settings.

Access the federated database using the following steps:

1. The DataDirect drivers must be installed in the directory \$dshome/../../branded_odbc. The shared library path is modified to include \$dshome/../../branded_odbc/lib. The Classic Federation environment variable will be set to \$dshome/.odbc.ini.
2. Start the external data source.
3. Add \$APT_ORCHHOME/branded_odbc to your PATH and \$APT_ORCHHOME/branded_odbc/lib to your LIBPATH, LD_LIBRARY_PATH, or SHLIB_PATH. The Classic Federation environment variable must be set to the full path of the **odbc.ini** file. The environment variable is operating system specific.
4. Access the external data source by using a valid user name and password.

National language support

National language support (NLS) in Classic Federation enterprise stage processes data in international languages with Unicode character sets. WebSphere DataStage uses International Components for Unicode (ICU) libraries to support National Language Support functionality. For information about National Language Support, see *WebSphere DataStage National Language Support Guide* and access the International Components for Unicode home page:

<http://www.ibm.com/software/globalization/icu>

Classic Federation enterprise stage operators support Unicode character data in these objects:

- Schema, table, and index names
- User names and passwords
- Column names
- Table and column aliases
- SQL*Net service names
- SQL statements
- File-name and directory paths.

International components for unicode character set parameter

The ICU character set parameters to optionally control character mapping.

The parameters which optionally control character mapping are as follows:

-db_cs icu_character_set

Specifies an ICU character set to map between federated CHAR and VARCHAR data and DataStage ustring data, and to map SQL statements for output to the federated database.

-nchar_cs icu_character_set

Specifies an ICU character set to map between federated NCHAR and NVARCHAR2 values and DataStage ustring data.

Mapping between federated and ICU character sets

To determine what federated character set corresponds to your ICU character-set specifications, WebSphere DataStage uses the table in \$APT_ORCHHOME/etc/ClassicFederation_cs.txt. The table is populated with some default values. Update this file if you use an ICU character set that has no entry in the file; otherwise UTF-8 is used. The Federated character set is specified in the sqldr control file as the CHARACTERSET option for loading your data.

Here is a sample ClassicFederation_cs.txt table:

Table 235. ClassicFederation_cs.txt table

ICU character set	Federated character set
UTF-8	UTF8
UTF-16	AL16UTF16
ASCL_ISO8859-1	WE8ISO8859P1
ISO-8859-1	WE8ISO8859P1
ASCL_MS1252	WE8ISO8859P1
EUC-JP	JA16EUC
ASCL-JPN-EUC	JA16EUC
ASCL_JPN-SJIS	JA16SJIS
Shift_JIS	JA16SJIS
US-ASCII	US7ASCII
ASCL-ASCII	US7ASCII

Read operations with classicfedread

Classic Federation enterprise stage reads records from the federated table datasource by using the classicfedread operator.

The classicfedread operation reads records from federated table by using an SQL query. These records are read into a WebSphere DataStage output data set.

The operator can perform both parallel and sequential database reads.

Here are the chief characteristics of the classicfedread operator:

- It translates the query result set (a two-dimensional array) row by row to a WebSphere DataStage data set.
- The output is a DataStage data set that can be used as input to a subsequent WebSphere DataStage operator.
- The size of external data source rows can be greater than that of WebSphere DataStage records.
- It either reads an external data source table or directs WebSphere DataStage to perform an SQL query.
- It optionally specifies commands to be run before the read operation is performed and after it completes.
- It can perform a join operation between a DataStage data set and an external data source. There can be one or more tables.

Related reference

“classicfedread: properties”

The read operations are performed with classicfedread operator properties

classicfedread: properties

The read operations are performed with classicfedread operator properties

Table 236. ClassicFedRead operator properties

Property	Value
Number of input data sets	0
Number of output data sets	1
Input interface schema	None
Output interface schema	Determined by the SQL query
Transfer behavior	None
Execution mode	Parallel and sequential
Partitioning method	MOD based
Collection method	Not applicable
Preserve-partitioning flag in the output data set	Clear
Composite stage	No

Classicfedread: syntax and options

classicfedread operator reads from the federated table.

The optional values that you provide are shown in italic typeface. When your value contains a space or a tab character, you must enclose it in single quotation marks.

```
classicfedread
]
[-close close_command]
  -datasource data_source_name
  [-db_cs icu_code_page [-use_strings]]
  [-fetcharraysize size
    [-isolation_level read_committed | read_uncommitted | repeatable_read | serializable]
    [-open open_command]
  [-partitionCol PartitionColumnName]
```

```

[-password password]
-query sql_query | -table table_name
  [-filter where_predicate]
  [-list select_predicate]
  [-username user_name]

```

You must specify either the `-query`, `-table` parameters, `-datasource` parameters. Specifying `-user`, and `-password` options are optional.

-close

Optionally specify an SQL statement to be run after the insert array is processed. You cannot commit work using this option. The statements are run only once on the conductor node.

-datasource

Specify the data source for all database connections. This option is required.

-db_cs

Optionally specify the ICU code page that represents the database character set. The default is ISO-8859-1.

-fetcharraysize

Specify the number of rows to retrieve during each fetch operation. The default number of rows is 1.

-isolation_level

Optionally specify the transaction level for accessing data. The default transaction level is decided by the database or possibly specified in the data source.

-open

Optionally specify an SQL statement to be run before the insert array is processed. The statements are run only once on the conductor node.

-partitioncol

Run in parallel mode. The default execution mode is sequential. Specify the key column; the data type of this column must be integer. The entry in this column should be unique. The column should preferably have the Monotonically Increasing Order property.

If you use the `-partitioncol` option, then you must follow the sample orchestrate shell scripts below to specify your `-query` and `-table` options.

-password

Specify the password used to connect to the data source. This option might or might not be required depending on the data source.

-query

Specify an SQL query to read from one or more tables. The `-query` and the `-table` option are mutually exclusive.

- Sample OSH for `-query` option:

```

classicfedread -data_source SQLServer -user sa -password asas
-query 'select * from SampleTable
where Col1 =2 and %orchmodColumn% ' -partitionCol Col1
>| OutDataSet.ds

```

-table

Specify the table to be read from. It might be fully qualified. This option has two suboptions:

- `-filter where_predicate`: Optionally specify the rows of the table to exclude from the read operation. This predicate is appended to the WHERE clause of the SQL statement to be run.
- `-list select_predicate`: Optionally specify the list of column names that appear in the SELECT clause of the SQL statement to be run. This option and the `-query` option are mutually exclusive.

- Sample OSH for `-table` option:

```
classicfedread -data_source SQLServer -user sa -password asas -table SampleTable  
-partitioncol Col1 > OutDataset.ds
```

-user

Specify the user name for connections to the data source. This option might be required depending on the data source.

-use_strings

Generate strings instead of ustrings in the WebSphere DataStage schema.

Column name conversion

A classicfedread operation is performed by converting an external data source query result set into a DataStage data set.

An external data source result set is defined by a collection of rows and columns. The classicfedread operator translates the query result set, to a WebSphere DataStage data set. The external data source query result set is converted to a WebSphere DataStage data set in the following way:

- The schema of an external data source result set must match the schema of a DataStage data set.
- The columns of an external data source row must correspond to the fields of a DataStage record. The name and data type of an external data source column must correspond to the name and data type of a DataStage field.
- The column names are used except when the external data source column name contains a character that DataStage does not support. In that case, two underscore characters _ _ replace the unsupported character.
- Both external data source columns and WebSphere DataStage fields support nulls. A null in an external data source column is stored as the keyword NULL in the corresponding DataStage field.

Data type conversion

The classicfedread operator converts external data source data types to orchestrate data types to successfully perform the read operation.

Data types that are not in the following table generate an error.

Table 237. Conversion of federation data types to OSH data types

Federated data type	DataStage data type
SQL_CHAR	STRING[n]
SQL_VARCHAR	STRING[max=n]
SQL_WCHAR	USTRING(n)
SQL_WVARCHAR	USTRING(max=n)
SQL_DECIMAL	DECIMAL(p,s)
SQL_NUMERIC	DECIMAL(p,s)
SQL_SMALLINT	INT16
SQL_INTEGER	INT32
SQL_REAL	DECIMAL(p,s)
SQL_FLOAT	DECIMAL(p,s)
SQL_DOUBLE	DECIMAL(p,s)
SQL_BIT	INT8 (0 or 1)
SQL_TINYINT	INT8
SQL_BIGINT	INT64
SQL_BINARY	RAW(n)

Table 237. Conversion of federation data types to OSH data types (continued)

Federated data type	DataStage data type
SQL_VARBINARY	RAW(max= <i>n</i>)
SQL_TYPE_DATE	DATE
SQL_TYPE_TIMEP	TIME[<i>p</i>]
SQL_TYPE_TIMESTAMP	TIMESTAMP[<i>p</i>]
SQL_GUID	STRING[36]

Reading external data source tables

You can read an external data source with the classicfedread operator with a query or only a table name.

To read an external data source table:

- You can specify the table name. WebSphere DataStage generates a default query that reads all records in the table.
- Explicitly specify the query.

Specifying the external data source table name

You can specify the external data source table name to read an external data source.

When you use the -table option, WebSphere DataStage issues this SELECT statement to read the table:

```
select [list]
from table_name and (filter);
```

Statements to narrow the read operation:

-list

Specifies the columns of the table to be read. By default, DataStage reads all columns.

-filter

Specifies the rows of the table to exclude from the read operation. By default, DataStage reads all rows.

You can optionally specify the -open and -close options. The open commands are run on the external data source before the table is open. The close commands are run on the external data source after it is closed.

Specifying an SQL SELECT statement

You can specify a SELECT statement to read an external data source.

When you use the -query option, an SQL query is passed to the classicfedread operator. The query does a select on the table as it is read into DataStage. The SQL statement contains joins, views, database links, synonyms, and so on. However, these restrictions apply to this option:

- It cannot contain bind variables.
- If you want to include the -filter or -list options, you must specify them as part of the query.
- The query runs sequentially.
- You can specify optional open and close commands.

Write operations with classicfedwrite

You can use the classicfedwrite operator to write data to an external data source.

The classicfedwrite operator sets up a connection to an external data source and inserts records into a table. The operator takes a single input data set. The write mode of the operator determines how the records of a data set are inserted into the table.

Below are the chief characteristics of the classicfedwrite operator:

- Translation includes the conversion of WebSphere DataStage data types to external data source data types.
- The classicfedwrite operator appends records to an existing table, unless you specify the create, replace, or truncate write mode.
- When you write to an existing table, the input data set schema must be compatible with the table schema.
- Each instance of a parallel write operator that runs on a processing node writes its partition of the data set to the external data source table. You can optionally specify external data source commands to be parsed and run on all processing nodes before the write operation runs or after it completes.

The default execution mode of the classicfedwrite operator is parallel. By default, the number of processing nodes is based on the configuration file.

To run sequentially, specify the [seq] argument. You can optionally set the resource pool or a single node on which the operator runs. You can set the resource pool or a single node using the config file.

Matched and unmatched fields

The schema of the external data source table determines the interface schema for the operator

When the operator determines the schema, it applies the following rules to determine which data set fields are written to the table:

1. Fields of the input data set are matched by name with fields in the input interface schema. WebSphere DataStage performs default data type conversions to match the input data set fields with the input interface schema.
2. You can also use the modify operator to perform explicit data type conversions. Modify operator is an external operator and alters the record schema of the input data set.
3. If the input data set contains fields that do not have matching components in the table, the operator generates an error and terminates the step.
4. WebSphere DataStage does not add new columns to an existing table if the data set contains fields that are not defined in the table. You can use the classicfedwrite -drop operation to drop extra fields from the data set. Columns in the external data source table that do not have corresponding fields in the input data set are set to their default value, if one is specified in the external data source table. If no default value is defined for the external data source column and if the column supports nulls, the column is set to null. Otherwise, WebSphere DataStage issues an error and terminates the step.
5. WebSphere DataStage data sets support nullable fields. If you write a data set to an existing table and a field contains a null, the external data source column must also support nulls. If not, WebSphere DataStage issues an error message and terminates the step. However, you can use the modify operator to convert a null in an input field to another value.

Classicfedwrite: syntax and options

classicfedwrite operator writes data to an external data source.

When your value contains a space or a tab character, you must enclose it in single quotation marks. Exactly one occurrence of the -table option is required.

```
classicfedwrite
  [-arraysize n]
  [-close close_command]
-datasourcename data_source_name
```

```
[-drop]
  [-db_cs icu_code_page]
  [-open open_command]
  [-password password]
  [-rowCommitInterval n]
[-mode create | replace | append | truncate]
  [-createstmt statement]
  -table table_name
    [-transactionLevels read_uncommitted | read_committed | repeatable read | serializable]
[-truncate]
  [-truncateLength n]
[-username user_name]
  [-useNchar]
```

-arraysize

Optionally specify the size of the insert array. The default size is 2000 records

-close

Optionally specify an SQL statement to be run after the insert array is processed. You cannot commit work by using this option. The statements are run only once on the conductor node

-datasourcename

Specify the data source for all database connections. This option is required.

-drop

Specify that unmatched fields in the WebSphere DataStage data set are dropped. An unmatched field is a field for which there is no identically named field in the data source table.

-db_cs

Optionally specify the ICU code page that represents the database character set in use. The default is ISO-8859-1.

-open

Optionally specify an SQL statement to be run before the insert array is processed. The statements are run only once on the conductor node.

-password

Specify the password for connections to the data source. This option might not be required depending on the data source.

-rowCommitInterval

Optionally specify the number of records to be committed before starting a new transaction. This option can be specified only if arraysize = 1. Otherwise, set the -rowCommitInterval parameter equal to the arrayszie parameter.

-mode

Specify the write mode as one of these modes:

append: This is the default mode. The table must exist, and the record schema of the data set must be compatible with the table. The classicfedwrite operator appends new rows to the table. The schema of the existing table determines the input interface of the operator.

create: The classicfedwrite operator creates a new table. If a table exists with the same name as the one being created, the step that contains the operator terminates with an error. The schema of the WebSphere DataStage data set determines the schema of the new table. The table is created with simple default properties. To create a table that is partitioned, indexed, in a non-default table space, or in some other non-standard way, you can use the -createstmt option with your own CREATE TABLE statement.

replace: The operator drops the existing table and creates a new one in its place. If a table exists with the same name as the one you want to create, the existing table is overwritten. The schema of the WebSphere DataStage data set determines the schema of the new table.

truncate: The operator retains the table attributes but discards existing records and appends new ones. The schema of the existing table determines the input interface of the operator. Each mode requires the specific user privileges shown in the table below. If a previous write operation failed, you can try again. Specify the replace write mode to delete any information in the output table that might exist from the previous attempt to run your program.

Table 238. External data source privileges for external data source writes

Write mode	Required privileges
Append	INSERT on existing tables
Create	TABLE CREATE
Replace	DROP, INSERT and TABLE CREATE on existing tables
Truncate	DELETE, INSERT on existing tables

-createtable

Optionally specify the CREATE TABLE statement to be used for creating the table when the -mode create parameter is specified.

-table

Specify the table to write to. The table name can be fully qualified.

-transactionLevels

Optionally specify the transaction level to access data. The default transaction level is decided by the database or possibly specified in the data source.

-truncate

Specify that column names are truncated to the maximum size that is allowed by the ODBC driver

-truncateLength

Specify the length to truncate column names.

-username

Specify the user name used for connections to the data source. This option might be required depending on the data source.

-useNchar

Read all nchars and nvarchar data types from the database.

classicfedwrite: properties

The write operations are performed with classicfedwrite operator properties

The table below gives the list of properties for classicfedwrite operator and their corresponding values.

Table 239. classicfedwrite operator properties

Property	Value
Number of input data sets	1
Number of output data sets	0
Input interface schema	Derived from the input data set
Output interface schema	None
Transfer behavior	None
Execution mode	Parallel by default or sequential
Partitioning method	Not applicable
Collection method	Any
Preserve-partitioning flag	The default is clear
Composite stage	No

Data conventions for write operations

External data source column names are identical to WebSphere DataStage field names, with the following restrictions:

- External data source column names are limited to 30 characters. If a WebSphere DataStage field name is longer, you can do one of the following:
 - Choose the -truncate or -truncateLength options to configure classicfedwrite to truncate WebSphere DataStage field names to the maximum length of the data source column name. If you choose -truncateLength, you can specify the number of characters to be truncate. The number must be less than the maximum length that the data source supports.
 - Use the modify operator to modify the field name. Modify operator is an external operator. The modify operator alters the record schema of the input data set.

Data type conversion

A data set written to an external data source cannot contain fields of certain data types. If the data set does, an error occurs and the corresponding step terminates.

WebSphere DataStage automatically modifies certain data types to those accepted by the external data source, as shown in the following table

Table 240. Mapping of osh data types to federated data types

WebSphere DataStage data type	Federated data type
string[n]	SQL_CHAR
string[max=n]	SQL_VARCHAR
wstring(n)	SQL_WCHAR
wstring(max=n)	SQL_WVARCHAR
decimal(p,s)	SQL_DECIMAL
decimal(p,s)	SQL_NUMERIC
int16	SQL_SMALLINT
int32	SQL_INTEGER
decimal(p,s)	SQL_REAL
decimal(p,s)	SQL_FLOAT
decimal(p,s)	SQL_DOUBLE
int8 (0 or 1)	SQL_BIT
int8	SQL_TINYINT
int64	SQL_BIGINT
raw(n)	SQL_BINARY
raw(max=n)	SQL_VARBINARY
date	SQL_TYPE_DATE
time[p]	SQL_TYPE_TIME
timestamp[p]	SQL_TYPE_TIMESTAMP
string[36]	SQL_GUID

This little command copies things.

Writing to multibyte databases

You can write data to a multibyte database with the classicfedread operator.

This can be done by:

1. Specifying char and varchar data types
2. Specifying nchar and nvarchar2 column sizes

Specifying char and varchar data types	Specifying nchar and nvarchar2 column sizes
Specify chars and varchars in bytes, with two bytes for each character. The following example specifies 10 characters: <code>create table orch_data(col_a varchar(20));</code>	Specify nchar and nvarchar2 columns in characters. The example below specifies 10 characters: <code>create table orch_data(col_a nvarchar2(10));</code>

Insert and update operations with classicfedupsert

The classicfedupsert operator inserts and updates external data source table records by data from a WebSphere DataStage data set. By default, classicfedupsert uses external data source host-array processing to optimize the performance of insertion records.

This operator receives a single data set as input and writes its output to an external data source table. You can request an optional output data set of records that fail to be inserted or updated.

The chief characteristics of the classicfedupsert operation are as follows:

- If an -insert_update statement is included, the insert is run first. Any records that fail to be inserted because of a unique-constraint violation are then used in the execution of the update statement. This is the default mode.
- By default, WebSphere DataStage uses host-array processing to enhance the performance of insert array processing. Each insert array is run with a single SQL statement. Updated records are processed individually.
- To process your insert statements individually, set the insert array size to 1
- The record fields can be of variable-length strings. You can either specify a maximum length or use the default maximum length of 80 characters.

This example specifies a maximum length of 50 characters:

```
record(field1:string[max=50])
```

The maximum length in this example below is by default 80 characters:

```
record(field1:string)
```

- When an insert statement is included and host array processing is specified, a WebSphere DataStage update field must also be a WebSphere DataStage insert field.
- The classicfedupsert operator converts all values to strings before the operation passes them to the external data source. The following OSH data types are supported:
 - int8, uint8, int16, uint16, int32, uint32, int64, and uint64
 - dfloat and sfloat
 - decimal
 - strings of fixed and variable length
 - timestamp
 - date

- By default, classicfedupsert does not produce an output data set. Using the -reject option, you can specify an optional output data set containing the records that fail to be inserted or updated. Its syntax is:

`-reject filename`

classicfedupsert: Properties

Table 241. classicfedupsert Properties and Values

Property	Value
Number of input data sets	1
Number of output data sets by default	None; 1 when you select the -reject option
Input interface schema	Derived from your insert and update statements
Transfer behavior	Rejected update records are transferred to an output data set when you select the -reject option
Execution mode	Parallel by default, or sequential
Partitioning method	Same You can override this partitioning method. However, a partitioning method of entire cannot be used. The partitioning method can be overridden using the GUI.
Collection method	Any
Combinable stage	Yes

Classicfedupsert: syntax and options

Inserts and updates table records on external data sources with data from WebSphere DataStage data set

When your value contains a space or a tab character, you must enclose it in single quotation marks.

```
classicfedupsert
[-close close_command]
-datasourcename data_source_name
[-db_cs icu_code_page]
-delete
-insert insert_statement |
[-insertArraySizen]
-mode
[-open open_command]
[-password password]
[-reject]
[-rowCommitInterval n]
-update update_statement |
[-username user_name]
```

Exactly one occurrence of the -update option is required. All others are optional.

Specify an International components for unicode character set to map between external data source char and varchar data and WebSphere DataStage ustring data, and to map SQL statements for output to an external data source. The default character set is UTF-8, which is compatible with the orchestrate shell jobs that contain 7-bit US-ASCII data.

-close

Optionally specify an SQL statement to be run after the insert array is processed. You cannot commit work using this option. The statements are run only once on the conductor node.

-datasourcename

Specify the data source for all database connections. This option is required.

-db_cs

Optionally specify the ICU code page that represents the database character set in use. The default is ISO-8859-1.

-delete

Optionally specify the delete statement to be run.

-insert

Optionally specify the insert statement to be run.

-update

Optionally specify the update or delete statement to be run.

insertArraySize

Optionally specify the size of the insert and update array. The default size is 2000 records.

-mode

Specify the upsert mode when two statement options are specified. If only one statement option is specified, the upsert mode is ignored.

insert_update

The insert statement is run first. If the insert fails due to a duplicate key violation (if the record being inserted exists), the update statement is run. This is the default mode.

update_insert

The update statement is run first. If the update fails because the record does not exist, the insert statement is run

delete_insert

The delete statement is run first; then the insert statement is run.

-open

Optionally specify an SQL statement to be run before the insert array is processed. The statements are run only once on the conductor node.

-password

Specify the password for connections to the data source. This option might not be required depending on the data source.

-reject

Specify that records that fail to be updated or inserted are written to a reject data set. You must designate an output data set for this purpose. If this option is not specified, an error is generated if records fail to update or insert.

-rowCommitInterval

Optionally specify the number of records to be committed before starting a new transaction. This option can be specified only if arrayszie = 1. Otherwise set the rowCommitInterval parameter to equal the insertArraySize.

-username

Specify the user name used to connect to the data source. This option might or might not be required depending on the data source.

Example of a federated table when a classicfedupsert operation is performed

This example shows the update of a federated table that has two columns named acct_id and acct_balance, where acct_id is the primary key. Two of the records cannot be inserted because of unique key constraints. Instead, they are used to update existing records. One record is transferred to the reject data set because its acct_id generates an error.

The following table shows the original data in the federated table before classicfedupsert runs. Compare table 1 with the table 2 to see whether the insert or update actions are run on the input data. Finally, look at table 3 to see what the data looks like after classicfedupsert runs.

Table 242. Original data in the federated table before classicfedupsert runs

acct_id	acct_balance
073587	45.64
873092	2001.89
675066	3523.62
566678	89.72

Table 243. Contents of the input file and their insert or update actions

acc_id(primary key)	acct_balance	classicfedupsert action
073587	45.64	Update
873092	2001.89	Insert
675066	3523.62	Update
566678	89.72	Insert

Table 244. Example data in the federated table after classicfedupsert runs

acct_id	acct_balance
073587	82.56
873092	67.23
675066	3523.62
566678	2008.56
865544	8569.23
678888	7888.23
995666	75.72

This Orchestrate shell syntax is an example of how to specify insert and update operations:

```
osh "import
  -schema record(acct_id:string[6] acct_balance:dfloat;)
  -file input.txt |
  hash -key acct_id |
  tsort -key acct_id |
  classicfedupsert -datasourcename dsn -username apt -password test
  -insert 'insert into accounts
    values(ORCHESTRATE.acct_id, ORCHESTRATE.acct_balance)'
  -update 'update accounts
    set acct_balance = ORCHESTRATE.acct_balance
    where acct_id = ORCHESTRATE.acct_id'
  -reject '/user/home/reject/reject.ds'"
```

Lookup Operations with classicfedlookup

With the classicfedlookup operator, you can perform a join between one or more external data source tables and a WebSphere DataStage data set. The resulting output data is a WebSphere DataStage data set contains both WebSphere DataStage and external data source data.

This join is performed by specifying a SELECT statement on one or more external data source tables and one or more key fields on which to perform the lookup.

This operator is particularly useful for sparse lookups, where the WebSphere DataStage data set which you are trying to match, is much smaller than the external data source table. If you expect to match 90% of your data, use the classicfedread and classicfedlookup operators.

Because classicfedlookup can perform lookups against more than one external data source table, the operation is useful when you join multiple external data source tables in one query.

The -query parameter corresponds to an SQL statement of this form:

```
select a,b,c from data.testtbl  
where  
Orchestrate.b = data.testtbl.c  
and  
Orchestrate.name = "Smith"
```

- The classicfedlookup operator replaces each WebSphere DataStage field name with a field value
- Submits the statement containing the value to the external data source
- Outputs a combination of external data source and WebSphere DataStage data.

Alternatively, you can use the -key and -table options to specify one or more key fields and one or more external data source tables. The following orchestrate shell options specify two keys and a single table:

```
-key a -key b -table data.testtbl
```

You get the same result as this SQL statement:

```
select * from data.testtbl  
where  
Orchestrate.a = data.testtbl.a  
and  
Orchestrate.b = data.testtbl.b
```

The resulting WebSphere DataStage output data set includes the WebSphere DataStage records and the corresponding rows from each referenced external data source table. When an external data source table has a column name that is the same as a WebSphere DataStage data set field name, the external data source column is renamed with the following syntax:

APT_integer_fieldname

An example is APT_0_lname. The integer component is incremented when duplicate names are encountered in additional tables. If the external data source table is not indexed on the lookup keys, the performance of this operator is likely to be poor.

classicfedlookup: properties

The lookup operations are performed with classicfedlookup operator properties

The table below contains the list of properties and the corresponding values for classicfedlookup operator.

Table 245. classicfedlookup Operator Properties

Property	Value
Number of input data sets	1
Number of output data sets	1; 2 if you include the -ifNotFound reject option
Input interface schema	Determined by the query
Output interface schema	Determined by the SQL query
Transfer behavior	Transfers all fields from input to output
Execution mode	Sequential or parallel (default)
Partitioning method	Not applicable

Table 245. *classicfedlookup Operator Properties (continued)*

Property	Value
Collection method	Not applicable
Preserve-partitioning flag in the output data set	Clear
Composite stage	No

classicfedlookup: syntax and options

Performs join operations between tables on external data sources and WebSphere DataStage data sets.

When your value contains a space or a tab character, you must enclose it in single quotation marks.

```
classicfedlookup
  [-close close_command]
  -datasourcename data_source_name
  -db_cs code_page_name
  -fetcharraysize n
  -ifNotFound fail | drop | reject | continue
  [-open open_command]
  -password passwd
  -query sql_query
  [-tabletable_name -key field [-key field ...]]
  -user username
```

You must specify either the -query option or one or more -table options with one or more -key fields.

The classicfedlookup operator is parallel by default. The options are as follows:

-close

Optionally specify an SQL statement to be executed after the insert array is processed. You cannot commit work using this option. The statement is executed only once on the conductor node.

-datasourcename

Specify the data source for all database connections. This option is required.

-db_cs

Optionally specify the ICU code page which represents the database character set in use. The default is ISO-8859-1. This option has this suboption:

-use_strings

If this suboption is not set, strings, not ustrings, are generated in the WebSphere DataStage schema.

-fetcharraysize

Specify the number of rows to retrieve during each fetch operation. The default number is 1.

-ifNotFound

Specify an action to be taken when a lookup fails.

fail

Stop job execution

drop

Drop the failed record from the output data set.

reject

Put records that are not found into a reject data set. You must designate an output data set for this option.

continue

Leave all records in the output data set which is outer join.

fail: Stop job execution.

drop: Drop the failed record from the output data set.

reject: Put records that are not found into a reject data set. You must designate an output data set for this option.

continue: Leave all records in the output data set (outer join).

-open

Optionally specify an SQL statement to be run before the insert array is processed. The statements are run only once on the conductor node.

-password

Specify the password for connections to the data source. This option might be required depending on the data source.

-query

Specify a lookup query to be run. This option and the -table option are mutually exclusive.

-table

Specify a table and key fields to generate a lookup query. This option and the -query option are mutually exclusive.

-filter *where_predicate*

Specify the rows of the table to exclude from the read operation. This predicate is appended to the WHERE clause of the SQL statement.

-selectlist *select_predicate*

Specify the list of column names that are in the SELECT clause of the SQL statement.

-key *field*

Specify a lookup key. A lookup key is a field in the table that joins with a field of the same name in the WebSphere DataStage data set. The -key option can be used more than once to specify more than one key field.

-user

Specify the user name for connections to the data source. This option might not be required depending on the data source.

Example of a classicfedlookup operation

Suppose you want to connect to the APT81 server as user user10 with the password test. You want to perform a lookup between a WebSphere DataStage data set and a table called target, on the key fields lname, fname, and DOB. You can specify classicfedlookup in either of two ways to accomplish this.

This orchestrate shell command uses the -table and -key options:

```
osh " classicfedlookup -key lname -key fname -key DOB
    < data1.ds > data2.ds "
```

This equivalent orchestrate shell command uses the -query option:

```
osh " classicfedlookup
    -query 'select * from target
        where lname = Orchestrate.lname
        and fname = Orchestrate.fname
        and DOB = Orchestrate.DOB'
    < data1.ds > data2.ds"
```

WebSphere DataStage prints the following items:

- lname
- fname

- DOB column names and values from the WebSphere DataStage input data set
- lname
- fname
- DOB column names and the values from the federated table

If a column name in the external data source table has the same name as a WebSphere DataStage output data set schema field name, the printed output shows the column in the external data source table renamed with this format:

`APT_integer_fieldname`

For example, lname is renamed to APT_0_lname.

Chapter 25. Header files

WebSphere DataStage comes with a range of header files that you can include in code when you are defining a Build stage.

The following sections list the header files and the classes and macros that they contain. See the header files themselves for more details about available functionality.

C++ classes - sorted by header file

apt_framework/ accessorbase.h

APT_AccessorBase
APT_AccessorTarget
APT_InputAccessorBase
APT_InputAccessorInterface
APT_OutputAccessorBase
APT_OutputAccessorInterface

apt_framework/ adapter.h

APT_AdapterBase
APT_ModifyAdapter
APT_TransferAdapter
APT_ViewAdapter

apt_framework/ collector.h

APT_Collector

apt_framework/ composite.h

APT_CompositeOperator

apt_framework/ config.h

APT_Config
APT_Node
APT_NodeResource
APT_NodeSet

apt_framework/ cursor.h

APT_CursorBase
APT_InputCursor
APT_OutputCursor

apt_framework/ dataset.h

APT_DataSet

apt_framework/ fieldsel.h

APT_FieldSelector

apt_framework/ fifocon.h

APT_FifoConnection

apt_framework/ gsubproc.h

APT_GeneralSubprocessConnection
APT_GeneralSubprocessOperator

apt_framework/ impexp/ impexp_ function.h
APT_GFImportExport

apt_framework/ operator.h
APT_Operator

apt_framework/ partitioner.h
APT_Partitioner
APT_RawField

apt_framework/ schema.h
APT_Schema
APT_SchemaAggregate
APT_SchemaField
APT_SchemaFieldList
APT_SchemaLengthSpec

apt_framework/ step.h
APT_Step

apt_framework/ subcursor.h
APT_InputSubCursor
APT_OutputSubCursor
APT_SubCursorBase

apt_framework/ tagaccessor.h
APT_InputTagAccessor
APT_OutputTagAccessor
APT_ScopeAccessorTarget
APT_TagAccessor

apt_framework/ type/ basic/ float.h
APT_InputAccessorToDFloat
APT_InputAccessorToSFloat
APT_OutputAccessorToDFloat
APT_OutputAccessorToSFloat

apt_framework/ type/ basic/ integer.h
APT_InputAccessorToInt16
APT_InputAccessorToInt32
APT_InputAccessorToInt64
APT_InputAccessorToInt8
APT_InputAccessorToUInt16
APT_InputAccessorToUInt32
APT_InputAccessorToUInt64
APT_InputAccessorToUInt8
APT_OutputAccessorToInt16
APT_OutputAccessorToInt32
APT_OutputAccessorToInt64
APT_OutputAccessorToInt8
APT_OutputAccessorToUInt16
APT_OutputAccessorToUInt32
APT_OutputAccessorToUInt64
APT_OutputAccessorToUInt8

apt_framework/ type/ basic/ raw.h
APT_InputAccessorToRawField
APT_OutputAccessorToRawField
APT_RawFieldDescriptor

apt_framework/ type/ conversion.h

APT_FieldConversion

APT_FieldConversionRegistry

apt_framework/ type/ date/ date.h

APT_DateDescriptor

APT_InputAccessorToDate

APT_OutputAccessorToDate

apt_framework/ type/ decimal/ decimal.h

APT.DecimalDescriptor

APT_InputAccessorToDecimal

APT_OutputAccessorToDecimal

apt_framework/ type/ descriptor.h

APT_FieldTypeDescriptor

APT_FieldTypeRegistry

apt_framework/ type/ function.h

APT_GenericFunction

APT_GenericFunctionRegistry

APT_GFComparison

APT_GFEquality

APT_GFPrint

apt_framework/ type/ protocol.h

APT_BaseOffsetFieldProtocol

APT_EMBEDDEDFieldProtocol

APT_FieldProtocol

APT_PrefixedFieldProtocol

APT_TraversedFieldProtocol

apt_framework/ type/ time/ time.h

APT_TimeDescriptor

APT_InputAccessorToTime

APT_OutputAccessorToTime

apt_framework/ type/ timestamp/ timestamp.h

APT_TimeStampDescriptor

APT_InputAccessorToTimeStamp

APT_OutputAccessorToTimeStamp

apt_framework/ utils/ fieldlist.h

APT_FieldList

apt_util/archive.h

APT_Archive

APT_FileArchive

APT_MemoryArchive

apt_util/ argvcheck.h

APT_ArgvProcessor

apt_util/basicstring.h

APT_BasicString

apt_util/ date.h

APT_ Date
apt_util/ dbinterface.h
APT_ DataBaseDriver
APT_ DataBaseSource
APT_ DBColumnDescriptor

apt_util/ decimal.h
APT_ Decimal

apt_util/ endian.h
APT_ ByteOrder

apt_util/ env_flag.h
APT_ EnvironmentFlag

apt_util/ errind.h
APT_ Error

apt_util/ errlog.h
APT_ ErrorLog

apt_util/ errorconfig.h
APT_ ErrorConfiguration

apt_util/ fast_alloc.h
APT_ FixedSizeAllocator
APT_ VariableSizeAllocator

apt_util/ fileset.h
APT_ FileSet

apt_util/ identifier.h
APT_ Identifier

apt_util/keygroup.h
APT_ KeyGroup

apt_util/locator.h
APT_ Locator

apt_util/persist.h
APT_ Persistent

apt_util/proplist.h
APT_ Property
APT_ PropertyList

apt_util/random.h
APT_ RandomNumberGenerator

apt_util/rtti.h
APT_ TypeInfo

apt_util/ string.h

APT_String
APT_StringAccum

apt_util/ time.h

APT_Time
APT_TimeStamp

apt_util/ustring.h

APT_UString

C++ macros - sorted by header file

apt_framework/ accessorbase.h

APT_DECLARE_ACCESSORS()
APT_IMPLEMENT_ACCESSORS()

apt_framework/ osh_name.h

APT_DEFINE_OSH_NAME()
APT_REGISTER_OPERATOR()

apt_framework/ type/ basic/ conversions_default.h

APT_DECLARE_DEFAULT_CONVERSION()
APT_DECLARE_DEFAULT_CONVERSION_WARN()

apt_framework/ type/ protocol.h

APT_OFFSET_OF()

apt_util/ archive.h

APT_DIRECTIONAL_SERIALIZATION()

apt_util/assert.h

APT_ASSERT() APT_DETAIL_FATAL()
APT_DETAIL_FATAL_LONG()
APT_MSG_ASSERT()
APT_USER_REQUIRE()
APT_USER_REQUIRE_LONG()

apt_util/condition.h

CONST_CAST() REINTERPRET_CAST()

apt_util/errlog.h

APT_APPEND_LOG()
APT_DUMP_LOG()
APT_PREPEND_LOG()

apt_util/exception.h

APT_DECLARE_EXCEPTION()
APT_IMPLEMENT_EXCEPTION()

apt_util/fast_alloc.h

APT_DECLARE_NEW_AND_DELETE()

apt_util/ logmsg.h

```
APT_DETAIL_LOGMSG()
APT_DETAIL_LOGMSG_LONG()
APT_DETAIL_LOGMSG_VERYLONG()
```

apt_util/persist.h

```
APT_DECLARE_ABSTRACT_PERSISTENT()
APT_DECLARE_PERSISTENT()
APT_DIRECTIONAL_POINTER_SERIALIZATION()
APT_IMPLEMENT_ABSTRACT_PERSISTENT()
APT_IMPLEMENT_ABSTRACT_PERSISTENT_V()
APT_IMPLEMENT_NESTED_PERSISTENT()
APT_IMPLEMENT_PERSISTENT()
APT_IMPLEMENT_PERSISTENT_V()
```

apt_util/rtti.h

```
APT_DECLARE_RTTI()
APT_DYNAMIC_TYPE()
APT_IMPLEMENT_RTTI_BASE()
APT_IMPLEMENT_RTTI_BEGIN()
APT_IMPLEMENT_RTTI_END()
APT_IMPLEMENT_RTTI_NOBASE()
APT_IMPLEMENT_RTTI_ONEBASE()
APT_NAME_FROM_TYPE()
APT_PTR_CAST()
APT_STATIC_TYPE()
APT_TYPE_INFO()
```

Product documentation

Documentation is provided in a variety of locations and formats, including in help that is opened directly from the product interface, in a suite-wide information center, and in PDF file books.

The information center is installed as a common service with IBM Information Server. The information center contains help for most of the product interfaces, as well as complete documentation for all product modules in the suite.

A subset of the product documentation is also available online from the product documentation library at publib.boulder.ibm.com/infocenter/iisinfsv/v8r1/index.jsp.

PDF file books are available through the IBM Information Server software installer and the distribution media. A subset of the information center is also available online and periodically refreshed at www.ibm.com/support/docview.wss?rs=14&uid=swg27008803.

You can also order IBM publications in hardcopy format online or through your local IBM representative.

To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order.

You can send your comments about documentation in the following ways:

- Online reader comment form: www.ibm.com/software/data/rcf/
- E-mail: comments@us.ibm.com

Contacting IBM

You can contact IBM for customer support, software services, product information, and general information. You can also provide feedback on products and documentation.

Customer support

For customer support for IBM products and for product download information, go to the support and downloads site at www.ibm.com/support/us/.

You can open a support request by going to the software support service request site at www.ibm.com/software/support/probsub.html.

My IBM

You can manage links to IBM Web sites and information that meet your specific technical support needs by creating an account on the My IBM site at www.ibm.com/account/us/.

Software services

For information about software, IT, and business consulting services, go to the solutions site at www.ibm.com/bournessolutions/us/en.

IBM Information Server support

For IBM Information Server support, go to www.ibm.com/software/data/integration/support/info_server/.

General information

To find general information about IBM, go to www.ibm.com.

Product feedback

You can provide general product feedback through the Consumability Survey at www.ibm.com/software/data/info/consumability-survey.

Documentation feedback

You can click the feedback link in any topic in the information center to comment on the information center.

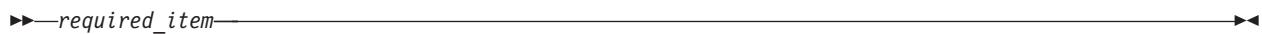
You can also send your comments about PDF file books, the information center, or any other documentation in the following ways:

- Online reader comment form: www.ibm.com/software/data/rcf/
- E-mail: comments@us.ibm.com

How to read syntax diagrams

The following rules apply to the syntax diagrams that are used in this information:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
 - The >>> symbol indicates the beginning of a syntax diagram.
 - The --> symbol indicates that the syntax diagram is continued on the next line.
 - The >--- symbol indicates that a syntax diagram is continued from the previous line.
 - The --->< symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).



- Optional items appear below the main path.



If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.



- If you can choose from two or more items, they appear vertically, in a stack.
If you must choose one of the items, one item of the stack appears on the main path.



If choosing one of the items is optional, the entire stack appears below the main path.



If one of the items is the default, it appears above the main path, and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.

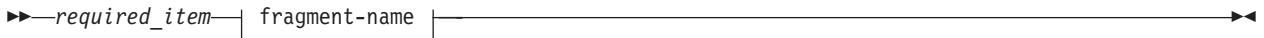


If the repeat arrow contains a comma, you must separate repeated items with a comma.



A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



Fragment-name:



- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown.
- Variables appear in all lowercase italic letters (for example, column-name). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).

Product accessibility

You can get information about the accessibility status of IBM products.

The IBM Information Server product modules and user interfaces are not fully accessible. The installation program installs the following product modules and components:

- IBM Information Server Business Glossary Anywhere
- IBM Information Server FastTrack
- IBM Metadata Workbench
- IBM WebSphere Business Glossary
- IBM WebSphere DataStage and QualityStage™
- IBM WebSphere Information Analyzer
- IBM WebSphere Information Services Director

For more information about a product's accessibility status, go to http://www.ibm.com/able/product_accessibility/index.html.

Accessible documentation

Accessible documentation for IBM Information Server products is provided in an information center. The information center presents the documentation in XHTML 1.0 format, which is viewable in most Web browsers. XHTML allows you to set display preferences in your browser. It also allows you to use screen readers and other assistive technologies to access the documentation.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

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

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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 and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM trademarks and certain non-IBM trademarks are marked on their first occurrence in this information with the appropriate symbol.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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

ITIL is a registered trademark and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

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

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

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

The United States Postal Service owns the following trademarks: CASS, CASS Certified, DPV, LACSLink, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACSLink licensee of the United States Postal Service.

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

Index

A

accessibility 811
aggtorec restructure operator 445
 example
 with multiple key options 448
 with toplevelkeys option 449
 without toplevelkeys option 448
 properties 446
 syntax and options 446
 toplevelkeys option 446
APT_AUTO_TRANSPORT_BLOCK_SIZE 79
APT_BUFFER_DISK_WRITE_INCREMENT 56, 61
APT_BUFFER_FREE_RUN 55
APT_BUFFER_MAXIMUM_TIMEOUT 56
APT_BUFFERING_POLICY 56
APT_CHECKPOINT_DIR 62
APT_CLOBBER_OUTPUT 62
APT_COLLATION_STRENGTH 69
APT_COMPILEOPT 58
APT_COMPILER 57
APT_CONFIG_FILE 62
APT_CONSISTENT_BUFFERIO_SIZE 61
APT_DATE CENTURY_BREAK_YEAR 65
APT_DB2INSTANCE_HOME 58
APT_DB2READ_LOCK_TABLE 58
APT_DB2WriteOperator
 write mode
 APT_DbmsWriteInterface::eAppend 717
 APT_DbmsWriteInterface::eReplace 717
 APT_DbmsWriteInterface::eTruncate 717
APT_DbmsWriteInterface::eAppend 717
APT_DbmsWriteInterface::eReplace 717
APT_DbmsWriteInterface::eTruncate 717
APT_DBNAME 58
APT_DEBUG_OPERATOR 59
APT_DEBUG_PARTITION 59
APT_DEBUG_SIGNALS 59
APT_DEBUG_STEP 59
APT_DEFAULT_TRANSPORT_BLOCK_SIZE 79
APT_DISABLE_COMBINATION 62
APT_DUMP_SCORE 72
APT_EncodeOperator
 example
 encode 117
 gzip and 117
APT_ERROR_CONFIGURATION 72
APT_EXECUTION_MODE 60, 63
APT_FILE_EXPORT_BUFFER_SIZE 72
APT_FILE_IMPORT_BUFFER_SIZE 72
APT_IMPEXP_CHARSET 69
APT_IMPORT_PATTERNUSES_FILESET 72
APT_INPUT_CHARSET 69
APT_IO_MAP/APT_IO_NOMAP and APT_BUFFERIO_MAP/
 APT_BUFFERIO_NOMAP 62
APT_IO_MAXIMUM_OUTSTANDING 68
APT_IOMGR_CONNECT_ATTEMPTS 68
APT_LATENCY_COEFFICIENT 79
APT_LINKER 58
APT_LINKOPT 58
APT_MAX_TRANSPORT_BLOCK_SIZE/
 APT_MIN_TRANSPORT_BLOCK_SIZE 79
APT_MONITOR_SIZE 64
APT_MONITOR_TIME 64
APT_MSG_FILELINE 74
APT_NO_PART_INSERTION 71
APT_NO_SAS_TRANSFORMS 77
APT_NO_SORT_INSERTION 77
APT_NO_STARTUP_SCRIPT 63
APT_OLD_BOUNDED_LENGTH 66
APT_OPERATOR_REGISTRY_PATH 66
APT_ORA_IGNORE_CONFIG_FILE_PARALLELISM 70
APT_ORA_WRITE_FILES 70
APT_ORACLE_NO_OPS 70
APT_ORACLE_PRESERVE_BLANKS 70
APT_ORAUPSER COMMIT_ROW_INTERVAL 71
APT_ORAUPSER COMMIT_TIME_INTERVAL 71
APT_ORAUPSER COMMIT_ROW_INTERVAL 625
APT_ORCHHOME 63
APT_OS_CHARSET 69
APT_OUTPUT_CHARSET 69
APT_PARTITION_COUNT 71
APT_PARTITION_NUMBER 71
APT_PERFORMANCE_DATA 14, 64
APT_PM_CONDUCTOR_HOSTNAME 68
APT_PM_DBX 60
APT_PM_NO_NAMED_PIPES 66
APT_PM_NO_TCPIP 68
APT_PM_PLAYER_MEMORY 74
APT_PM_PLAYER_TIMING 74
APT_PM_STARTUP_PORT 68
APT_PM_XLDB 61
APT_PM_XTERM 61
APT_PREVIOUS_FINAL_DELIMITER_COMPATIBLE 72
APT_RDBMS_COMMIT_ROWS 58
APT_RECORD_COUNTS 66, 74
APT_SAS_ACCEPT_ERROR 75
APT_SAS_CHARSET 76
APT_SAS_CHARSET_ABORT 76
APT_SAS_DEBUG 76
APT_SAS_DEBUG_IO 76
APT_SAS_DEBUG_LEVEL 76
APT_SAS_NO_PSDS_USTRING 77
APT_SAS_S_ARGUMENT 77
APT_SAS_SCHEMASOURCE_DUMP 77
APT_SAVE_SCORE 67
APT_SHOW_COMPONENT_CALLS 67
APT_STACK_TRACE 67
APT_STARTUP_SCRIPT 63
APT_STARTUP_STATUS 64
APT_STRING_CHARSET 69
APT_STRING_PADCHAR 72
APT_SYBASE_NULL_AS_EMPTY 78
APT_SYBASE_PRESERVE_BLANKS 78
APT_TERA_64K_BUFFERS 78
APT_TERA_NO_ERR_CLEANUP 78
APT_TERA_NO_PERM_CHECKS 78
APT_TERA_NO_SQL_CONVERSION 78
APT_TERA_SYNC_DATABASE 78
APT_TERA_SYNC_PASSWORD 78
APT_TERA_SYNC_USER 79
APT_THIN_SCORE 64
APT_WRITE_DS_VERSION 67

asesybaselookup and sybaselookup operators
 example 747
 properties 742
 syntax 742
 asesybaselookup and sybaselookup operatorsr 741
 asesybasereade and sybasereade operators 720
 action 721
 examples
 reading a table and modifying a field name 725
 properties 720
 syntax 723
 asesybaseupsert and sybaseupsert operators 736
 action 737
 example 740
 properties 737
 syntax 738
 asesybasewrite and sybasewrite operators 726
 action 727
 examples
 creating a table 734
 writing to a table using modify 735
 writing to an existing table 733
 properties 727
 syntax 730
 write modes 729
 writing to a multibyte database 726
 asesybasewrite operator
 options 730
 associativity of operators 259

B

bottlenecks 22
 build stage macros 43
 build stages 31

C

changeapply operator
 actions on change records 85
 example 90
 key comparison fields 87
 schemas 86
 transfer behavior 86
 changecapture operator
 data-flow diagram 92
 determining differences 92
 example
 dropping output results 97
 general 96
 key and value fields 92
 properties 92
 syntax and options 93
 transfer behavior 92
 charts, performance analysis 14
 checksum operator
 properties 99
 syntax and options 99
 Classic Federation interface operators
 accessing federated database 787
 classicfedlookup 800
 classicfedread 789
 classicfedupsert 797
 classicfedwrite 793
 national language support 787
 unicode character set 788

classicfedlookup operator
 options 802
 classicfedread
 -query 792
 column name conversion 791
 data type conversion 791
 classicfedread operator
 options 790
 classicfedupsert operator
 action 797
 classicfedwrite
 -drop 793
 -truncate 796
 char and varchar 797
 data type conversion 796
 classicfedwrite operator
 options 794
 collecting key 437
 collecting keys
 for sortmerge collecting 441
 collection methods 438
 collection operators
 roundrobin 438
 sortmerge 440
 collector
 ordered 437
 compare operator
 data-flow diagram 100
 example
 sequential execution 103
 restrictions 101
 results field 101
 syntax and options 101
 copy operator
 example 106
 sequential execution 107
 properties 104
 syntax and options 105
 CPU utilization 17
 custom stages 31
 resource estimation 19
 customer support 811

D

data set
 sorted
 RDBMS and 488, 502

DB2
 configuring DataStage access 633
 APT_DBNAME 634
 DB2 required privileges 633
 db2nodes.cfg system file 634
 configuring Orchestrate access 672
 db2lookup operator 668
 environment variable
 DB2INSTANCE 634
 joining DB2 tables and a DataStage data set 633
 joining DB2 tables and an Orchestrate data set 668
 operators that modify Orchestrate data types 647
 Orchestrate field conventions 647
 specifying DB2 table rows 639
 specifying the SQL SELECT statement 640
 using a node map 672

DB2 interface operators
 db2load 645
 db2lookup 668

DB2 interface operators (*continued*)
 db2part 665
 db2read 637
 db2upsert 661
 db2write 645
 establishing a remote connection 634
 handing # and \$ in column names 634
 national language support 636
 running multiple operators in a single step 635
 using the -padchar option 635

DB2 partitioning
 db2part operator 665

DB2DBDFT 58

DB2INSTANCE 634

db2load operator
 data-flow diagram 645
 DB2 field conventions 647
 default SQL INSERT statement 646
 example
 appending data to a DB2 table. 657
 DB2 table with an unmatched column 660
 unmatched Orchestrate fields 659
 writing data in truncate mode 658

matching fields 649

operators that modify Orchestrate data types for DB2 647

options 649

properties 645

requirements for using 657

specifying the select list 647

translation anomalies 672

write modes 648

db2lookup operator 668
 properties 669
 syntax and options 669

db2nodes command syntax 672

db2part operator 665
 example 667
 syntax and options 666

db2read operator
 data flow diagram 637
 example
 reading a DB2 table sequentially with the query option 644
 reading a DB2 table with the table option 643
 reading a table in parallel with the query option 644

operator action 637

Orchestrate conversion
 DB2 column names 638
 properties 637
 specifying DB2 table rows 639
 specifying the DB2 table name 639
 specifying the SQL SELECT statement 640
 syntax and options 641
 translation anomaly 672

db2upsert operator
 operator action 662
 partitioning 661
 properties 661

db2write operator 645
 data-flow diagram 645
 DB2 field conventions 647
 default SQL INSERT statement 646
 example
 appending data to a DB2 table 657
 DB2 table with an unmatched column 660
 unmatched Orchestrate fields 659
 writing data in truncate mode 658

db2write operator (*continued*)
 matching fields 649
 operators that modify Orchestrate data types for DB2 647
 properties 645
 specifying the select list 647
 syntax and options 649
 translation anomalies 672
 write modes 648

diff operator
 data flow diagram 108
 example
 dropping output results 113
 typical 113
 properties 108
 syntax and options 110

disk space 17, 23

disk utilization 17

documentation
 accessible 811

duplicates, removing 226

dynamic models 18
 creating 17
 examples 21, 22
 overview 17

E

encode operator
 data flow diagram 115
 encoded data sets and partitioning 116
 encoding data sets 116
 example 117
 properties 115

encoded data sets and partitioning 116

encoding data sets 116

entire partitioner 411
 data flow diagram 412
 properties 413
 syntax 413
 using 412

environment variable
 APT_DBNAME 634
 APT_ORAUPSERV_COMMIT_ROW_INTERVAL 625
 APT_PERFORMANCE_DATA 14, 64
 DB2INSTANCE 634

example build stage 46

examples
 resource estimation
 bottlenecks 22
 input projections 23
 merging data 21
 overview 20
 sorting data 22

export utility operator
 data set versus export schema 351
 example
 exporting to a single file 354
 exporting to multiple files 355

export formats 346

properties 345

specifying data destination 351
 file sets 352
 files and named pipes 352
 list of destination programs 353
 nodes and directories 353

specifying export schema 351
 default exporting of all fields 351

export utility operator (*continued*)
specifying export schema (*continued*)
defining properties 351
exporting all fields with overrides and new
properties 351
exporting selected fields 351

F

field_export restructure operator

data-flow diagram 450

example 452

properties 451

transfer behavior 450, 453

field_import restructure operator 453

data flow diagram 453

example 455

properties 454

syntax and options 454

transfer behavior 454

fields

changing data type 173

data-type conversion errors 176

data-type conversions

date field 176, 262

decimal field 182

raw field length extraction 184

string conversions and lookup tables 189, 276

string field 185

time field 192

timestamp field 195

default data-type conversion 174

duplicating with a new name 173

explicitly converting the data type 176

transfer behavior of the modify operator 172

filter operator 117

data flow diagram 117

example

comparing two fields 122

evaluating input records 123

filtering wine catalog recipients 123

testing for a null 123

input data types 121

Job Monitor information 119

customizing messages 120

example messages 120

properties 118

supported boolean expressions 121

fullouterjoin operator 524

example 525

output data set 524

syntax and options 524

funnel operator

syntax and options 127

G

general operators 84

filter 117

lookup 143

modify 170

generator operator

data-flow diagram 130

defining record schema 134

example

general 132

generator operator (*continued*)

example (*continued*)

parallel execution 133

using with an input data set 133

generator options for DataStage data types 134

properties 130

record schema

date fields 136

decimal fields 137

null fields 140

numeric fields 135

raw fields 137

string fields 138

time fields 139

timestamp fields 139

supported data types 132

syntax and options 130

usage details 131

gzip UNIX utility

using with the encode operator 117

H

hash partitioner 413

data flow diagram 415

example 414

properties 416

syntax and option 416

using 415

head operator

example

extracting records from a large data set 143

syntax and options 141

hpread INFORMIX interface operator 687

data flow diagram 688

example 690

properties 688

special features 687

syntax and options 689

hplwrite INFORMIX interface operator 690

data flow diagram 690

listing of examples 692

properties 691

special features 690

syntax and options 691

I

IBM format files 333

exporting data 346

IBM support 811

ICU character sets 788

import utility operator

import source file data types 333

properties 332

syntax and options 333

import/export utility introduction

ASCII and EBCDIC conversion 326

ASCII to EBCDIC 329

EBCDIC to ASCII 326

table irregularities 331

error handling 325

failure 325

warning 326

format of imported data 315

implicit import and export 322

import/export utility introduction (*continued*)
 default export schema 324
 default import schema 324
 no specified schema 322
 overriding defaults 324
record schema
 comparison of partial and complete 318
 defining complete schema 317
 defining partial schema 319
 export example 316
 exporting with partial schema 320
 import example 316
import/export utility properties
 field properties 358
 date 362
 decimal 362
 nullable 363
 numeric 359
 raw 362
 tagged subrecord 364
 time 362
 timestamp 362
 wstring 362
 vector 363
 property description 367
 ascii 368
 big_endian 369
 binary 369
 c_format 370
 check_intact 372
 date_format 372
 days_since 376
 default 376
 delim 378
 delim_string 379
 drop 380
 ebcdic 381
 export_ebcdic_as_ascii 381
 fill 381
 final_delim 382
 final_delim_string 383
 fix_zero 383
 generate 384
 import_ascii_as_ebcdic 384
 in_format 385
 intact 386
 julian 386
 link 386
 little_endian 387
 max_width 388
 midnight_seconds 388
 native_endian 389
 nofix_zero 389
 null_field 389
 null_length 390
 out_format 391
 packed 392
 padchar 393
 position 394
 precision 395
 prefix 395
 print_field 396
 quote 397
 record_delim 398
 record_delim_string 364, 398
 record_format 399
 record_length 399

import/export utility properties (*continued*)
 property description (*continued*)
 record_prefix 399
 reference 400
 round 400
 scale 401
 separate 402
 skip 402
 tagcase 403
 text 403
 time_format 404
 timestamp_format 407
 vector_prefix 408
 width 388, 409
 zoned 409
 record-level properties 358
 setting properties 356
 defining field-level properties 357
 table of all record and field properties 364
INFORMIX
 accessing 675
 configuring your environment 675
INFORMIX interface operators
 hplread 687
 hplwrite 690
 infhread 693
 xpsread 698
 xpswrite 700
 Informix interface read operators
 operator action 675
INFORMIX interface read operators
 column name conversion 677
 data type conversion 677
 example 678
 operator action 676
INFORMIX interface write operators
 column name conversion 680
 data type conversion 681
 dropping unmatched fields 682
 example
 appending data 683
 handling unmatched fields 685
 table with unmatched column 686
 writing in truncate mode 684
 execution modes 680
 limitations
 column name length 682
 field types 682
 write modes 681
 writing fields 682
 infhread INFORMIX interface operator 693
 data flow diagram 693
 properties 693
 syntax and options 694
 inffwrite INFORMIX interface operator 695
 data flow diagram 695
 listing of examples 698
 properties 696
 syntax and options 696
innerjoin operator 518
 example 519
 syntax and option 519
input projections
 example 23
 making 20
 overview 17

iWay
accessing 773
iWay operators 773
 iwayread 773
iwaylookup operator 777
iwayread operator 773
 syntax 775

J

job performance
 analysis 14
 data
 design-time recording 14
 runtime recording 14
 overview 14
join operators
 fullouterjoin 524
 general characteristics
 input data set requirements 517
 memory use 517
 multiple keys and values 517
 properties 516
 transfer behavior 517
innerjoin 518
leftouterjoin 520
rightouterjoin 522
See also individual operator entries [join operators
 zzz] 515

L

leftouterjoin operator 520
 example 521
 syntax and option 520
legal notices 817
lookup operator 143
 create-only mode 149
 dropping a field with the transform operator 152
 example
 handling duplicate fields 152
 interest rate lookup 151
 handling error conditions 151
 lookup with single table record 150
operator behavior 143
properties 145
renaming duplicate fields 152
table characteristics 150, 151, 152

M

makerangemap utility
 creating a range map 433
 example commands 433
 specifying sample size 433
 syntax and options 431
 using with the range partitioner 433
makesubrec restructure operator 457
 data flow diagram 457
 properties 458
 subrecord length 458
 syntax and options 459
 transfer behavior 458
makevect restructure operator
 data flow diagram 460

makevect restructure operator (*continued*)
 example
 general 462
 missing input fields 462
 non-consecutive fields 461
 properties 461
 syntax and option 461
 transfer behavior 461
merge operator
 data flow diagram 154
 example
 application scenario 166
 handling duplicate fields 164, 165, 166
merging operation 160
merging records 156
 example diagram 160
 master record and update record 157
 multiple update data sets 157
missing records 168
 handling bad master records 168
 handling bad update records 168
properties 154
syntax and options 154
 typical data flow 159
models
 resource estimation
 accuracy 18
 creating 17
 dynamic 18
 static 18
modify operator 170
 aggregate schema components 201
 changing field data type 173
 data-flow diagram 171
 data-type conversion errors 176
 data-type conversion table 203
 data-type conversions
 date field 176, 262
 decimal field 182
 raw field length extraction 184
 string conversions and lookup tables 189, 276
 string field 185
 time field 192
 timestamp field 195
 default data-type conversion 174
 duplicating a field with a new name 173
 explicitly converting the data type 176
 null support 197
 partial schemas 200
 properties 171
 transfer behavior 172
 vectors 201
modulus partitioner
 data flow diagram 417
 example 418
 syntax and option 418

N

node maps
 for DB2 interface operators 672
nulls
 modify operator 197

O

ODBC interface operators 527
accessing ODBC from DataStage 527
national language support 527
odbclockup 547
odbcread 528
odbcupsert 543
odbctime 534
odbclockup operator 547
data flow diagram 548
example 551
options 549
properties 549
syntax 549
odbcread operator 528
action 531
column name conversion 531
data flow diagram 529
examples
reading a table and modifying a field name 533
options 530
syntax 529
odbcreatordata type conversion 532
odbcupsert operator 543
action 544
data flow diagram 543
example 546
options 545
properties 544
syntax 545
odbctime operator 534
action 535
data flow diagram 535
examples
creating a table 541
writing to existing table 540
writing using the modify operator 542
options 538
properties 535
syntax 538
writing to a multibyte database 535
operator
associativity 259
writerangemap 311, 428
operators
DB2 interface 673
general 84
join
See join operators 515
Oracle interface 631
precedence 259
sort 513
Teradata interface 719
Oracle
accessing from DataStage
PATH and LIBPATH requirements 599
data type restrictions 611
indexed tables 610
joining Oracle tables and a DataStage data set 627
Orchestrate to Oracle data type conversion 611
performing joins between data sets 605
reading tables
SELECT statement 604
specifying rows 604
SQL query restrictions 604
writing data sets to
column naming conventions 611

Oracle (*continued*)

writing data sets to (*continued*)

dropping data set fields 613

Oracle interface operators

handing # and \$ in column names 599

national language support 527, 600

oralookup 627

oraread 601

oraupsert 622

orawrite 609

preserving blanks in fields 599

oralookup operator 627

example 631

properties 629

syntax and options 629

oraread operator 601

column name conversion 603

data flow diagram 601

data-type conversion 603

operator action 602

Oracle record size 604

properties 601

specifying an Oracle SELECT statement 604

specifying processing nodes 602

specifying the Oracle table 604

syntax and options 605

oraupsert operator 622

data-flow diagram 622

environment variables 625

APT_ORAUPINSERT_COMMIT_ROW_INTERVAL 625

example 627

operator action 623, 662

properties 623, 661

orawrite operator 609

column naming conventions 611

data flow diagram 609

data-type restrictions 611

example

creating an Oracle table 621

using the modify operator 621

writing to an existing table 620

execution modes 611

matched and unmatched fields 613

operator action 610

Orchestrate to Oracle data type conversion 611

properties 610

required privileges 613

syntax and options 613

write modes 612

writing to indexed tables 610

ordered collection operator

properties 438

OSH_BUILDOP_CODE 57

OSH_BUILDOP_HEADER 57

OSH_BUILDOP_OBJECT 57

OSH_BUILDOP_XLC_BIN 57

OSH_DUMP 75

OSH_ECHO 75

OSH_EXPLAIN 75

OSH_PRELOAD_LIBS 67

OSH_PRINT_SCHEMAS 75

OSH_STDOUT_MSG 67

P

Parallel SAS data set format 557

partition sort
 RDBMS and 502

partitioners
 entire 411
 hash 413
 random 419
 range 421
 roundrobin 433
 same 435

pcompress operator
 compressed data sets
 partitioning 206
 using orchadmin 207

data-flow diagram 205

example 207

mode
 compress 205
 expand 205

properties 205

syntax and options 205

UNIX compress facility 206

peek operator
 syntax and options 209

performance analysis
 charts 14
 overview 14
 recording data
 at design time 14
 at run time 14

PFTP operator 211
 data flow diagram 212
 properties 212
 restartability 218

pivot operator
 properties 219
 syntax and options 219

precedence of operators 259

product accessibility
 accessibility 815

projections, input
 example 23
 making 20
 overview 17

promotesubrec restructure operator
 data-flow diagram 463
 example 464
 properties 464
 syntax and option 464

properties
 classicfedlookup 801
 classicfedupsert 798
 classicfedwrite 795

Properties
 classicfedread 789

psort operator
 data flow diagram 503
 performing a total sort 508
 properties 504
 specifying sorting keys 502
 syntax and options 504

random partitioner (continued)
 using 420

range map
 creating with the makerangemap utility 433

range partitioner 421
 algorithm 422
 data flow diagram 426
 example 425
 properties 426
 specifying partitioning keys 422
 syntax and options 426
 using 425

range partitioning
 and the psort operator 510

remdup operator
 data-flow diagram 221
 data-flow with hashing and sorting 224
 effects of case-sensitive sorting 225
 example
 case-insensitive string matching 226
 removing all but the first duplicate 226
 using two keys 226

example osh command with hashing and sorting 224

properties 222
 removing duplicate records 223
 syntax and options 222
 usage 226
 retaining the first duplicate record 224
 retaining the last duplicate record 224
 using the last option 226

remove duplicates 226

reports, resource estimation 20

resource estimation
 custom stages 19
 examples
 finding bottlenecks 22
 input projections 23
 merging data 21
 overview 20
 sorting data 22

models
 accuracy 18
 creating 17
 dynamic 18
 static 18

overview 17

projections 20

reports 20

restructure operators
 aggtorec 445
 field_import 453
 makesubrec 457
 splitsubrec 465
 tagswitch 477

rightouterjoin operator 522
 example 523
 syntax and option 522

roundrobin collection operator 438
 syntax 440

roundrobin partitioner 433
 data flow diagram 434
 properties 434
 syntax 434
 using 434

R

random partitioner 419
 data flow diagram 420
 properties 421
 syntax 421

S

- same partitioner 435
 - data flow diagram 435
 - properties 436
 - syntax 436
- sample operator
 - data-flow diagram 227
 - example 229
 - properties 228
 - syntax and options 228
- SAS data set format 558
- SAS DATA steps
 - executing in parallel 563
- SAS interface library
 - configuring your system 555
 - converting between DataStage and SAS data types 560
- DataStage example 562
 - example data flow 556
 - executing DATA steps in parallel 563
 - executing PROC steps in parallel 569
 - getting input from a DataStage or SAS data set 559
 - getting input from a SAS data set 558
 - overview 553
- Parallel SAS data set format 557
 - parallelizing SAS code
 - rules of thumb for parallelizing 573
 - SAS programs that benefit from parallelization 573
 - pipeline parallelism and SAS 555
 - SAS data set format 558
 - sequential SAS data set format 557
 - using SAS on sequential and parallel systems 553
 - writing SAS programs 553
- SAS interface operators
 - controlling ustring truncation 579
 - determining SAS mode 577
 - environment variables 581
 - generating a Proc Contents report 580
 - long name support 580
 - sas 587
 - sascontents 595
 - sasin 582
 - sasout 592
 - specifying an output schema 579
- sas operator 587
 - data-flow diagram 587
 - properties 587
- SAS operator
 - syntax and options 588
- SAS PROC steps
 - executing in parallel 569
- sascontents operator
 - data flow diagram 595
 - data-flow diagram 595
 - properties 596
 - syntax and options 596
- sasin operator
 - data flow diagram 583
 - data-flow diagram 583
 - properties 583
 - syntax and options 583
- sasout operator 592
 - data flow diagram 593
 - properties 593
 - syntax and options 593
- scratch space 17, 23
- screen readers 811
- sequence operator
 - example 230
 - properties 230
 - syntax and options 230
- sequential SAS data set format 557
- software services 811
- sort operators
 - tsort 485
- sortfunnel operator
 - input requirements 126
 - setting primary and secondary keys 127
 - syntax and options 127
- sortmerge collection operator 440
 - collecting keys 441
 - ascending order 443
 - case-sensitive 443
 - data types 442
 - primary 442
 - secondary 442
 - data flow diagram 441
 - properties 443
 - syntax and options 443
- splitsubrec restructure operator 465
 - data-flow diagram 465
 - example 466
 - properties 465
 - syntax and option 466
- splitvect restructure operator
 - data-flow diagram 467
 - example 468
 - properties 468
 - syntax and option 468
- SQL Server interface operators 749
 - accessing SQL Server from DataStage 749
 - national language support 749, 773
 - sqlsrvrread 750
 - sqlsrvrupsert 764
 - sqlsrvrwrite 756
- SQL Server interface options
 - sqlsrvrlookup 768
- sqlsrvrlookup operator 768
 - data flow diagram 769, 779
 - example 772, 782
 - options 770
 - properties 769
 - syntax 770
- sqlsrvrread operator 750
 - action 750, 774
 - column name conversion 751
 - data flow diagram 750, 774
 - data type conversion 751, 774
 - example 755
 - options 753, 775, 780
 - properties 750, 774
 - syntax 753
- sqlsrvrupsert operator 764
 - action 764
 - data flow diagram 764
 - example 767
 - options 766
 - properties 764
 - syntax 765
- sqlsrvrwrite operator 756
 - action 757
 - data flow diagram 756
 - examples
 - creating a table 762

sqlsrvrwrite operator (*continued*)
 examples (*continued*)

- writing to a table using modify 763
- writing to an existing table 761

 options 759

 properties 756

 syntax 759

 write modes 758

static models 18

- creating 17
- overview 17

support, customer 811

switch operator

- data flow diagram 231
- discard value 232
- example 235
- Job Monitor information 237
 - example messages 237
- properties 233
- syntax and options 233

Sybase environment variables 78

Sybase interface operators 719

- accessing Sybase from DataStage 719
- asesybaselookup and sybaselookup 741
- asesybasereade and sybasereade 720
- asesybaseupsert and sybaseupsert 736
- asesybasewrite and sybasewrite 726
 - national language support 720

sybaselookup operator

- options 745

sybasereade operator

- column name conversion 721
- data type conversion 721
- options 724

sybaseupsert operator

- options 739

sybasewrite operator

- data type conversion 728

syntax

- classicfedlookup 802
- classicfedread 789
- classicfedupsert 798
- classicfedwrite 793

T

tagbatch restructure operator

- added, missing, and duplicate fields 471
- data-flow diagram 470
- example
 - with missing and duplicate cases 475
 - with multiple keys 476
 - with simple flattening of tag cases 474
- input data set requirements 472
- operator action and transfer behavior 470
- properties 471
- syntax and options 472
- tagged fields and operator limitations 469

tagswitch restructure operator 477

- case option 478
- data flow diagram 477
- example
 - with default behavior 480
 - with one case chosen 481
- input and output interface schemas 478
- properties 478
- syntax and options 479

tagswitch restructure operator (*continued*)

- using 478

tail operator

- example
 - default behavior 239
 - using the nrecs and part options 239
- properties 238

Teradata

- writing data sets to
 - data type restrictions 715
 - dropping data set fields 714
 - write mode 714

Teradata interface operators

- terawrite operator 711

teraread operator

- column name and data type conversion 708
- data-flow diagram 707
- national language support 705
- properties 707
- restrictions 709
- specifying the query 707
- syntax and options 709

terawrite operator 711

- column name and data type conversion 712
- correcting load errors 713
- data-flow diagram 712
- limitations 714
- national language support 705
- properties 712
- restrictions 715
- syntax and options 715
- write mode
 - append 714
 - create 714
 - replace 714
 - truncate 714
 - writing fields 714

trademarks 819

transform operator

- example applications 288
- lookup-table functions 262
- specifying lookup tables 253
- syntax and options 241

Transformation Language

- built-in functions
 - bit-manipulation functions 285
 - data conversion and data mapping 261
 - mathematical functions 274
 - miscellaneous functions 286, 287
 - null handling functions 274
 - string functions 277
 - time and timestamp transformations 269
 - cstring functions 281
- conditional branching 260
 - if ... else 260
- data types and record fields 255
- differences with the C language 287
 - casting 288
 - flow control 288
 - labeled statements 288
 - local variables 287
 - names and keywords 287
 - operators 287
 - pre-processor commands 288
- expressions 257
 - language elements 257
 - operators 257

Transformation Language (*continued*)

- general structure 251
- local variables 256
- names and keywords 251
- specifying datasets
 - complex field types 256
 - simple fieldtypes 255

tsort operator

- configuring 487
- data flow diagram 489
- example
 - sequential execution 493
- performing a total sort 495
- properties 489
- sorting keys
 - case sensitivity 489
 - restrictions 488
 - sort order 489
- specifying sorting keys 488
- syntax and options 490
- using a sorted data set 487
- using a sorted data set with a sequential operator 487

U

UNIX compress utility

- invoked from the pcompress operator 206

W

wrapped stages 31

writerangemap operator 311, 428

- data-flow diagram 311, 428
- properties 312, 429

X

xpread INFORMIX interface operator 698

- data-flow diagram 698
- properties 699
- syntax and options 699

xpswrite INFORMIX interface operator 700

- data-flow diagram 700
- listing of examples 703
- properties 701
- syntax and options 701

IBM[®]

Printed in USA

LC18-9892-02



Spine information:

IBM WebSphere DataStage and QualityStage

Version 8 Release 1

Parallel Job Advanced Developer Guide