IBM InfoSphere DataStage and QualityStage
Version 8 Release 7

Parallel Job Developer's Guide

SC19-3459-00

IBM InfoSphere DataStage and QualityStage
Version 8 Release 7

Parallel Job Developer's Guide

SC19-3459-00

Note

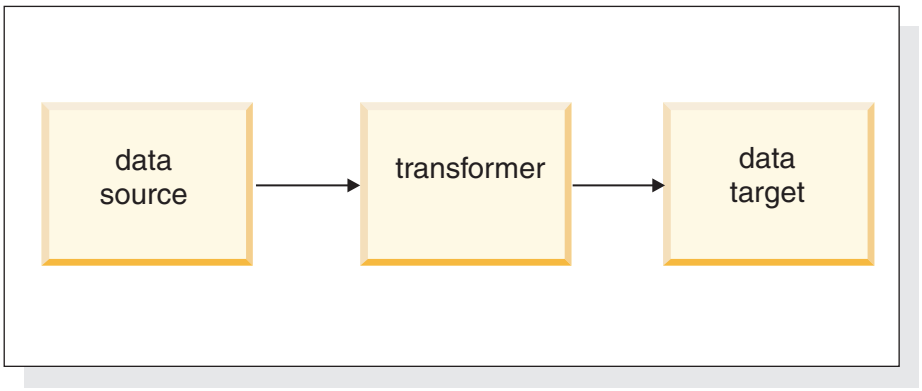
Before using this information and the product that it supports, read the information in "Notices and trademarks" on page 695.

Contents

Chapter 1. InfoSphere DataStage parallel jobs

InfoSphere® DataStage® jobs consist of individual stages. Each stage describes a particular process, this might be accessing a database or transforming data in some way. For example, one stage might extract data from a data source, while another transforms it. Stages are added to a job and linked together using the Designer.

The following diagram represents one of the simplest jobs you could have: a data source, a Transformer (conversion) stage, and the final database. The links between the stages represent the flow of data into or out of a stage. In a parallel job each stage would correspond to a process. You can have multiple instances of each process to run on the available processors in your system.



You must specify the data you want at each stage, and how it is handled. For example, do you want all the columns in the source data, or only a select few? Are you going to rename any of the columns? How are they going to be transformed?

You lay down these stages and links on the canvas of the InfoSphere DataStage Designer. You specify the design as if it was sequential, InfoSphere DataStage determines how the stages will become processes and how many instances of these will actually be run.

InfoSphere DataStage also allows you to store reusable components in the Repository which can be incorporated into different job designs. You can import these components, or entire jobs, from other InfoSphere DataStage Projects using the Designer. You can also import meta data directly from data sources and data targets.

Guidance on how to construct your job and define the required meta data using the Designer is in the *InfoSphere DataStage Designer Client Guide*. Chapter 4 onwards of this manual describe the individual stage editors that you might use when developing parallel jobs.

Chapter 2. Designing parallel jobs

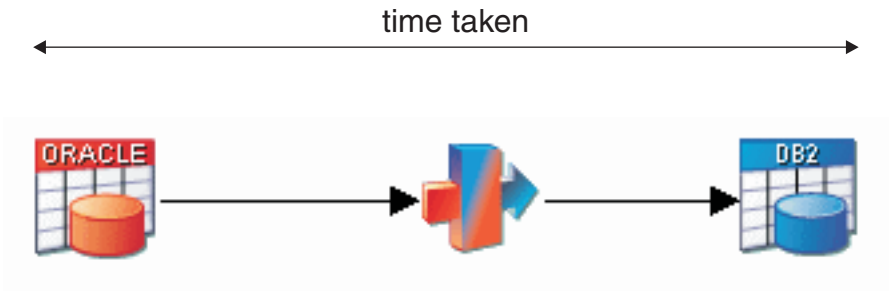
Parallel jobs brings the power of parallel processing to your data extraction and transformation applications. This chapter gives a basic introduction to parallel processing, and describes some of the key concepts in designing parallel jobs for InfoSphere DataStage. If you are new to InfoSphere DataStage, you should read the introductory topics about the IBM® InfoSphere DataStage and QualityStage™ Designer first so that you are familiar with the Designer client interface and the way jobs are built from stages and links.

Parallel processing

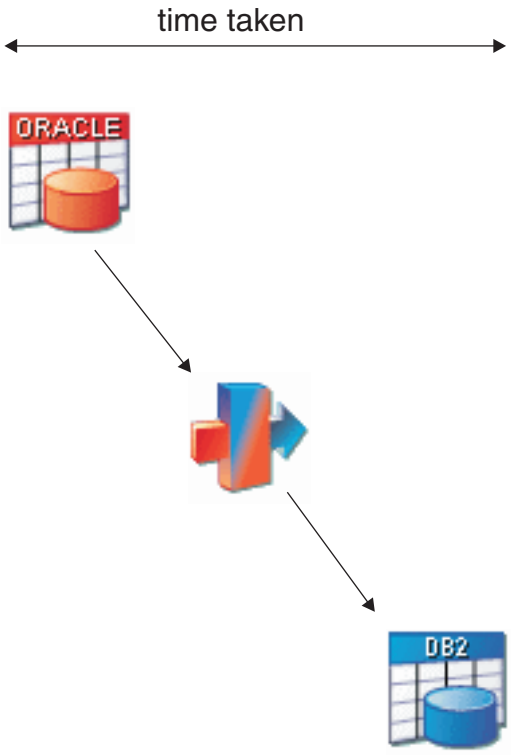
There are two basic types of parallel processing; pipeline and partitioning. InfoSphere DataStage allows you to use both of these methods. The following sections illustrate these methods using a simple parallel job which extracts data from a data source, transforms it in some way, then writes it to another data source. In all cases this job would appear the same on your Designer canvas, but you can configure it to behave in different ways (which are shown diagrammatically).

Pipeline parallelism

If you ran the example job on a system with at least three processors, the stage reading would start on one processor and start filling a pipeline with the data it had read. The transformer stage would start running on another processor as soon as there was data in the pipeline, process it and start filling another pipeline. The stage writing the transformed data to the target database would similarly start writing as soon as there was data available. Thus all three stages are operating simultaneously. If you were running sequentially, there would only be one instance of each stage. If you were running in parallel, there would be as many instances as you had partitions (see next section).



conceptual representation of job running with no parallelism



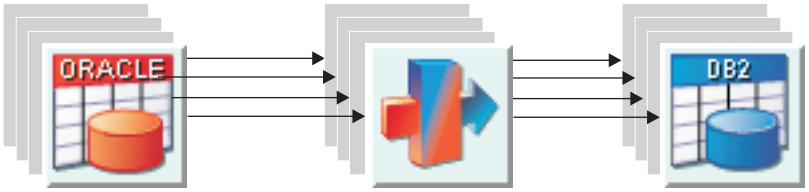
conceptual representation of same job using pipeline parallelism

Partition parallelism

Imagine you have the same simple job as described above, but that it is handling very large quantities of data. In this scenario you could use the power of parallel processing to your best advantage by partitioning the data into a number of separate sets, with each partition being handled by a separate instance of the job stages.

Using partition parallelism the same job would effectively be run simultaneously by several processors, each handling a separate subset of the total data.

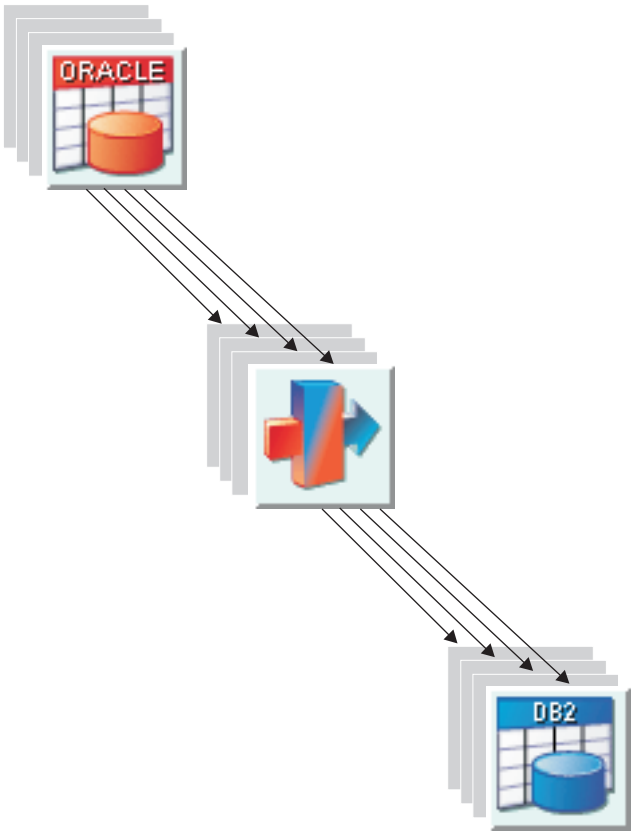
At the end of the job the data partitions can be collected back together again and written to a single data source.



Conceptual representation of job using partition parallelism

Combining pipeline and partition parallelism

In practice you will be combining pipeline and partition parallel processing to achieve even greater performance gains. In this scenario you would have stages processing partitioned data and filling pipelines so the next one could start on that partition before the previous one had finished.

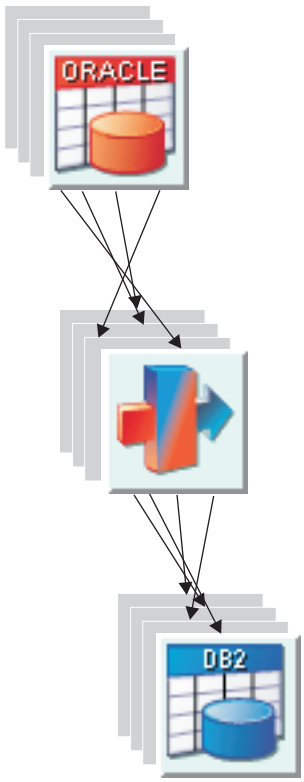


Conceptual representation of job using pipeline and partition parallelism

Repartitioning data

In some circumstances you might want to actually repartition your data between stages. This might happen, for example, where you want to group data differently. Say you have initially processed data based on customer last name, but now want to process on data grouped by zip code. You will need to repartition to ensure that all customers sharing the same zip code are in the same group. InfoSphere

DataStage allows you to repartition between stages as and when needed (although note there are performance implications if you do this and you might affect the balance of your partitions – see *Parallel Job Advanced Developer Guide*).



Conceptual representation of data repartitioning

Further details about how InfoSphere DataStage actually partitions data, and collects it together again, see "Partitioning, repartitioning, and collecting data" on page 8.

Parallel processing environments

The environment in which you run your parallel jobs is defined by your system's architecture and hardware resources. All parallel processing environments are categorized as one of:

- SMP (symmetric multiprocessing), in which some hardware resources might be shared among processors. The processors communicate via shared memory and have a single operating system.
- Cluster or MPP (massively parallel processing), also known as shared-nothing, in which each processor has exclusive access to hardware resources. MPP systems are physically housed in the same box, whereas cluster systems can be physically dispersed. The processors each have their own operating system, and communicate via a high-speed network.

SMP systems allow you to scale up the number of processors, which might improve performance of your jobs. The improvement gained depends on how your job is limited:

- CPU-limited jobs. In these jobs the memory, memory bus, and disk I/O spend a disproportionate amount of time waiting for the processor to finish its work. Running a CPU-limited application on more processors can shorten this waiting time so speed up overall performance.

- Memory-limited jobs. In these jobs CPU and disk I/O wait for the memory or the memory bus. SMP systems share memory resources, so it might be harder to improve performance on SMP systems without hardware upgrade.
- Disk I/O limited jobs. In these jobs CPU, memory and memory bus wait for disk I/O operations to complete. Some SMP systems allow scalability of disk I/O, so that throughput improves as the number of processors increases. A number of factors contribute to the I/O scalability of an SMP, including the number of disk spindles, the presence or absence of RAID, and the number of I/O controllers.

In a cluster or MPP environment, you can use the multiple processors and their associated memory and disk resources in concert to tackle a single job. In this environment, each processor has its own dedicated memory, memory bus, disk, and disk access. In a shared-nothing environment, parallelization of your job is likely to improve the performance of CPU-limited, memory-limited, or disk I/O-limited applications.

The configuration file

One of the great strengths of InfoSphere DataStage is that, when designing jobs, you don't have to worry too much about the underlying structure of your system, beyond appreciating its parallel processing capabilities. If your system changes, is upgraded or improved, or if you develop a job on one platform and implement it on another, you don't necessarily have to change your job design.

InfoSphere DataStage learns about the shape and size of the system from the configuration file. It organizes the resources needed for a job according to what is defined in the configuration file. When your system changes, you change the file not the jobs.

The configuration file describes available processing power in terms of processing nodes. These might, or might not, correspond to the actual number of processors in your system. You might, for example, want to always leave a couple of processors free to deal with other activities on your system. The number of nodes you define in the configuration file determines how many instances of a process will be produced when you compile a parallel job.

Every MPP, cluster, or SMP environment has characteristics that define the system overall as well as the individual processors. These characteristics include node names, disk storage locations, and other distinguishing attributes. For example, certain processors might have a direct connection to a mainframe for performing high-speed data transfers, while others have access to a tape drive, and still others are dedicated to running an RDBMS application. You can use the configuration file to set up node pools and resource pools. A pool defines a group of related nodes or resources, and when you design a parallel job you can specify that execution be confined to a particular pool.

The configuration file describes every processing node that InfoSphere DataStage will use to run your application. When you run a parallel job, InfoSphere DataStage first reads the configuration file to determine the available system resources.

When you modify your system by adding or removing processing nodes or by reconfiguring nodes, you do not need to alter or even recompile your parallel job. Just edit the configuration file.

The configuration file also gives you control over parallelization of your job during the development cycle. For example, by editing the configuration file, you can first run your job on a single processing node, then on two nodes, then four, then eight, and so on. The configuration file lets you measure system performance and scalability without actually modifying your job.

You can define and edit the configuration file using the Designer client.

Partitioning, repartitioning, and collecting data

You have already seen how you can use partitioning of data to implement parallel processing in your job. These topics take a closer look at how you can partition data in your jobs, and collect it together again.

Partitioning

In the simplest scenario you probably won't be bothered how your data is partitioned. It is enough that it is partitioned and that the job runs faster. In these circumstances you can safely delegate responsibility for partitioning to InfoSphere DataStage. Once you have identified where you want to partition data, InfoSphere DataStage will work out the best method for doing it and implement it.

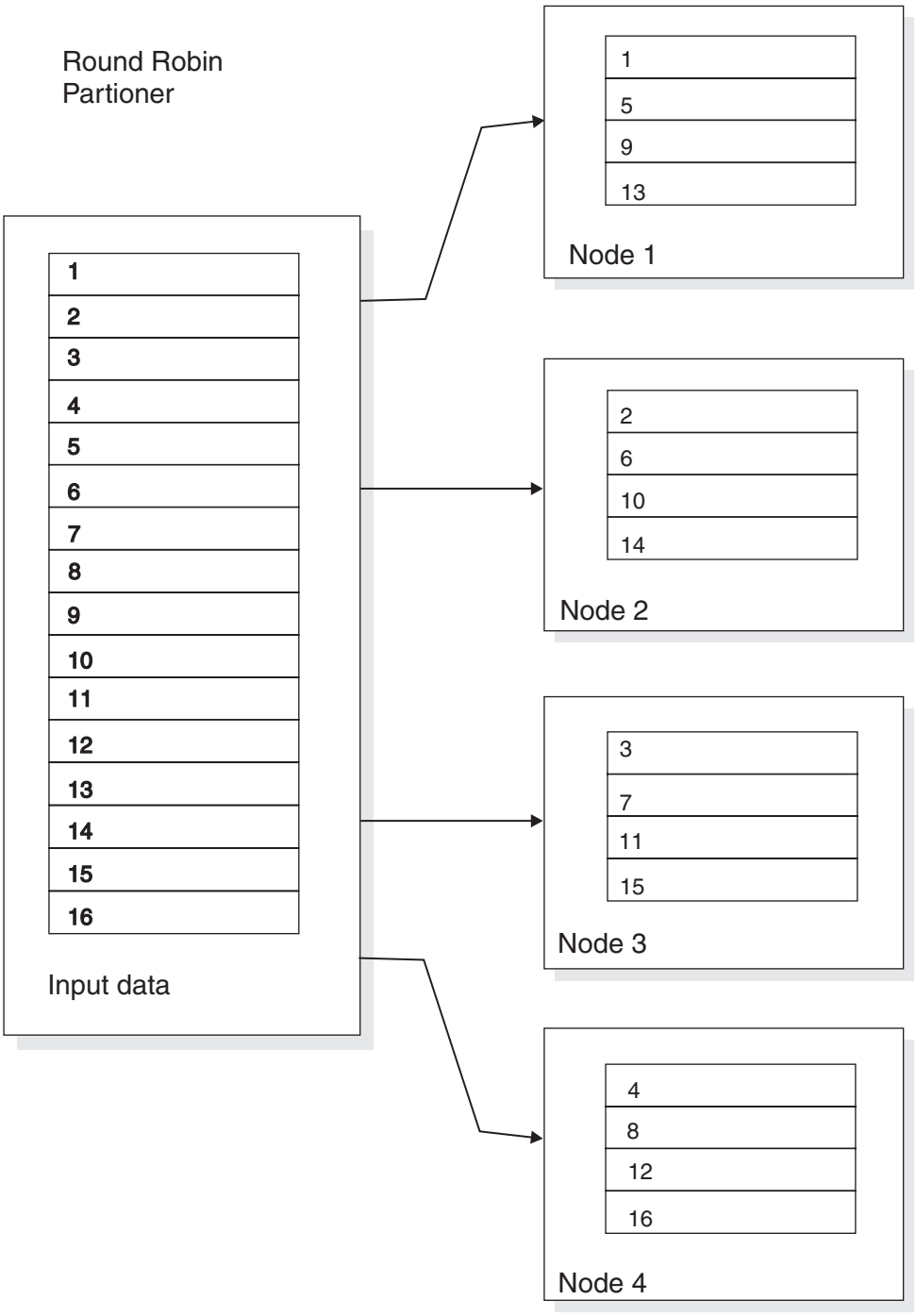
The aim of most partitioning operations is to end up with a set of partitions that are as near equal size as possible, ensuring an even load across your processors.

When performing some operations however, you will need to take control of partitioning to ensure that you get consistent results. A good example of this would be where you are using an aggregator stage to summarize your data. To get the answers you want (and need) you must ensure that related data is grouped together in the same partition before the summary operation is performed on that partition. InfoSphere DataStage lets you do this.

There are a number of different partitioning methods available, note that all these descriptions assume you are starting with sequential data. If you are repartitioning already partitioned data then there are some specific considerations:

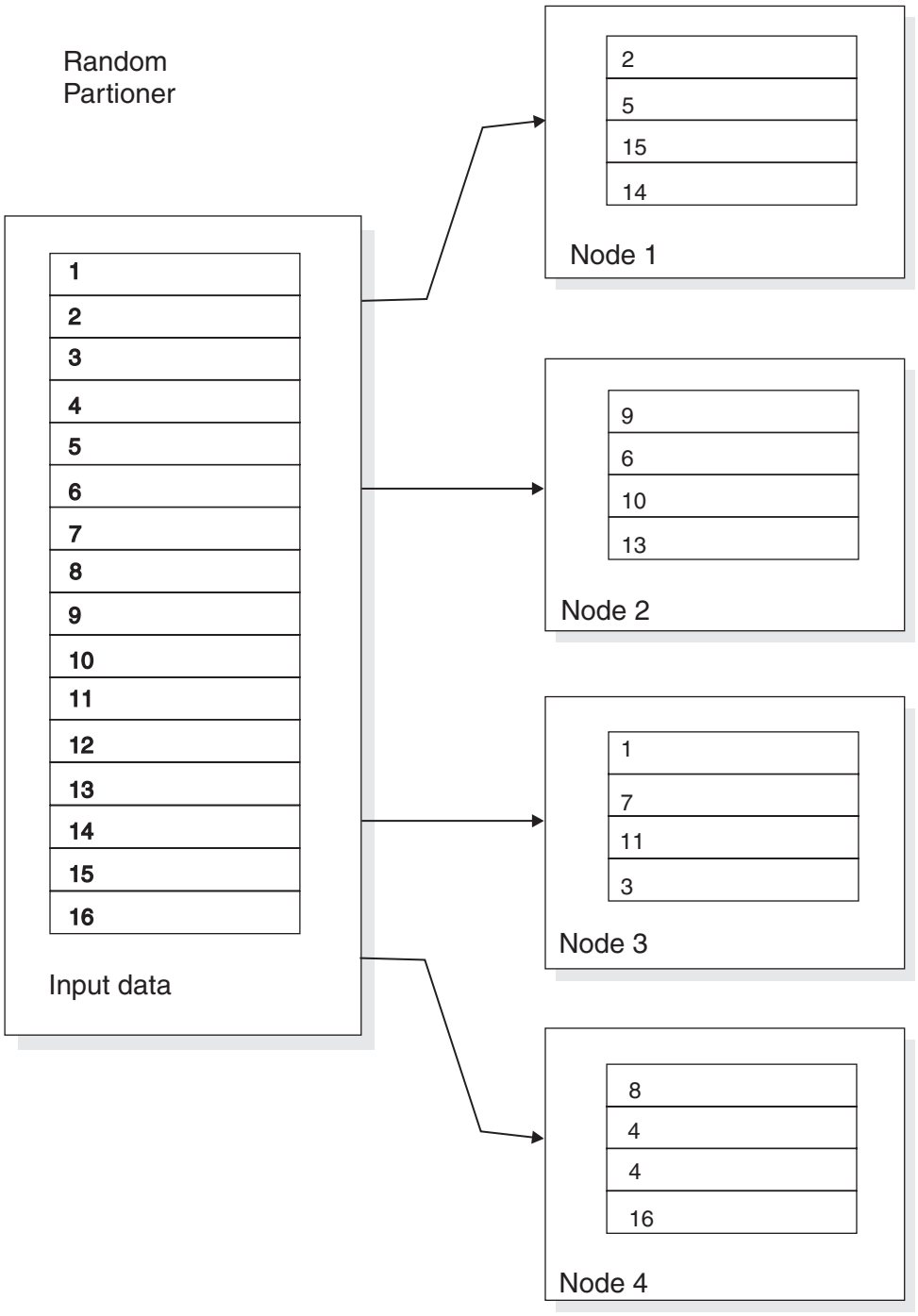
Round robin partitioner

The first record goes to the first processing node, the second to the second processing node, and so on. When InfoSphere 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 round robin method always creates approximately equal-sized partitions. This method is the one normally used when InfoSphere DataStage initially partitions data.



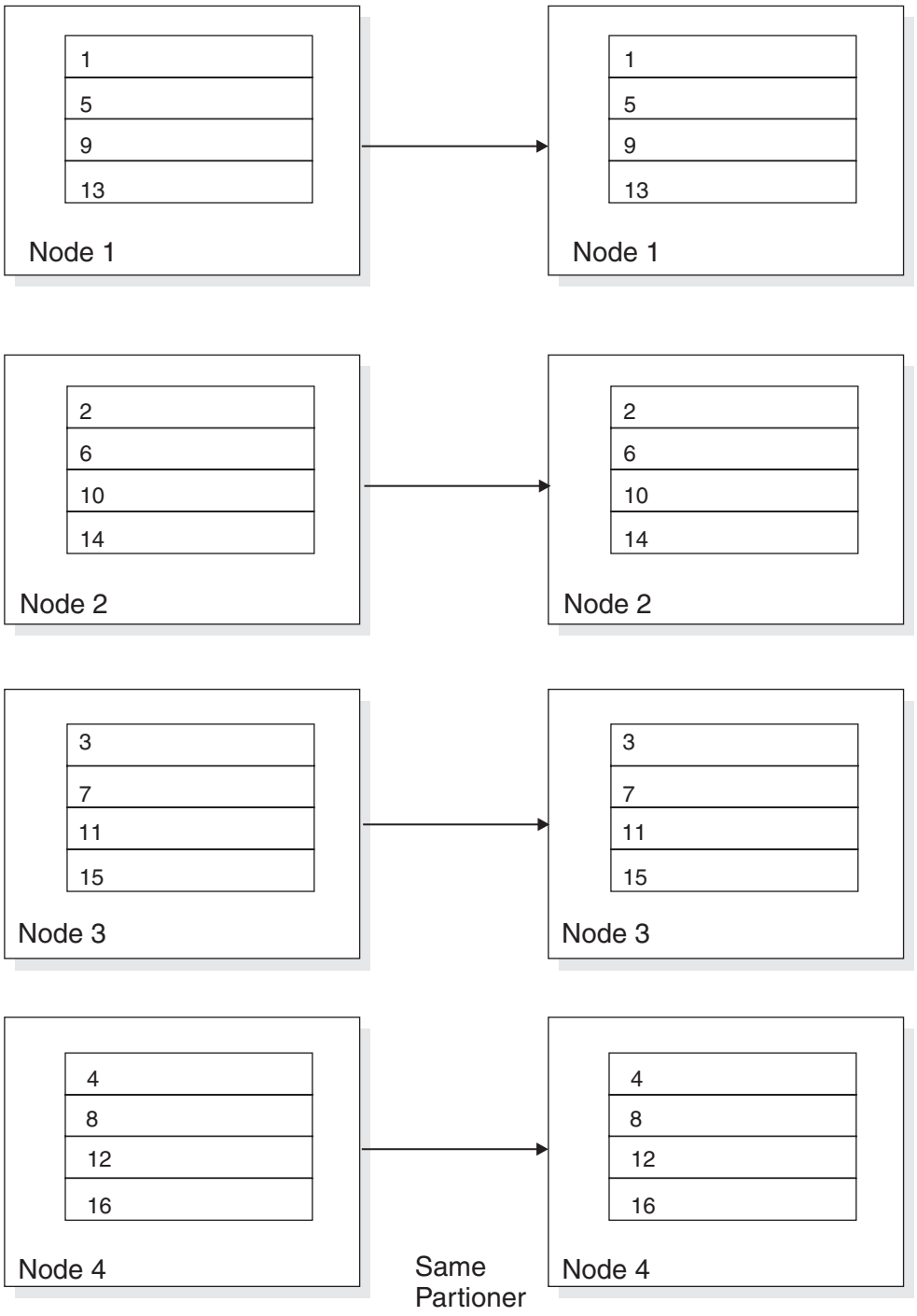
Random partitioner

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 partition. The random partitioning has a slightly higher overhead than round robin because of the extra processing required to calculate a random value for each record.



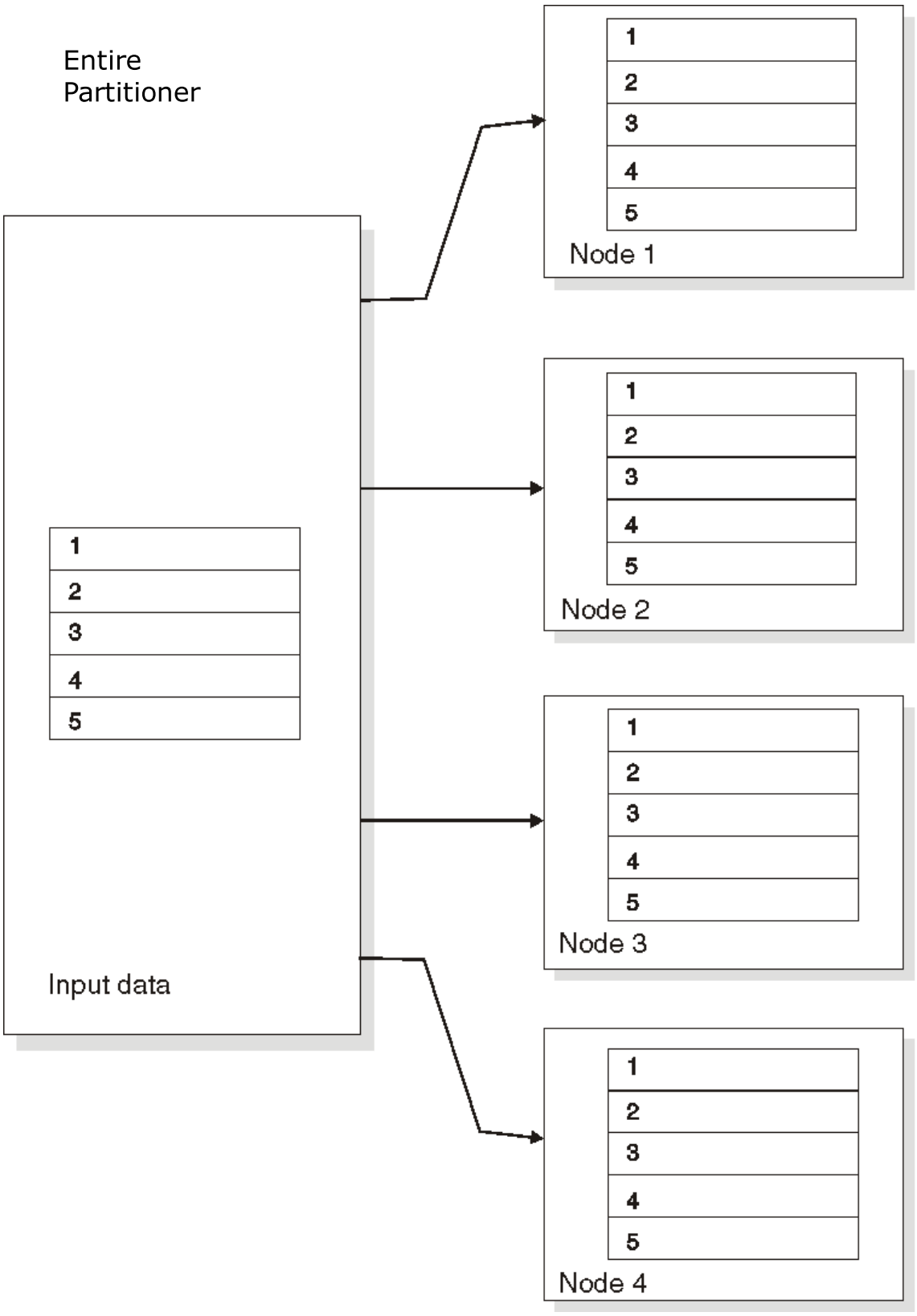
Same partitioner

The stage using the data set as input performs no repartitioning and takes as input the partitions output by the preceding stage. With this partitioning method, records stay on the same processing node; that is, they are not redistributed. Same is the fastest partitioning method. This is normally the method InfoSphere DataStage uses when passing data between stages in your job.



Entire partitioner

Every instance of a stage 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. You are most likely to use this partitioning method with stages that create lookup tables from their input.


Entire
Partitioner
1
2
3
4
5
Input data
1
2
3
4
5
Node 1
1
2
3
4
5
Node 2
1
2
3
4
5
Node 3
1
2
3
4
5
Node 4

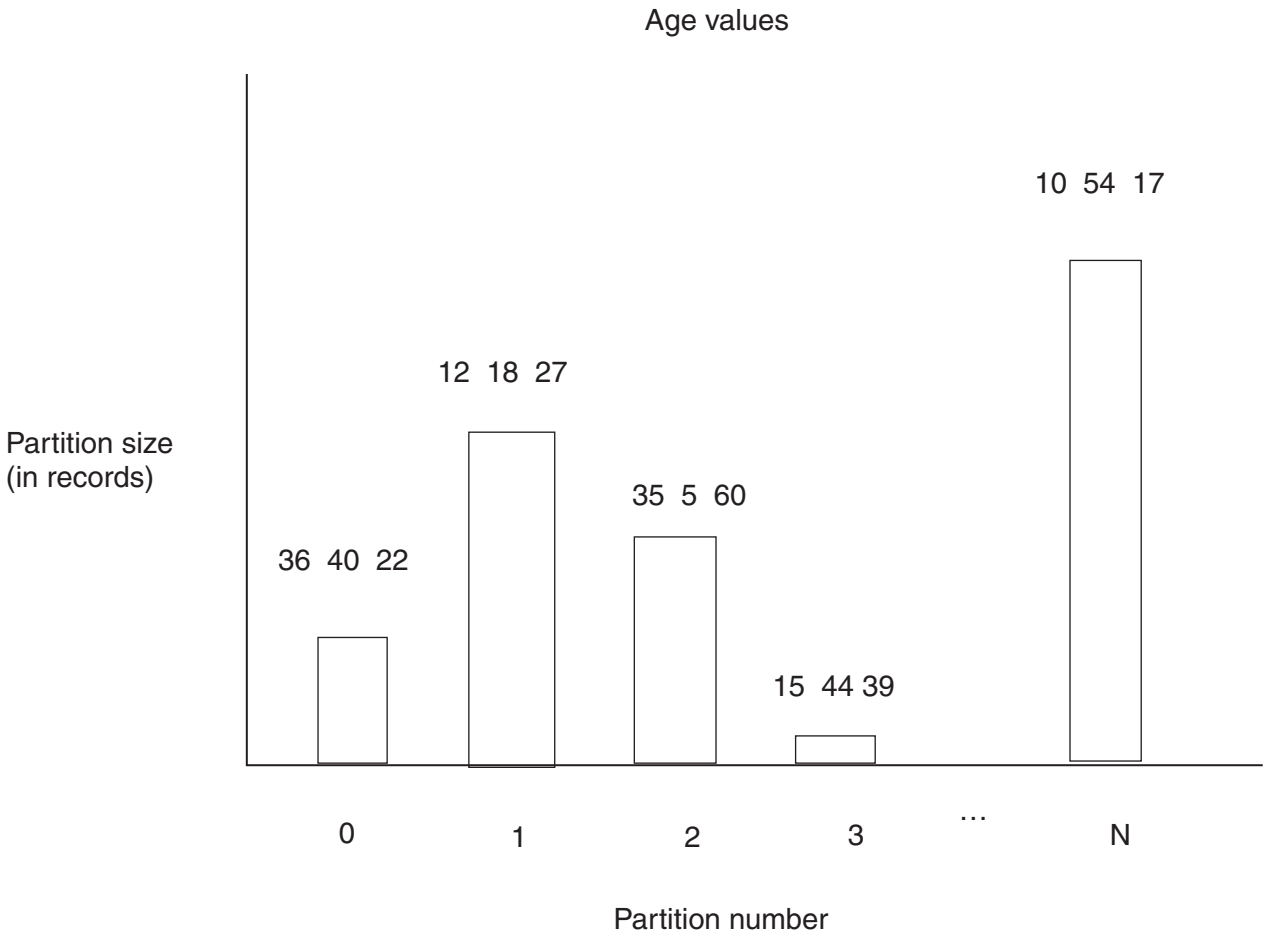
Hash partitioner

Partitioning is based on a function of one or more columns (the hash partitioning keys) in each record. The hash partitioner examines one or more fields of each input record (the hash key fields). Records with the same values for all hash key fields are assigned to the same processing node.

This method is useful for ensuring that related records are in the same partition, which might be a prerequisite for a processing operation. For example, for a remove duplicates operation, 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.

Hash partitioning does not necessarily result in an even distribution of data between partitions. 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 diagram 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.



As you can see, 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.

You must define a single primary partitioning key for the hash partitioner, and you might define as many secondary keys as are required by your job. Note, however, that each column 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 columns in the row.

You specify which columns are to act as hash keys on the Partitioning tab of the stage editor The data type of a partitioning key might be any data type except raw, subrecord, tagged aggregate, or vector. By default, the hash 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.

Modulus partitioner

Partitioning is based on a key column modulo the number of partitions. This method is similar to hash by field, but involves simpler computation.

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 a partitioner is executed on three processing nodes it has three partitions.

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 data is as follows:

Table 1. Input data

Column name	SQL type
bucket	Integer
date	Date

The bucket is specified as the key field, on which the modulus operation is calculated.

Here is the input data set. Each line represents a row:

Table 2. Input data set

bucket	date
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.

Table 3. Output data set

Partition 0	Partition 1	Partition 2	Partition 3
	61821 1960-06-27	21870 1960-01-01	64123 1960-03-30
	22677 1960-09-24	87702 1960-12-22	44919 1961-06-18
	47051961-12-13	47330 1961-03-21	
	88193 1962-03-12	90746 1961-09-15	

Here are three sample modulus operations corresponding to the values of three of the key fields:

- 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.

Range partitioner

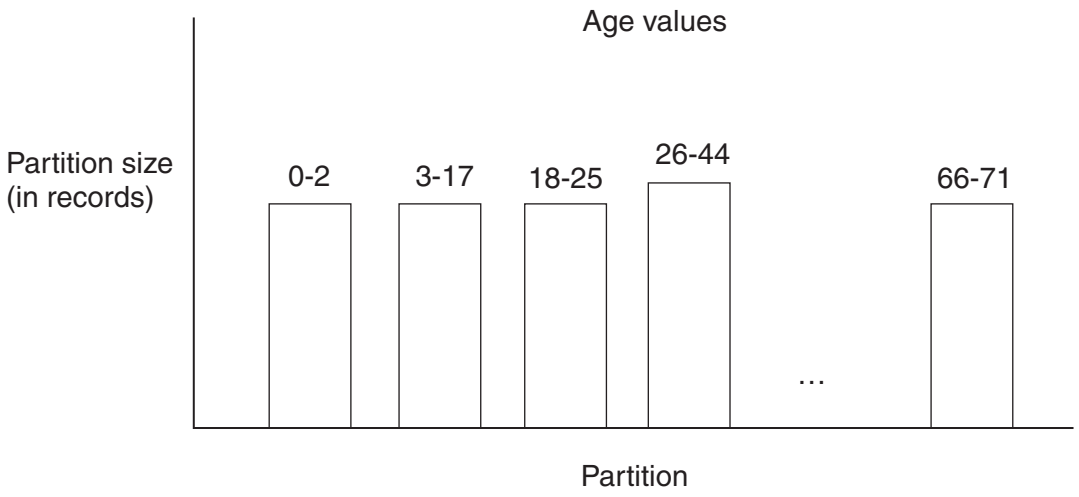
Divides a data set into approximately equal-sized partitions, each of which contains records with key columns within a specified range. This method is also useful for ensuring that related records are in the same partition.

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.

In order to use a range partitioner, you have to make a range map. You can do this using the Write Range Map stage, which is described in “Write Range Map stage” on page 540.

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.

An example of the results of a range partition is shown below. 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.



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 uses a range map to calculate partition boundaries. 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 round robin 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.

In order to perform range partitioning your job requires a write range map stage to calculate the range partition boundaries in addition to the stage that actually uses the range partitioner. The write range map stage 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. In order for the stage to determine the partition boundaries, you pass it a sorted sample of the data set to be range partitioned. From this sample, the stage can determine the appropriate partition boundaries for the entire data set.

When you come to actually partition your data, you specify the range map to be used by clicking on the property icon, next to the Partition type field, the Partitioning/Collection properties dialog box appears and allows you to specify a range map.

DB2 partitioner

Partition an input data set in the same way that DB2® would partition it. For example, if you use this method to partition an input data set containing update information for an existing DB2 table, records are assigned to the processing node containing the corresponding DB2 record. Then, during the execution of the parallel operator, both the input record and the DB2 table record are local to the processing node. Any reads and writes of the DB2 table would entail no network activity.

See the *DB2 Parallel Edition for AIX®, Administration Guide and Reference* for more information on DB2 partitioning.

To use DB2 partitioning on a stage, select a Partition type of DB2 on the Partitioning tab, then click the **Properties** button to the right. In the Partitioning/Collection properties dialog box, specify the details of the DB2 table whose partitioning you want to replicate).

Auto partitioner

The most common method you will see on the InfoSphere DataStage stages is Auto. This just means that you are leaving it to InfoSphere DataStage to determine the best partitioning method to use depending on the type of stage, and what the previous stage in the job has done. Typically InfoSphere DataStage would use round robin when initially partitioning data, and same for the intermediate stages of a job.

Collecting

Collecting is the process of joining the multiple partitions of a single data set back together again into a single partition. There are various situations where you might want to do this. There might be a stage in your job that you want to run sequentially rather than in parallel, in which case you will need to collect all your partitioned data at this stage to make sure it is operating on the whole data set.

Similarly, at the end of a job, you might want to write all your data to a single database, in which case you need to collect it before you write it.

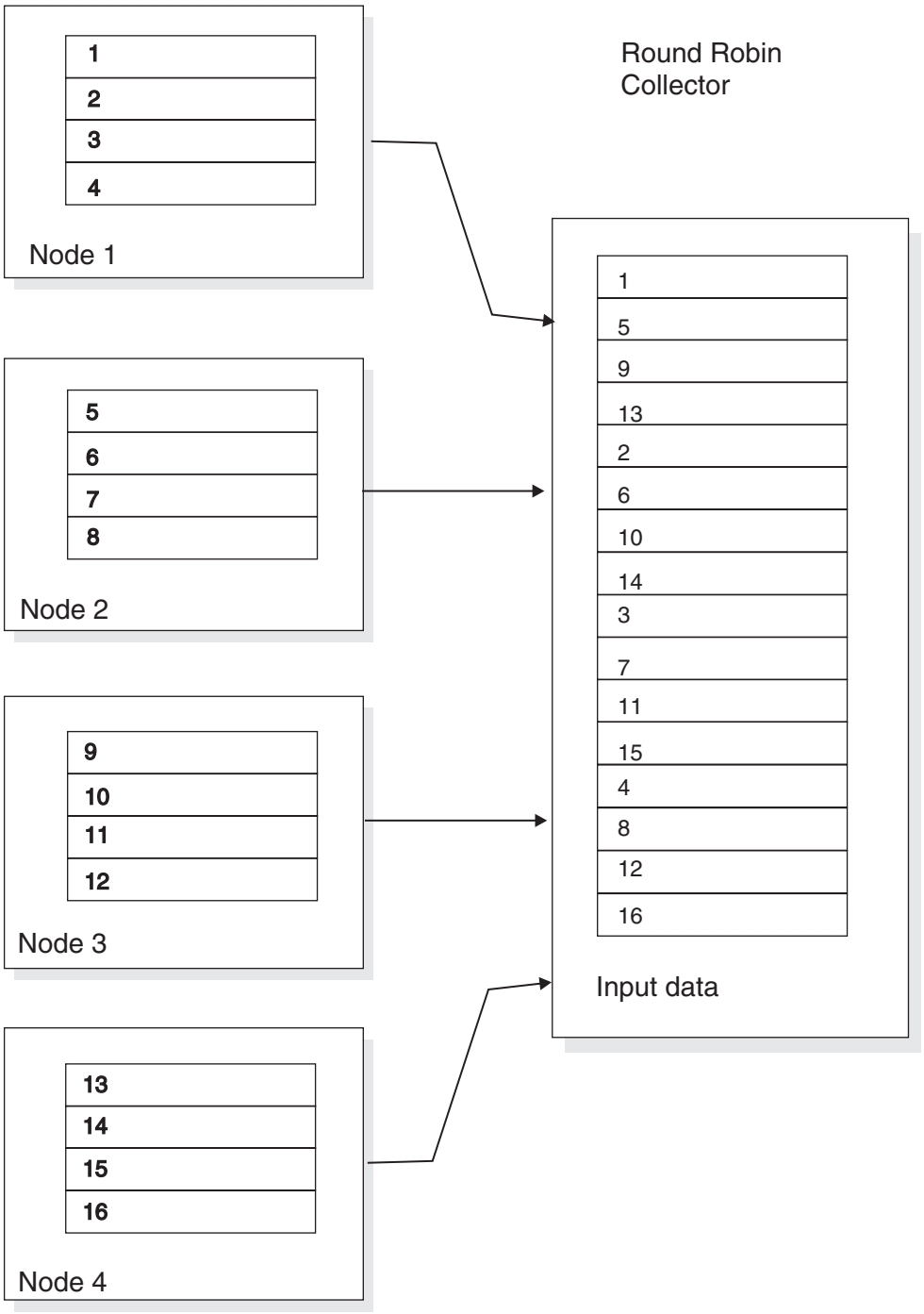
There might be other cases where you do not want to collect the data at all. For example, you might want to write each partition to a separate flat file.

Just as for partitioning, in many situations you can leave DataStage to work out the best collecting method to use. There are situations, however, where you will want to explicitly specify the collection method.

Note that collecting methods are mostly non-deterministic. That is, if you run the same job twice with the same data, you are unlikely to get data collected in the same order each time. If order matters, you need to use the sorted merge collection method.

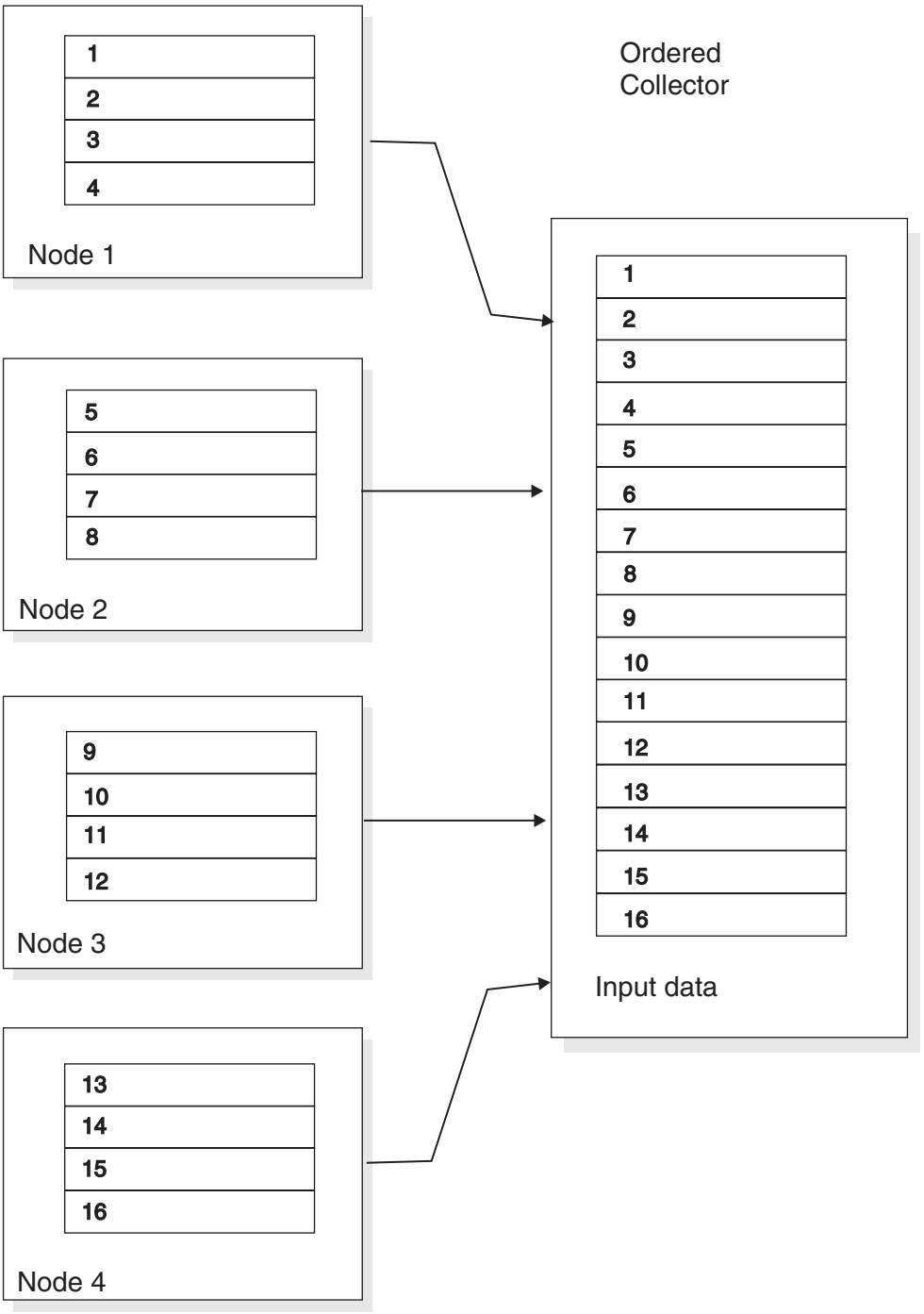
Round robin collector

Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, starts over. After reaching the final record in any partition, skips that partition in the remaining rounds.



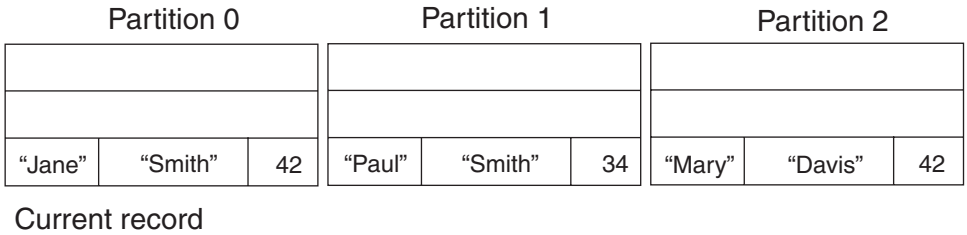
Ordered collector

Reads all records from the first partition, then all records from the second partition, and so on. This collection method preserves the order of totally sorted input data sets. In a totally sorted data set, both the records in each partition and the partitions themselves are ordered. This might be useful as a preprocessing action before exporting a sorted data set to a single data file.

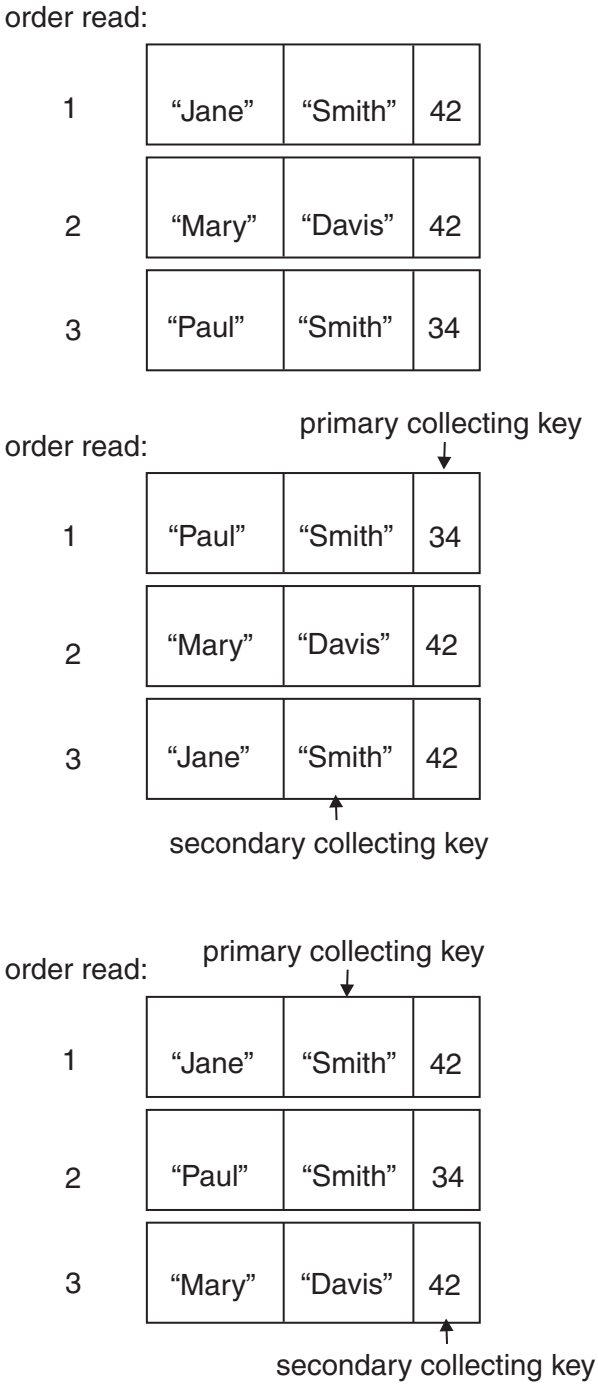


Sorted merge collector

Read records in an order based on one or more columns of the record. The columns used to define record order are called collecting keys. Typically, you use the sorted merge collector with a partition-sorted data set (as created by a sort stage). In this case, you specify as the collecting key fields those fields you specified as sorting key fields to the sort stage. For example, the figure below shows the current record in each of three partitions of an input data set to the 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 figure shows the order of the three records read by the sort merge collector, based on different combinations of collecting keys.



You must define a single primary collecting key for the sort merge collector, and you might define as many secondary keys as are required by your job. Note, however, that each column 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 columns in the row. You define the keys on the Partitioning tab, and the key you define first is the primary key.

The data type of a collecting key can be any type except raw, subrec, tagged, or vector.

By default, the sort merge 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.

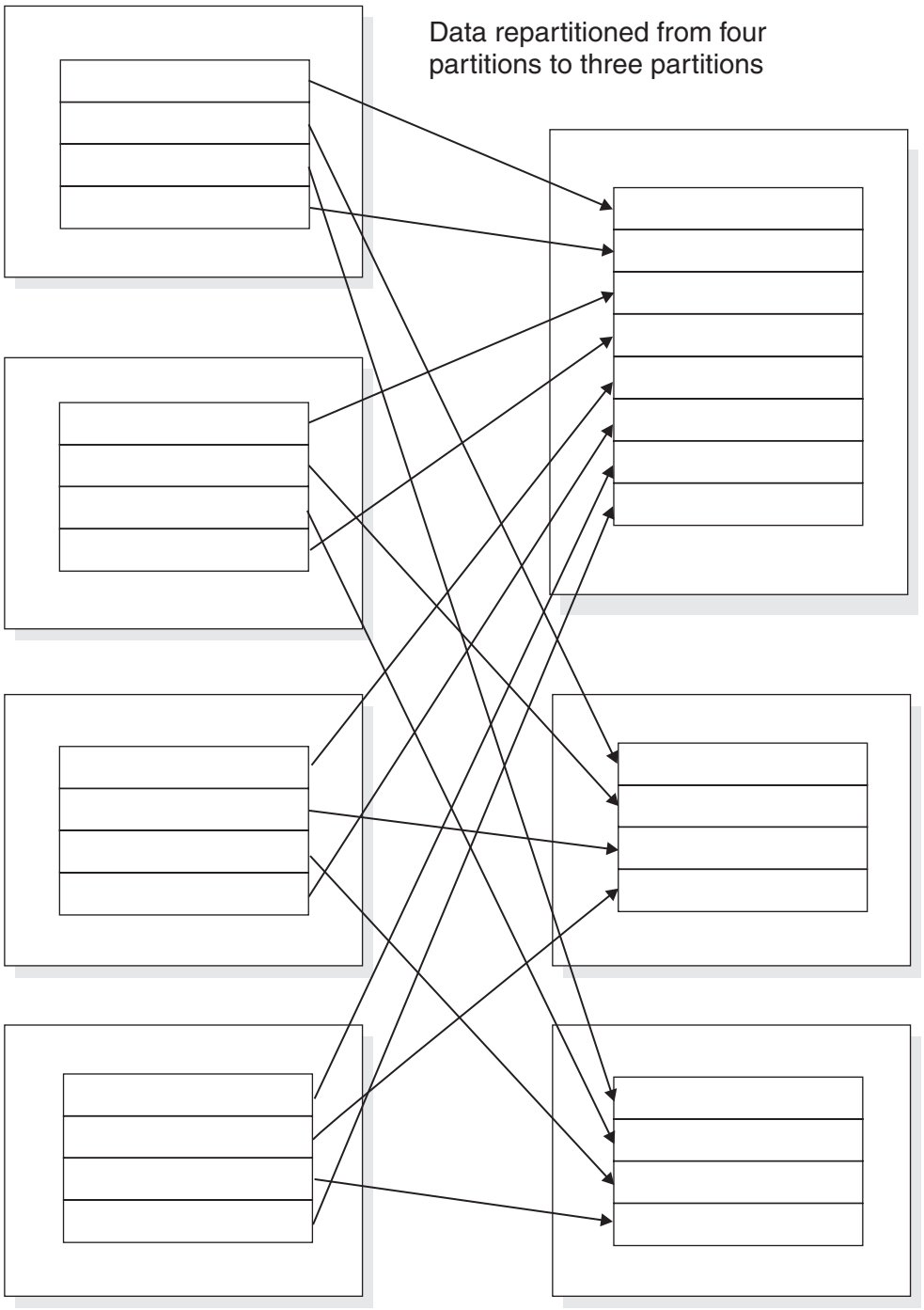
Auto collector

The most common method you will see on the parallel stages is Auto. This normally means that InfoSphere DataStage will eagerly read any row from any input partition as it becomes available, but if it detects that, for example, the data needs sorting as it is collected, it will do that. This is the fastest collecting method.

Repartitioning data

If you decide you need to repartition data within your parallel job there are some particular considerations as repartitioning can affect the balance of data partitions.

For example, if you start with four perfectly balanced partitions and then subsequently repartition into three partitions, you will lose the perfect balance and be left with, at best, near perfect balance. This is true even for the round robin method; this only produces perfectly balanced partitions from a sequential data source. The reason for this is illustrated below. Each node partitions as if it were a single processor with a single data set, and will always start writing to the first target partition. In the case of four partitions repartitioning to three, more rows are written to the first target partition. With a very small data set the effect is pronounced; with a large data set the partitions tend to be more balanced.



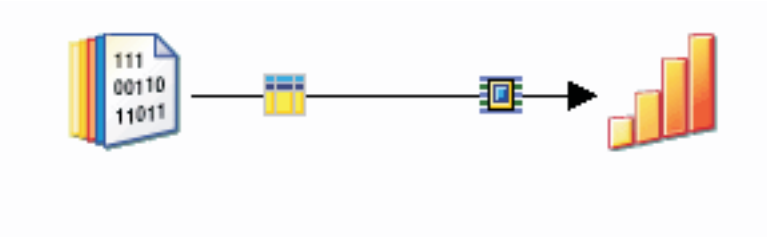
The mechanics of partitioning and collecting

This section gives a quick guide to how partitioning and collecting is represented in a parallel job.

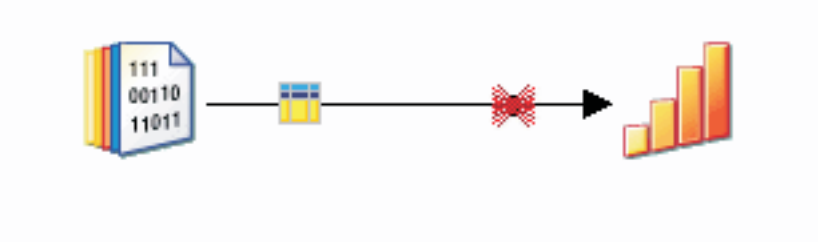
Partitioning icons

Each parallel stage in a job can partition or repartition incoming data before it operates on it. Equally it can just accept the partitions that the data comes in. There is an icon on the input link to a stage which shows how the stage handles partitioning.

In most cases, if you just lay down a series of parallel stages in a DataStage job and join them together, the auto method will determine partitioning. This is shown on the canvas by the auto partitioning icon:

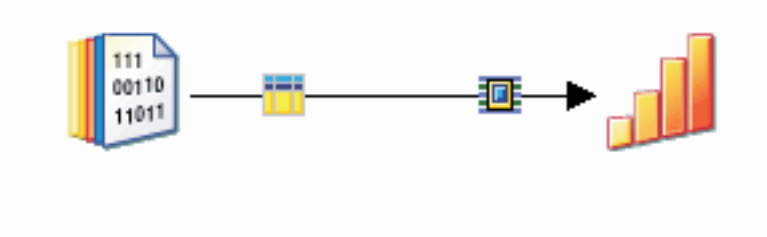


In some cases, stages have a specific partitioning method associated with them that cannot be overridden. It always uses this method to organize incoming data before it processes it. In this case an icon on the input link tells you that the stage is repartitioning data:



If you have a data link from a stage running sequentially to one running in parallel the following icon is shown to indicate that the data is being partitioned:

You can specify that you want to accept the existing data partitions by choosing a partitioning method of same. This is shown by the following icon on the input link:



Partitioning methods are set on the Partitioning tab of the Inputs pages on a stage editor.

Preserve partitioning flag

A stage can also request that the next stage in the job preserves whatever partitioning it has implemented. It does this by setting the preserve partitioning flag for its output link. Note, however, that the next stage might ignore this request.

In most cases you are best leaving the preserve partitioning flag in its default state. The exception to this is where preserving existing partitioning is important. The flag will not prevent repartitioning, but it will warn you that it has happened when you run the job. If the Preserve Partitioning flag is cleared, this means that the current stage doesn't care what the next stage in the job does about partitioning. On some stages, the Preserve Partitioning flag can be set to Propagate. In this case the stage sets the flag on its output link according to what the previous stage in the job has set. If the previous job is also set to Propagate, the setting from the stage before is used and so on until a Set or Clear flag is encountered earlier in the job. If the stage has multiple inputs and has a flag set to Propagate, its Preserve Partitioning flag is set if it is set on any of the inputs, or cleared if all the inputs are clear.

Collector icon

A stage in the job which is set to run sequentially will need to collect partitioned data before it operates on it. There is an icon on the input link to a stage which shows that it is collecting data:

Sorting data

You will probably have requirements in your parallel jobs to sort data. InfoSphere DataStage has a sort stage, which allows you to perform complex sorting operations. There are situations, however, where you require a fairly simple sort as a precursor to a processing operation. For these purposes, InfoSphere DataStage allows you to insert a sort operation in most stage types for incoming data. You do this by selecting the **Sorting** option on the Input page Partitioning tab . When you do this you can specify:

- Sorting keys. The field(s) on which data is sorted. You must specify a primary key, but you can also specify any number of secondary keys. The first key you define is taken as the primary.
- Stable sort (this is the default and specifies that previously sorted data sets are preserved).
- Unique sort (discards records if multiple records have identical sorting key values).
- Case sensitivity.
- Sort direction. Sorted as EBCDIC (ASCII is the default).

If you have NLS enabled, you can also specify the collate convention used.

Some InfoSphere DataStage operations require that the data they process is sorted (for example, the Merge operation). If InfoSphere DataStage detects that the input data set is not sorted in such a case, it will automatically insert a sort operation in order to enable the processing to take place unless you have explicitly specified otherwise.

Data sets

Inside a InfoSphere DataStage parallel job, data is moved around in data sets. These carry meta data with them, both column definitions and information about the configuration that was in effect when the data set was created. If for example, you have a stage which limits execution to a subset of available nodes, and the data set was created by a stage using all nodes, InfoSphere DataStage can detect that the data will need repartitioning.

If required, data sets can be landed as persistent data sets, represented by a Data Set stage (see "Data Set Stage.") This is the most efficient way of moving data between linked jobs. Persistent data sets are stored in a series of files linked by a control file (note that you should not attempt to manipulate these files using UNIX tools such as RM or MV. Always use the tools provided with InfoSphere DataStage).

Note: The example screenshots in the individual stage descriptions often show the stage connected to a Data Set stage. This does not mean that these kinds of stage can only be connected to Data Set stages.

Metadata

Metadata is information about data. It describes the data flowing through your job in terms of column definitions, which describe each of the fields making up a data record.

InfoSphere DataStage has two alternative ways of handling metadata, through table definitions, or through Schema files. By default, parallel stages derive their meta data from the columns defined on the Outputs or Input page Column tab of your stage editor. Additional formatting information is supplied, where needed, by a Formats tab on the Outputs or Input page. In some cases you can specify that the stage uses a schema file instead by explicitly setting a property on the stage editor and specify the name and location of the schema file. Note that, if you use a schema file, you should ensure that runtime column propagation is turned on. Otherwise the column definitions specified in the stage editor will always override any schema file.

Where is additional formatting information needed? Typically this is where you are reading from, or writing to, a file of some sort and InfoSphere DataStage needs to know more about how data in the file is formatted.

You can specify formatting information on a row basis, where the information is applied to every column in every row in the dataset. This is done from the Formats tab (the Formats tab is described with the stage editors that support it; for example, for Sequential files, see page Input Link Format Tab). You can also specify formatting for particular columns (which overrides the row formatting) from the Edit Column Metadata dialog box for each column (see page Field Level).

Runtime column propagation

InfoSphere DataStage is also flexible about meta data. It can cope with the situation where meta data isn't fully defined. You can define part of your schema and specify that, if your job encounters extra columns that are not defined in the meta data when it actually runs, it will adopt these extra columns and propagate them through the rest of the job. This is known as runtime column propagation (RCP). This can be enabled for a project via the Administrator client, and set for individual links via the Output Page Columns tab for most stages, or in the Output page General tab for Transformer stages. You should always ensure that runtime column propagation is turned on if you want to use schema files to define column meta data.

Table definitions

A table definition is a set of related columns definitions that are stored in the Repository. These can be loaded into stages as and when required.

You can import a table definition from a data source via the Designer. You can also edit and define new table definitions in the Designer (see *InfoSphere DataStage Designer Client Guide*). If you want, you can edit individual column definitions once you have loaded them into your stage.

You can also simply type in your own column definition from scratch on the Outputs or Input page **Column** tab of your stage editor. When you have entered a set of column definitions you can save them as a new table definition in the Repository for subsequent reuse in another job.

Schema files and partial schemas

You can also specify the meta data for a stage in a plain text file known as a schema file. This is not stored in the Repository but you could, for example, keep it in a document management or source code control system, or publish it on an intranet site.

The format of schema files is described in Schemas.

Note: If you are using a schema file on an NLS system, the schema file needs to be in UTF-8 format. It is, however, easy to convert text files between two different maps with a InfoSphere DataStage job. Such a job would read data from a text file using a Sequential File stage and specifying the appropriate character set on the NLS Map page. It would write the data to another file using a Sequential File stage, specifying the UTF-8 map on the NLS Map page.

Some parallel job stages allow you to use a partial schema. This means that you only need define column definitions for those columns that you are actually going to operate on. Partial schemas are also described in Schemas.

Remember that you should turn runtime column propagation on if you intend to use schema files to define column metadata.

Data types

When you work with parallel job column definitions, you will see that they have an SQL type associated with them. This maps onto an underlying data type which you use when specifying a schema via a file, and which you can view in the Parallel tab of the Edit Column Meta Data dialog box. The underlying data type is what a parallel job data set understands. The following table summarizes the underlying data types that columns definitions can have:

Table 4. Underlying data types

SQL Type	Underlying Data Type	Size	Description
Date	date	4 bytes	Date with month, day, and year
Decimal Numeric	decimal	(Roundup(p)+1)/2	Packed decimal, compatible with IBM packed decimal format
Float Real	sfloat	4 bytes	IEEE single-precision (32-bit) floating point value
Double	dfloat	8 bytes	IEEE double-precision (64-bit) floating point value
TinyInt	int8 uint8	1 byte	Signed or unsigned integer of 8 bits (Extended (unsigned) option for unsigned)
SmallInt	int16 uint16	2 bytes	Signed or unsigned integer of 16 bits (Extended (unsigned) option for unsigned)
Integer	int32 uint32	4 bytes	Signed or unsigned integer of 32 bits (Extended (unsigned) option for unsigned)

Table 4. Underlying data types (continued)

SQL Type	Underlying Data Type	Size	Description
BigInt[1]	int64 uint64	8 bytes	Signed or unsigned integer of 64 bits (Extended (unsigned) option for unsigned)
Binary Bit LongVarBinary VarBinary	raw	1 byte per character	Untypes collection, consisting of a fixed or variable number of contiguous bytes and an optional alignment value
Unknown Char LongVarChar VarChar	string	1 byte per character	ASCII character string of fixed or variable length (without the extended(Unicode) option selected)
NChar NVarChar LongNVarChar	ustring	multiple bytes per character	ASCII character string of fixed or variable length (without the extended(Unicode) option selected)
Char LongVarChar VarChar	ustring	multiple bytes per character	ASCII character string of fixed or variable length (with the extended(Unicode) option selected)
Char	subrec	sum of lengths of subrecord fields	Complex data type comprising nested columns
Char	tagged	sum of lengths of subrecord fields	Complex data type comprising tagged columns, of which one can be referenced when the column is used
Time	time	5 bytes	Time of day, with resolution of seconds.
Time	time(microseconds)	5 bytes	Time of day, with resolution of microseconds (Extended (Microseconds) option selected).
Timestamp	timestamp	9 bytes	Single field containing both date and time value
Timestamp	timestamp(microseconds)	9 bytes	Single field containing both date and time value, with resolution of microseconds (Extended (Microseconds) option selected).

[1]BigInt values map to long long integers on all supported platforms except Tru64 where they map to longer integers. For all platforms except Tru64, the c_format is:

```
'%[padding_character][integer]lld'
```

Because Tru64 supports real 64-bit integers, its c_format is:

```
%[padding_character][integer]ld'
```

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.

For this example c_format specification: '%09lld' the padding character is zero (0), and the integers 123456 and 123456789 are printed out as 000123456 and 123456789.

When you work with mainframe data using the CFF stage, the data types are as follows:

Table 5. Mainframe data types

COBOL Data Type	Size	COBOL Usage Representation	Underlying Data Type	Property
binary, native binary	2 bytes	S9(1-4) COMP/COMP-5	int16	
binary, native binary	4 bytes	S9(5-9) COMP/COMP-5	int32	
binary, native binary	8 bytes	S9(10-18) COMP/COMP-5	int64	
binary, native binary	2 bytes	9(1-4) COMP/COMP-5	uint16	
binary, native binary	4 bytes	9(5-9) COMP/COMP-5	uint32	
binary, native binary	8 bytes	9(10-18) COMP/COMP-5	uint64	
character	*n* bytes	X(*n*)	string[*n*]	
character for filler	*n* bytes	X(*n*)	raw(*n*)	
varchar	*n* bytes	X(*n*)	string[max=n]	
decimal	(*x*+*y*)/2+1 bytes	9(*x*)V9(*y*)COMP-3	decimal[*x*+*y*,*y*]	packed
decimal	(*x*+*y*)/2+1 bytes	S9(*x*)V9(*y*)COMP-3	decimal[*x*+*y*,*y*]	packed
display_numeric	*x*+*y* bytes	9(*x*)V9(*y*)	decimal[*x*+*y*,*y*] or string[*x*+*y*]	zoned
display_numeric	*x*+*y* bytes	S9(*x*)V9(*y*)	decimal[*x*+*y*,*y*] or string[*x*+*y*]	zoned, trailing
display_numeric	*x*+*y* bytes	S9(*x*)V9(*y*) sign is trailing	decimal[*x*+*y*,*y*]	zoned, trailing
display_numeric	*x*+*y* bytes	S9(*x*)V9(*y*) sign is leading	decimal[*x*+*y*,*y*]	zoned, leading
display_numeric	*x*+*y*+1 bytes	S9(*x*)V9(*y*) sign is trailing separate	decimal[*x*+*y*,*y*]	separate, trailing
display_numeric	*x*+*y*+1 bytes	S9(*x*)V9(*y*) sign is leading separate	decimal[*x*+*y*,*y*]	separate, leading
float	4 bytes 8 bytes	COMP-1 COMP-2	sfloat dfloat	floating point

Table 5. Mainframe data types (continued)

COBOL Data Type	Size	COBOL Usage Representation	Underlying Data Type	Property
graphic_n, graphic_g	*n**2 bytes	N(*n*) or G(*n*) DISPLAY-1	ustring[*n*]	
vargraphic_g/n	*n**2 bytes	N(*n*) or G(*n*) DISPLAY-1	ustring[max=*n*]	
group			subrec	

Strings and ustrings

If you have NLS enabled, parallel jobs support two types of underlying character data types: strings and ustrings. String data represents unmapped bytes, ustring data represents full Unicode (UTF-16) data.

The Char, VarChar, and LongVarChar SQL types relate to underlying string types where each character is 8-bits and does not require mapping because it represents an ASCII character. You can, however, specify that these data types are extended, in which case they are taken as ustrings and do require mapping. (They are specified as such by selecting the **Extended** check box for the column in the Edit Meta Data dialog box.) An **Extended** field appears in the columns grid, and extended Char, VarChar, or LongVarChar columns have `Unicode' in this field. The NChar, NVarChar, and LongNVarChar types relate to underlying ustring types so do not need to be explicitly extended.

Complex data types

Parallel jobs support three complex data types:

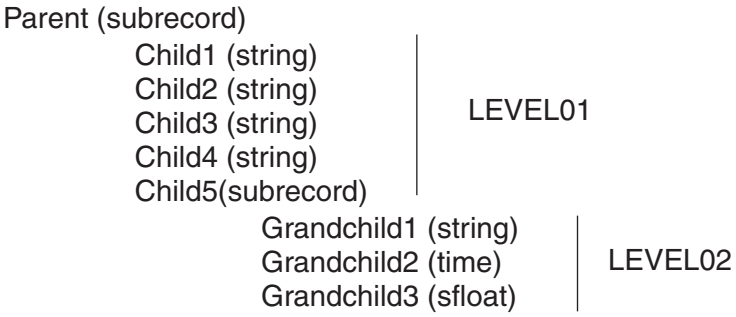
- Subrecords
- Tagged subrecords
- Vectors

When referring to complex data in InfoSphere DataStage column definitions, you can specify fully qualified column names, for example:

```
Parent.Child5.Grandchild2
```

Subrecords

A subrecord is a nested data structure. The column with type subrecord does not itself define any storage, but the columns it contains do. These columns can have any data type, and you can nest subrecords one within another. The LEVEL property is used to specify the structure of subrecords. The following diagram gives an example of a subrecord structure.



Tagged subrecord

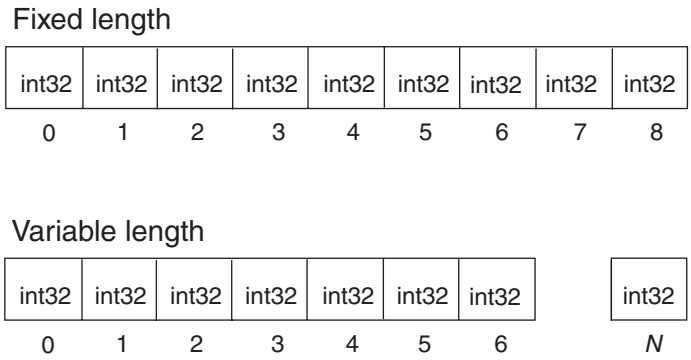
This is a special type of subrecord structure, it comprises a number of columns of different types and the actual column is ONE of these, as indicated by the value of a tag at run time. The columns can be of any type except subrecord or tagged. The following diagram illustrates a tagged subrecord.

Parent (tagged)
- Child1 (string)
- Child2 (int8)
- Child3 (raw)

Tag = Child1, so column has data type of string

Vector

A vector is a one dimensional array of any type except tagged. All the elements of a vector are of the same type, and are numbered from 0. The vector can be of fixed or variable length. For fixed length vectors the length is explicitly stated, for variable length ones a property defines a link field which gives the length at run time. The following diagram illustrates a vector of fixed length and one of variable length.



Date and time formats

Parallel jobs provide flexible handling of date and time formats.

You use formatting strings at various places in parallel jobs to specify the format of dates, times, and timestamps.

Date formats

Format strings are used to control the format of dates.

A date format string can contain one or a combination of the following elements:

Table 6. 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

Table 6. Date format tags (continued)

Tag	Variable width availability	Description	Value range	Options
%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 (%). Separate the string's components with a literal character.

The default date format is %yyyy-%mm-%dd.

Where indicated the tags can represent variable-width date 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 text elements such as day or month name in uppercase text on output.

w Use this option to render text elements such as day or month name in lowercase text on output.

t Use this option to render text elements such as day or month name in titlecase (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)-%yyyy
21-Augus-2006

%dd-%(mmmm,-10)-%yyyy
21-August    -2005

%dd-%(mmmm,+10)-%yyyy
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 *IBM InfoSphere DataStage and QualityStage Globalization 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 septembre 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 7. 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.

This property is mutually exclusive with days_since, text, and julian.

You can include literal text in your date format, for example as separators. 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

Time formats

Format strings are used to control the format of times.

The possible components of the time format string are given in the following table:

Table 8. 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

When you specify a time string, prefix each component of the format string with the percent symbol. Separate the string's components with a literal character.

The default time format is %hh:%nn:%ss.

Where indicated the tags can represent variable-width date elements. Variable-width time 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 *IBM InfoSphere DataStage and QualityStage Globalization 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 9. 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

Timestamp formats

Format strings are used to control the format of timestamps.

The timestamp format is the date format and the time format combined. The two formats can be in any order and their elements can be mixed. The formats are described in "Date formats" on page 31 and "Time formats" on page 35.

You must prefix each component of the format string with the percent symbol (%).

Incorporating server job functionality

You can incorporate Server job functionality in your Parallel jobs by the use of Server Shared Container stages. This allows you to, for example, use Server job plug-in stages to access data source that are not directly supported by Parallel jobs. (Some plug-ins have parallel versions that you can use directly in a parallel job.)

You create a new shared container in the Designer, add Server job stages as required, and then add the Server Shared Container to your Parallel job and connect it to the Parallel stages. Server Shared Container stages used in Parallel jobs have extra pages in their Properties dialog box, which enable you to specify details about parallel processing and partitioning and collecting data.

You can only use Server Shared Containers in this way on SMP systems (not MPP or cluster systems).

The following limitations apply to the contents of such Server Shared Containers:

- There must be zero or one container inputs, zero or more container outputs, and at least one of either.
- There can be no disconnected flows - all stages must be linked to the input or an output of the container directly or via an active stage. When the container has an input and one or more outputs, each stage must connect to the input and at least one of the outputs.
- There can be no synchronization by having a passive stage with both input and output links.

For details on how to use Server Shared Containers, see in *InfoSphere DataStage Designer Client Guide*. This also tells you how to use Parallel Shared Containers, which enable you to package parallel job functionality in a reusable form.

Chapter 3. Parallel Jobs and NLS

These topics give details about NLS in InfoSphere DataStage parallel jobs. They cover:

- Maps and locales available in parallel jobs
- Considerations about character data in parallel jobs
- How to use maps and locales in parallel jobs
- Creating new maps for parallel jobs
- Creating new locales for parallel jobs

How NLS Mode Works

NLS mode works by using two types of *character set*:

- The NLS *internal character set*
- *External character sets* that cover the world's different languages

In NLS mode, InfoSphere DataStage maps between the two character sets when it's needed.

The mechanism for handling NLS differs for parallel and server jobs. They each use a different internal character set, so each uses a different set of maps for converting data. Note that it is certain types of string (that is, character) data that needs mapping, purely numeric data types never require it.

Parallel and server jobs also use different locales.

Internal Character Sets

The internal character set can represent at least 64,000 characters. Each character in the internal character set has a unique *code point*. This is a number that is by convention represented in hexadecimal format. You can use this number to represent the character in programs. InfoSphere DataStage easily stores many languages.

The NLS internal character sets conform to the Unicode standard. The Unicode consortium specify a number of ways to represent code points, called Unicode Transformation Formats (UTF). Server jobs use UTF-8, parallel jobs use UTF-16.

Because the two types of job use different internal character sets, a different set of maps are provided for conversion to and from each one (although equivalents to commonly used server job maps are provided for parallel jobs).

For more information about Unicode, see the Unicode Consortium's World Wide Web page at **http://www.unicode.org**.

Mapping

When you need to transform or transfer data, NLS maps the data to or from the external character set you want to use. NLS includes map tables for many of the character sets used in the world (see the list in *IBM InfoSphere DataStage and QualityStage Globalization Guide*). You can specify mapping at different levels within InfoSphere DataStage:

- A project-wide default. In theInfoSphere DataStage and QualityStage Administrator client you specify a default map for all server jobs in a project, and a default map for all parallel jobs in a project.
- A job default. In the InfoSphere DataStage and QualityStage Designer client, you can specify a default map used by a particular job that overrides the project default.

- A stage map. Certain parallel and server stages allow you to specify that they use a particular map. This overrides both the project default and the job detail.
- A column map. Certain parallel and server stages support per-column mapping. This allows you to specify a separate map for particular data columns. This overrides the project default, job default, and stage maps.

If your files contain only ASCII 7-bit characters, they need not be mapped.

Locales

An InfoSphere DataStage NLS *locale* is a set of *national conventions*. A locale is viewed as a separate entity from a character set. You need to consider the language, character set, and conventions for data formatting that one or more groups of people use. You define the character set independently, although for national conventions to work correctly, you must also use the appropriate character sets. For example, Venezuela and Ecuador both use Spanish as their language, but have different data formatting conventions.

Locales do not respect national boundaries. One country can use several locales, for example, Canada uses two and Belgium uses three. Several countries can use one locale, for example, a multinational business could define a worldwide locale to use in all its offices. See *IBM InfoSphere DataStage and QualityStage Globalization Guide* for a list of all the locales that are supplied with InfoSphere DataStage and the territories and languages associated with them.

Server jobs allow you to choose locales separately for several different aspects of National conventions:

- The format for times and dates
- The format for displaying numbers
- How to display monetary values
- Whether a character is alphabetic, numeric, nonprinting, and so on
- The order in which characters should be sorted (collation)

You can mix locales if required, for example you could specify times and dates in one locale and monetary conventions in another.

Parallel jobs allow you to choose locales separately for:

- The order in which characters should be sorted (collation)

You can specify locales at different levels within InfoSphere DataStage:

- A project-wide default. In the Administrator client you specify default locales for all server jobs in a project, and a default locale for all parallel jobs in a project.
- A job default. In the Designer client, you can specify default locales used by a particular job that overrides the project default.
- A stage locale. Certain parallel stages allow you to specify that they use a particular locale. This overrides both the project default and the job default.

This manual uses the term *territory* rather than *country* to describe an area that uses a locale.

Time and Date

Most territories have a preferred style for presenting times and dates. For times, this is usually a choice between a 12-hour or 24-hour clock. For dates, there are more variations. Here are some examples of formats used by different locales to express 9.30 at night on the first day of April in 1990:

Territory	Time	Date	InfoSphere DataStage Locale
France	21h30	1.4.90	FR-FRENCH
U.S.	9:30 p.m.	4/1/90	US-ENGLISH
Japan	21:30	90.4.1	JP-JAPANESE

Numeric

This convention defines how numbers are displayed, including:

- The character used as the decimal separator (the radix character)
- The character used as a thousands separator
- Whether leading zeros should be used for numbers 1 through -1

For example, the following numbers can all mean one thousand, depending on the locale you use:

Territory	Number	InfoSphere DataStage Locale
Ireland	1,000	IE-ENGLISH
Netherlands	1.000	NL-DUTCH
France	1 000	FR-FRENCH

Monetary

This convention defines how monetary values are displayed, including:

- The character used as the decimal separator. This can differ from the decimal separator used in numeric formats.
- The character used as a thousands separator. This can differ from the thousands separator used in numeric formats.
- The local currency symbol for the territory, for example, $, £, or ¥.
- The string used as the international currency symbol, for example, USD (US Dollars), NOK (Norwegian Kroner), JPY (Japanese Yen).
- The number of decimal places used in local monetary values.
- The number of decimal places used in international monetary values.
- The sign used to indicate positive monetary values.
- The sign used to indicate negative monetary values.
- The relative positions of the currency symbol and any positive or negative signs in monetary values.

Here are examples of monetary formats different locales use:

Currency	Format	InfoSphere DataStage Locale
U.S. Dollars	$123.45	US-ENGLISH
UK Pounds	£37,000.00	GB-ENGLISH
German Marks	DM123,45	DE-GERMAN
German Euros	€123,45	DE-GERMAN-EURO

Character Type

This convention defines whether a character is alphabetic, numeric, nonprinting, and so on. This convention also defines any casing rules, for example, some letters take an accent in lowercase but not in uppercase.

Collation

This convention defines the order in which characters are collated, that is, sorted. There can be many variations in collation order within a single character set. For example, the character Ä follows A in Germany, but follows Z in Sweden.

Maps and Locales in InfoSphere DataStage Parallel Jobs

A large number of maps and locales are installed when you install InfoSphere DataStage with NLS enabled. You can view what maps and locales are currently loaded and which ones are available from the Administrator client.

Procedure

1. Open the Administrator client.
2. Click the **Projects** tab to go to the Projects page.
3. Select a project and click the **NLS...** button to open the **Project NLS Settings** dialog box for that project. By default this shows all the maps currently loaded for server jobs. Click the **Parallel Maps** tab to view the available parallel job maps. Map names beginning with ASCL are the parallel version of the maps available in server jobs.
4. To view loaded locales, click the **Parallel Locales** tab. Click on the down arrow next to each locale category to see drop down list of loaded locales. Select the **Show all locales** option to have the drop down lists show all the maps available for loading.

Using Maps in Parallel Jobs

You need to use a map whenever you are reading certain types of character data into InfoSphere DataStage or writing it out of InfoSphere DataStage. The map tells InfoSphere DataStage how to convert the external character set into the internal Unicode character set.

You do **not** need to map data if you are:

- Handling purely numeric data.
- Reading or writing 7-bit ASCII data.

InfoSphere DataStage allows you to specify the map to use at various points in a job design:

- You can specify the default map for a project. This is used by all stages in all jobs in a project unless specifically overridden in the job design.
- You can specify the default map for a job. This is used by all stages in a job (replacing the project default) unless overridden in the job design.
- You can specify a map for a particular stage in your job (depending on stage type). This overrides both the project default and the job default.
- For certain stages you can specify a map for individual columns, this overrides the project, job, and stage default maps.

Character Data in Parallel Jobs

You only need to specify a character set map where your job is processing character data. InfoSphere DataStage has a number of character types which can be specified as the SQL type of a column:

- Char

- VarChar
- LongVarChar
- NChar
- NVarChar
- LongNVarChar

InfoSphere DataStage parallel jobs store character data as string (byte per character) or ustring (unicode string).

The Char, VarChar, and LongVarChar relate to underlying string types where each character is 8-bits and does not require mapping because it represents an ASCII character. You can, however, specify that these data types are extended, in which case they are taken as ustrings and do require mapping. They are specified as such by selecting the **Extended** check box for the column in the Edit Metadata dialog box (opened for that column by selecting **Edit Row...** from the columns grid shortcut menu). An **Extended** field appears in the columns grid, and extended Char, VarChar, or LongVarChar columns have `Unicode' in this field. The NChar, NVarChar, and LongNVarChar types relate to underlying ustring types so do not need to be explicitly extended.

If you have selected Allow per-column mapping for this table (on the NLS page of the Table Definition dialog box or the NLS Map tab of a stage editor), you can select a character set map in the NLS Map field, otherwise the default map is used.

Specifying a Project Default Map

You specify the default map for a project in the InfoSphere DataStage Administrator Client.

Procedure

1. Open the Administrator client.
2. Click the **Projects** tab to go to the Projects page.
3. Select the project for which you want to set a default map and click the **NLS...** button to open the Project NLS Settings dialog box for that project. By default this shows all the maps currently loaded for server jobs. Click the **Parallel Maps** tab.
4. Choose the map you want from the **Default map name** list. Click **OK**. The selected map is now the default one for that project and is used by all the jobs in that project.

Specifying a Job Default Map

You specify a default map for a particular job in the InfoSphere DataStage Designer, using the Job Properties dialog box:

Procedure

1. Open the job for which you want to set the map in the Designer client.
2. Open the Job Properties dialog box for that job (choose **Edit** > **Job Properties**).
3. Click the **NLS** tab to go to the NLS page:
4. Choose the map you want from the **Default map for stages** list.
5. Click **OK**. The selected map is now the default one for that job and is used by all the stages in that job.

Specifying a Stage Map

You specify a map for a particular stage to use in the stage editor dialog in the InfoSphere DataStage Designer. You can specify maps for all types of stage that read or write data from/to an external data source.

About this task

Processing, Restructure, and Development/Debug stages deal with data that has already been input to InfoSphere DataStage and so has already been mapped.

Certain File stages, for example Data Set and Lookup File Set, represent data held by InfoSphere DataStage and so do not require mapping.

Procedure

1. Open the stage editor in the job in the Designer client. Select the **NLS Map** tab on the Stage page:
2. Do one of the following:
   - Choose the map you want from the **Map name for use with stage** list.
   - Click the arrow button next to the map name. This allows you to select an existing job parameter or specify a new one. When the job is run, InfoSphere DataStage will use the value of that parameter for the name of the map to use.
3. Click **OK**. The selected map or job parameter are used by the stage.

Specifying a Column Map

Certain types of parallel job stage allow you to specify a map that is used for a particular column in the data handled by that stage. All the stages that require mapping allow per-column mapping except for the Database stages.

Procedure

1. Open the stage editor in the job. Click on the **NLS Map** tab on the Stage page:
2. Select the **Allow per-column mapping** option. Then go to the **Inputs** or Outputs page (depending on whether you are writing or reading data) and select the **Columns** tab:
3. The columns grid now has an extra field called **NLS Map**. Choose the map you want for a particular column from the drop down list.
4. Click **OK**.

Using Locales in Parallel Jobs

Locales allows you to specify that data is sorted in accordance with the conventions of a certain territory. Note that there is not always a direct relationship between locale and language.

In parallel jobs you can set a default locale for a project, for an individual job, or for a particular stage. The default is for data to be sorted in accordance with the Unicode Collation Algorithm (UCA/14651). If you select a specific locale, you are effectively overriding certain features of the UCA collation base.

Note: Although you cannot specify date and time formats or decimal separators using the locale mechanism, there are ways to set these in parallel jobs. See "Defining Date/Time and Number Formats" for details.

Specifying a Project Default Locale

You specify the default locale for a project in the InfoSphere DataStage Administrator Client.

Procedure

1. Open the Administrator client.
2. Click the **Projects** tab to go to the Projects page.
3. Select the project for which you want to set a default map and click the **NLS...** button to open the Project NLS Settings dialog box for that project. Click the **Parallel Locales** tab to go to the Parallel Locales page.

4. Click on the arrow next to the **Collate** category and choose a locale from the drown down list. The setting OFF indicates that sorting will be carried out according to the base UCA rules.
5. Click **OK**. The selected locale is now the default one for that category in the project and is used by all the jobs in that project.

Specifying a Job Default Locale

You specify a default locale for a particular job in the InfoSphere DataStage Designer, using the Job Properties dialog.

Procedure

1. Open the job for which you want to set the locale in the Designer client.
2. Open the Job Properties dialog box for that job (choose **Edit** > **Job Properties**).
3. Click the **NLS** tab to go to the NLS page:
4. Choose a locale from the **Default collation locale for stages** list. The setting OFF indicates that sorting will be carried out according to the base UCA rules.
5. Click **OK**. The selected locale is now the default one for the job and is used by all the stages in that job.

Specifying a Stage Locale

You can specify a locale for stages that explicitly sort.

About this task

Stages that involve sorting of data allow you to specify a locale, overriding the project and job default.

You can also specify a sort on the Partitioning tab of most stages, depending on partition method chosen. This sort is performed before the incoming data is processed by the stage. You can specify a locale for this sort that overrides the project and job default.

Procedure

1. Open the stage editor and go to the NLS Locale tab of the Stage page.
2. Choose the required locale from the list and click **OK**. The stage will sort according to the conventions specified by that locale. The setting OFF indicates that sorting will be carried out according to the base UCA rules.

Defining Date/Time and Number Formats

Although you cannot set new formats for dates and times or numbers using the locales mechanism, there are other ways of doing this in parallel jobs. You can do this at project level, at job level, for certain types of individual stage, and at column level.

Specifying Formats at Project Level

You can specify date/time and number formats for a project in the InfoSphere DataStage Administrator Client.

Procedure

1. Open the Administrator client.
2. Click the **Projects** tab to go to the Projects page.
3. Select the project for which you want to set a default map and click the **Properties** button to open the Project Properties dialog box for that project. Click the **Parallel** tab to go to the Parallel page.

4. The page shows the current defaults for date, time, timestamp, and decimal separator. To change the default, clear the corresponding **System default** check box, then either select a new format from the drop down list or type in a new format.
5. Click **OK** to set the new formats as defaults for the project.

Specifying Formats at Job Level

You specify date/time and number formats for a particular job in the InfoSphere DataStage Designer, using the Job Properties dialog.

Procedure

1. Open the job for which you want to set the formats in the Designer client.
2. Open the Job Properties dialog box for that job (choose **Edit** > **Job Properties**).
3. Click the **Defaults** tab to go to the Defaults page:
4. The page shows the current defaults for date, time, timestamp, and decimal separator. To change the default, clear the corresponding **Project default** check box, then either select a new format from the drop down list or type in a new format.
5. Click **OK** to set the new formats as defaults for the job.

Specifying Formats at Stage Level

Stages that have a **Format** tab on their editor allow you to override the project and job defaults for date and time and number formats.

About this task

These stages are:

- Sequential File stage
- File Set stage
- External Source stage
- External Target stage
- Column Import stage
- Column Export stage

Procedure

1. Open the stage editor for the stage you want to change and go to the **Formats** tab on either the Input or Output page (as appropriate).
2. To change the decimal separator, select the Decimal category under the Type defaults category in the Properties tree, then click **Decimal separator** in the **Available properties to add** list. You can then choose a new value in the **Decimal separator** box that appears in the upper right of the dialog box.
3. To change the date format, select the Date category under the Type defaults category in the Properties tree, then click **Format string** in the **Available properties to add** list. You can then specify a new format in the **Format string** box that appears in the upper right of the dialog box:
4. To change the time format, select the Time category under the Type defaults category in the Properties tree, then click **Format string** in the **Available properties to add** list. You can then specify a new format in the **Format string** box that appears in the upper right of the dialog box:
5. To change the timestamp format, select the Timestamp category under the Type defaults category in the Properties tree, then click **Format string** in the **Available properties to add** list. You can then specify a new format in the **Format string** box that appears in the upper right of the dialog box:

Specifying Formats at Column Level

You can specify date/time and number formats at column level either from Columns tabs of stage editors, or from the Columns page of a Table Definition dialog box.

Procedure

1. In the columns grid, select the column for which you want to specify a format, right click and select **Edit Row...** from the shortcut menu. The Edit Column Metadata dialog box appears:
2. The information shown in the Parallel tab varies according to the type of the column you are editing. In the example it is a date column. To change the format of the date, select the Date type category in the Properties tree, then click **Format string** in the **Available properties to add** list. You can then specify a new format in the **Format string** box that appears in the top right of the dialog box:
3. Click **Apply** to implement the change, then click **Close**.

Results

The method for changing time, timestamp, and decimal separator are similar. When you select a column of the time, timestamp, numeric, or decimal type the available properties allow you to specify a new format for that column.

Chapter 4. Stage editors

The Parallel job stage editors all use a generic user interface (with the exception of the Transformer stage, Shared Container, and Complex Flat File stages). This chapter describes the generic editor and gives a guide to using it.

Parallel jobs have a large number of stages available. They are organized into groups in the tool palette or you can drag all the stages you use frequently to the **Favorites** category.

The stage editors are divided into the following basic types:

- **Database**. These are stages that read or write data contained in a database. Examples of database stages are the Oracle Enterprise and DB2/UDB Enterprise stages.
- **Development/Debug**. These are stages that help you when you are developing and troubleshooting parallel jobs. Examples are the Peek and Row Generator stages.
- **File**. These are stages that read or write data contained in a file or set of files. Examples of file stages are the Sequential File and Data Set stages.
- **Processing**. These are stages that perform some processing on the data that is passing through them. Examples of processing stages are the Aggregator and Transformer stages.
- **Real Time**. These are the stages that allow Parallel jobs to be made available as web services. They are part of the optional Web Services package.
- **Restructure**. These are stages that deal with and manipulate data containing columns of complex data type. Examples are Make Subrecord and Make Vector stages.

Parallel jobs also support local containers and shared containers. Local containers allow you to tidy your designs by putting portions of functionality in a container, the contents of which are viewed on a separate canvas. Shared containers are similar, but are stored separately in the repository and can be reused by other parallel jobs. Parallel jobs can use both Parallel Shared Containers and Server Shared Containers. Using shared containers is described in *InfoSphere DataStage Designer Client Guide*.

The following table lists the available stage types and gives a quick guide to their function:

Table 10. Stage editors

Stage	Type	Function
Data Set	File	Allows you to read data from or write data to a persistent data set.
Sequential File	File	Allows you to read data from or write data to one or more flat files.
File Set	File	Allows you to read data from or write data to a file set. File sets enable you to spread data across a set of files referenced by a single control file.
Lookup File Set	File	Allows you to create a lookup file set or reference one for a lookup.
External Source	File	Allows you to read data that is output from one or more source programs.
External Target	File	Allows you to write data to one or more source programs.

Table 10. Stage editors (continued)

Stage	Type	Function
Complex Flat File	File	Allows you to read or write complex flat files on a mainframe machine. This is intended for use on USS systems (note that it uses a different interface from other file stages).
SAS Data Set	File	Allows you to read data from or write data to a parallel SAS data set in conjunction with an SAS stage.
DB2/UDB Enterprise	Database	Allows you to read data from and write data to a DB2 database.
Oracle Enterprise	Database	Allows you to read data from and write data to a Oracle database.
Teradata Enterprise	Database	Allows you to read data from and write data to a Teradata database.
Informix® Enterprise	Database	Allows you to read data from and write data to an Informix database.
Transformer	Processing	Handles extracted data, performs any conversions required, and passes data to another active stage or a stage that writes data to a target database or file.
BASIC Transformer	Processing	Same as Transformer stage, but gives access to InfoSphere DataStage BASIC functions.
Aggregator	Processing	Classifies incoming data into groups, computes totals and other summary functions for each group, and passes them to another stage in the job.
Join	Processing	Performs join operations on two or more data sets input to the stage and then outputs the resulting data set.
Merge	Processing	Combines a sorted master data set with one or more sorted update data sets.
Lookup	Processing	Used to perform lookup operations on a data set read into memory from any other Parallel job stage that can output data or provided by one of the database stages that support reference output links. It can also perform a look up on a lookup table contained in a Lookup File Set stage.
Sort	Processing	Sorts input columns.
Funnel	Processing	Copies multiple input data sets to a single output data set.
Remove Duplicates	Processing	Takes a single sorted data set as input, removes all duplicate records, and writes the results to an output data set.

Table 10. Stage editors (continued)

Stage	Type	Function
Compress	Processing	Uses the UNIX *compress* or *GZIP* utility to compress a data set. It converts a data set from a sequence of records into a stream of raw binary data.
Expand	Processing	Uses the UNIX *uncompress* or *GZIP* utility to expand a data set. It converts a previously compressed data set back into a sequence of records from a stream of raw binary data.
Copy	Processing	Copies a single input data set to a number of output data sets.
Modify	Processing	Alters the record schema of its input data set.
Filter	Processing	Transfers, unmodified, the records of the input data set which satisfy requirements that you specify and filters out all other records.
External Filter	Processing	Allows you to specify a UNIX command that acts as a filter on the data you are processing.
Change Capture	Processing	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.
Change Apply	Processing	Takes the *change* data set, that contains the changes in the before and after data sets, from the Change Capture stage and applies the encoded change operations to a *before* data set to compute an *after* data set.
Difference	Processing	Performs a record-by-record comparison of two input data sets, which are different versions of the same data set.
Compare	Processing	Performs a column-by-column comparison of records in two presorted input data sets.
Encode	Processing	Encodes a data set using a UNIX encoding command that you supply.
Decode	Processing	Decodes a data set using a UNIX decoding command that you supply.
Switch	Processing	Takes a single data set as input and assigns each input record to an output data set based on the value of a selector field.
FTP	Processing	Allows you to FTP data to another machine.

Table 10. Stage editors (continued)

Stage	Type	Function
SAS (SAS Connectivity Guide)	Processing	Allows you to execute part or all of an SAS application in parallel.
Generic	Processing	Lets you incorporate an Orchestrate® Operator in your job.
Surrogate Key	Processing	Generates one or more surrogate key columns and adds them to an existing data set.
Column Import	Restructure	Imports data from a single column and outputs it to one or more columns.
Column Export	Restructure	Exports data from a number of columns of different data types into a single column of data type string or binary.
Make Subrecord	Restructure	Combines specified vectors in an input data set into a vector of subrecords whose columns have the names and data types of the original vectors.
Split Subrecord	Restructure	Creates one new vector column for each element of the original subrecord.
Combine Records	Restructure	Combines records, in which particular key-column values are identical, into vectors of subrecords.
Promote Subrecord	Restructure	Promotes the columns of an input subrecord to top-level columns.
Make Vector	Restructure	Combines specified columns of an input data record into a vector of columns of the same type.
Split Vector	Restructure	Promotes the elements of a fixed-length vector to a set of similarly named top-level columns.
Head	Development/ Debug	Selects the first *N* records from each partition of an input data set and copies the selected records to an output data set.
Tail	Development/ Debug	Selects the last *N* records from each partition of an input data set and copies the selected records to an output data set.
Sample	Development/ Debug	Samples an input data set.
Peek	Development/ Debug	Lets you print record column values either to the job log or to a separate output link as the stage copies records from its input data set to one or more output data sets.
Row Generator	Development/ Debug	Produces a set of mock data fitting the specified meta data.

Table 10. Stage editors (continued)

Stage	Type	Function
Column Generator	Development/ Debug	Adds columns to incoming data and generates mock data for these columns for each data row processed.
Write Range Map	Development/ Debug	Allows you to write data to a range map. The stage can have a single input link.

All of the stage types use the same basic stage editor, but the pages that actually appear when you edit the stage depend on the exact type of stage you are editing. The following sections describe all the page types and sub tabs that are available. The individual descriptions of stage editors in the following chapters tell you exactly which features of the generic editor each stage type uses.

The stage page

All stage editors have a Stage page. This contains a number of subsidiary tabs depending on the stage type. The only field the Stage page itself contains gives the name of the stage being edited.

General tab

All stage editors have a General tab, this allows you to enter an optional description of the stage. Specifying a description here enhances job maintainability.

Properties tab

A Properties tab appears on the Stage page where there are general properties that need setting for the particular stage you are editing. Properties tabs can also occur under Input and Output pages where there are link-specific properties that need to be set.

The properties for most general stages are set under the Stage page. The following figure shows an example Properties tab.

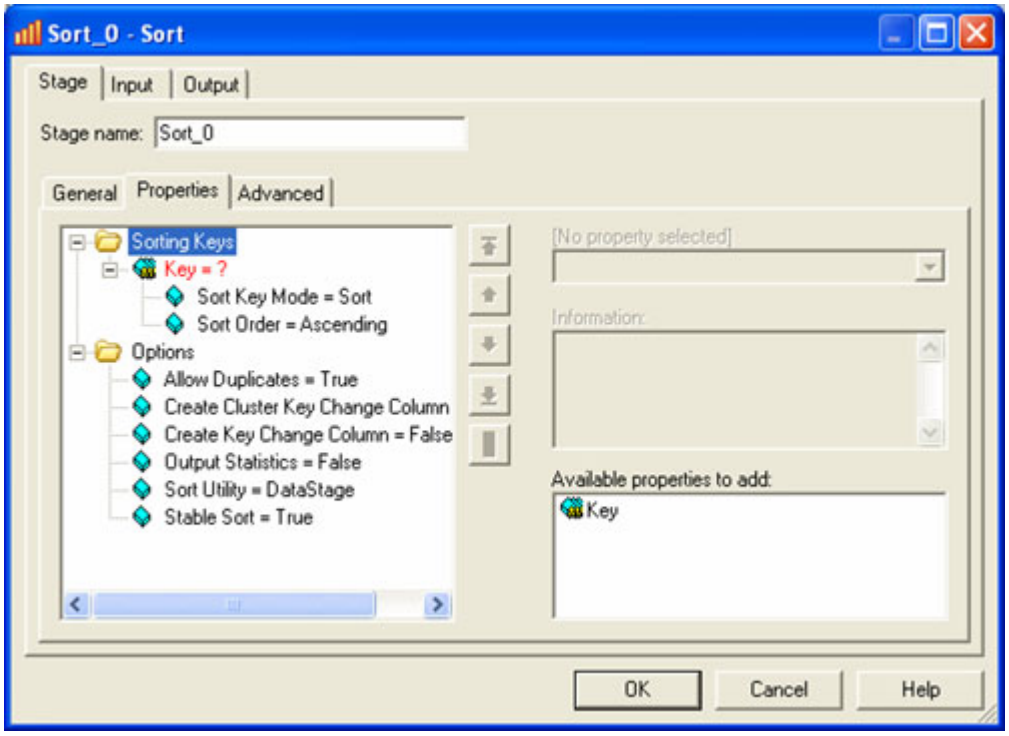


Figure 1. Properties tab

The available properties are displayed in a tree structure. They are divided into categories to help you find your way around them. All the mandatory properties are included in the tree by default and cannot be removed. Properties that you must set a value for (that is, which have not got a default value) are shown in the warning color (red by default), but change to black when you have set a value. You can change the warning color by opening the Options dialog box (select **Tools** > **Options ...** from the Designer main menu) and choosing the Transformer item from the tree. Reset the Invalid column color by clicking on the color bar and choosing a new color from the palette.

To set a property, select it in the list and specify the required property value in the property value field. The title of this field and the method for entering a value changes according to the property you have selected. In the example above, the Key property is selected so the Property Value field is called Key and you set its value by choosing one of the available input columns from a drop down list. Key is shown in red because you must select a key for the stage to work properly. The Information field contains details about the property you currently have selected in the tree. Where you can browse for a property value, or insert a job parameter whose value is provided at run time, a right arrow appears next to the field. Click on this and a menu gives access to the Browse Files dialog box or a list of available job parameters (job parameters are defined in the Job Properties dialog box - see *InfoSphere DataStage Designer Client Guide*).

Some properties have default values, and you can always return to the default by selecting it in the tree and choosing **Set to default** from the shortcut menu.

Some properties are optional. These appear in the **Available properties to add** field. Click on an optional property to add it to the tree or choose to add it from the shortcut menu. You can remove it again by selecting it in the tree and selecting **Remove** from the shortcut menu.

Some properties can be repeated. In the example above you can add multiple key properties. The **Key** property appears in the **Available properties to add** list when you select the tree top level Properties node. Click on the Key item to add multiple key properties to the tree. Where a repeatable property

expects a column as an argument, a dialog is available that lets you specify multiple columns at once. To open this, click the column button next to the properties tree.

The Column Selection dialog box opens. The left pane lists all the available columns, use the arrow right keys to select some or all of them (use the left arrow keys to move them back if you change your mind). A separate property will appear for each column you have selected.

Some properties have dependents. These are properties which somehow relate to or modify the parent property. They appear under the parent in a tree structure.

For some properties you can supply a job parameter as their value. At runtime the value of this parameter will be used for the property. Such properties will have an arrow next to their Property Value box. Click the arrow to get a drop-down menu, then choose **Insert job parameter** get a list of currently defined job parameters to chose from (see *InfoSphere DataStage Designer Client Guide* for information about job parameters).

You can switch to a multiline editor for entering property values for some properties. Do this by clicking on the arrow next to their Property Value box and choosing **Switch to multiline editor** from the menu.

The property capabilities are indicated by different icons in the tree as follows:

non-repeating property with no dependents

non-repeating property with dependents

repeating property with no dependents

repeating property with dependents

The properties for individual stage types are described in the chapter about the stage.

Advanced tab

All stage editors have an Advanced tab. This allows you to:

- Specify the execution mode of the stage. This allows you to choose between Parallel and Sequential operation. If the execution mode for a particular type of stage cannot be changed, then this drop down list is disabled. Selecting Sequential operation forces the stage to be executed on a single node. If you have intermixed sequential and parallel stages this has implications for partitioning and collecting data between the stages. You can also let InfoSphere DataStage decide by choosing the **default** setting for the stage (the drop down list tells you whether this is parallel or sequential).
- Set or clear the preserve partitioning flag (this field is not available for all stage types). It indicates whether the stage wants to preserve partitioning at the next stage of the job. You choose between Set, Clear and Propagate. For some stage types, Propagate is not available. The operation of each option is as follows:
  - **Set**. Sets the preserve partitioning flag, this indicates to the next stage in the job that it should preserve existing partitioning if possible.

- **Clear**. Clears the preserve partitioning flag. Indicates that this stage does not care which partitioning method the next stage uses.
- **Propagate**. Sets the flag to Set or Clear depending on what the previous stage in the job has set (or if that is set to Propagate the stage before that and so on until a preserve partitioning flag setting is encountered).

  You can also let InfoSphere DataStage decide by choosing the **default** setting for the stage (the drop down list tells you whether this is set, clear, or propagate).

- Specify the combinability mode. Under the covers InfoSphere DataStage can combine the operators that underlie parallel stages so that they run in the same process. This saves a significant amount of data copying and preparation in passing data between operators.

  The combinability mode setting tells InfoSphere DataStage your preferences for combining for a particular stage. It has three possible settings:
  - **Auto**. Use the default combination setting.
  - **Combinable**. Ignore the operator's default setting and combine if at all possible (some operators are marked as noncombinable by default).
  - **Don't Combine**. Never combine operators.

    In most cases the setting should be left to Auto.
- Specify node map or node pool or resource pool constraints. The configuration file allows you to set up pools of related nodes or resources. The Advanced tab allows you to limit execution of a stage to a particular node or resource pool. You can also use a map to specify a group of nodes that execution will be limited to just in this stage. Supply details as follows:
  - **Node pool and resource constraints**. Specify constraints in the grid. Select Node pool or Resource pool from the **Constraint** drop-down list. Select a Type for a resource pool and, finally, select the name of the pool you are limiting execution to. You can select multiple node or resource pools. This is only enabled if you have defined multiple pools in the configuration file.
  - **Node map constraints**. Select the option box and type in the nodes to which execution will be limited in the text box. You can also browse through the available nodes to add to the text box. Using this feature conceptually sets up an additional node pool which does not appear in the configuration file.

    The lists of available nodes, available node pools, and available resource pools are derived from the configuration file.

Link ordering tab

This tab allows you to order the links for stages that have more than one link and where ordering of the links is required.

The tab allows you to order input links or output links as needed. Where link ordering is not important or is not possible the tab does not appear.

The link label gives further information about the links being ordered. In the example you are looking at the Link Ordering tab for a Join stage. The join operates in terms of having a left link and a right link, and this tab tells you which actual link the stage regards as being left and which right. If you use the arrow keys to change the link order, the link name changes but not the link label. In the example, if you pressed the down arrow button, DSLink27 would become the left link, and DSLink26 the right.

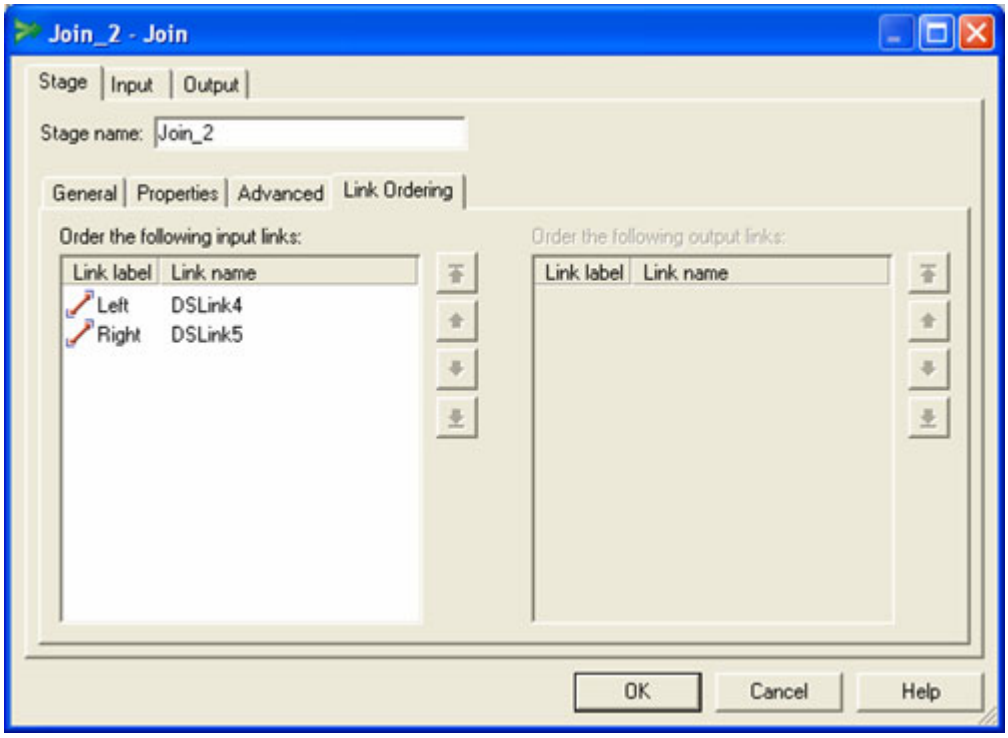


Figure 2. Link Ordering tab: Join stage

A Join stage can only have one output link, so in the example the **Order the following output links** section is disabled.

The following example shows the Link Ordering tab from a Merge stage. In this case you can order both input links and output links. The Merge stage handles reject links as well as a stream link and the tab allows you to order these, although you cannot move them to the stream link position. Again the link labels give the sense of how the links are being used.

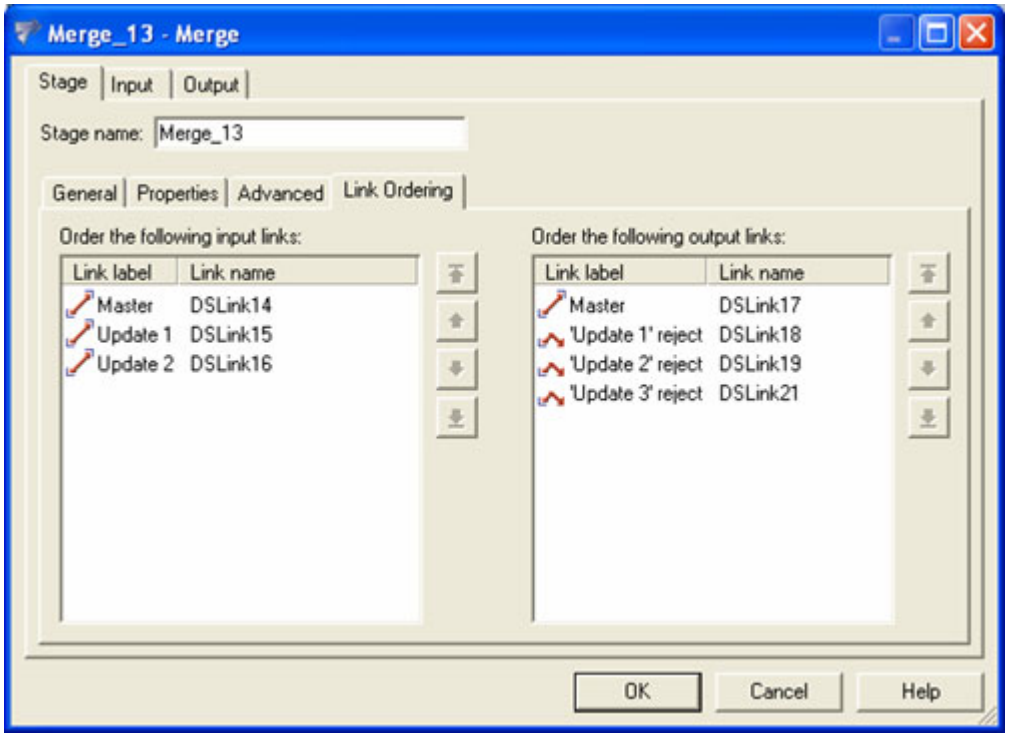


Figure 3. Link Ordering tab: Merge stage

The individual stage descriptions tell you whether link ordering is possible and what options are available.

NLS Map tab

If you have NLS enabled on your system, some of your stages will have an NLS Map tab. This allows you to override the project default character set map for this stage, and in some cases, allows you to enable per-column mapping. When per-column mapping is enabled, you can override the character set map for particular columns (an NLS map field appears on the Columns tab allowing you to do this).

Select a map from the list, or click the arrow button next to the list to specify a job parameter.

The following stage types currently support this feature:

- Sequential File
- File Set
- Lookup File Set
- External Source
- External Target
- DB2/UDB Enterprise (not per-column mapping)
- Oracle Enterprise (not per-column mapping)

NLS Locale tab

If you have NLS enabled on your system, some of your stages will have an NLS Locale tab. It lets you view the current default collate convention, and select a different one for the stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated, for example, the character Ä follows A in Germany, but follows Z in Sweden.

Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

The following types of stage have an NLS Locale tab:

- Stages that evaluate expressions, such as the Transformer.
- Stages that need to evaluate the order of key columns.
- The Sort Stage.

Inputs page

The Input page gives information about links going into a stage. In the case of a file or database stage an input link carries data being written to the file or database. In the case of a processing or restructure stage it carries data that the stage will process before outputting to another stage. Where there are no input links, the stage editor has no Input page.

Where it is present, the Input page contains various tabs depending on stage type. The only field the Input page itself contains is **Input name**, which gives the name of the link being edited. Where a stage has more than one input link, you can select the link you are editing from the **Input name** drop-down list.

The Input page also has a **Columns...** button. Click this to open a window showing column names from the meta data defined for this link. You can drag these columns to various fields in the Input page tabs as required.

Certain stage types will also have a **View Data...** button. Press this to view the actual data associated with the specified data source or data target. The button is available if you have defined meta data for the link. Note the interface allowing you to view the file will be slightly different depending on stage and link type.

General tab

The Input page always has a General tab. this allows you to enter an optional description of the link. Specifying a description for each link enhances job maintainability.

Properties tab

Some types of file and database stages can have properties that are particular to specific input links. In this case the Input page has a Properties tab. This has the same format as the Stage page Properties tab (see "Properties Tab").

Partitioning tab

Most parallel stages have a default partitioning or collecting method associated with them. This is used depending on the execution mode of the stage (that is, parallel or sequential) and the execution mode of the immediately preceding stage in the job. For example, if the preceding stage is processing data sequentially and the current stage is processing in parallel, the data will be partitioned before it enters the current stage. Conversely if the preceding stage is processing data in parallel and the current stage is sequential, the data will be collected as it enters the current stage.

You can, if required, override the default partitioning or collecting method on the Partitioning tab. The selected method is applied to the incoming data as it enters the stage on a particular link, and so the Partitioning tab appears on the Input page. You can also use the tab to repartition data between two parallel stages. If both stages are executing sequentially, you cannot select a partition or collection method

and the fields are disabled. The fields are also disabled if the particular stage does not permit selection of partitioning or collection methods. The following table shows what can be set from the Partitioning tab in what circumstances:

Preceding Stage	Current Stage	Partition Tab Mode
Parallel	Parallel	Partition
Parallel	Sequential	Collect
Sequential	Parallel	Partition
Sequential	Sequential	None (disabled)

The Partitioning tab also allows you to specify that the data should be sorted as it enters.

The Partitioning tab has the following fields:

- **Partition type**. Choose the partitioning (or collecting) type from the drop-down list. The following partitioning types are available:
  - **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for many stages.
  - **Entire**. Every processing node receives the entire data set. No further information is required.
  - **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the Available list.
  - **Modulus**. The records are partitioned using a modulus function on the key column selected from the Available list. This is commonly used to partition on tag fields.
  - **Random**. The records are partitioned randomly, based on the output of a random number generator. No further information is required.
  - **Round Robin**. The records are partitioned on a round robin basis as they enter the stage. No further information is required.
  - **Same**. Preserves the partitioning already in place. No further information is required.
  - **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
  - **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

    The following collection types are available:
  - **(Auto)**. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available. This is the fastest collecting method and is the default collection method for many stages. In some circumstances InfoSphere DataStage will detect further requirements for collected data, for example, it might need to be sorted. Using Auto mode will ensure data is sorted if required.
  - **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on. Requires no further information.
  - **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
  - **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.
- **Available**. This lists the input columns for the input link. Key columns are identified by a key icon. For partitioning or collecting methods that require you to select columns, you click on the required column in the list and it appears in the **Selected** list to the right. This list is also used to select columns to sort on.

- **Selected**. This list shows which columns have been selected for partitioning on, collecting on, or sorting on and displays information about them. The available information is whether a sort is being performed (indicated by an arrow), if so the order of the sort (ascending or descending) and collating sequence (sort as EBCDIC), and whether an alphanumeric key is case sensitive or not. Nullable columns are marked to indicate if null columns take first or last position. You can select sort order, case sensitivity, collating sequence, and nulls position from the shortcut menu. If applicable, the Usage field indicates whether a particular key column is being used for sorting, partitioning, or both.
- **Sorting**. The check boxes in the section allow you to specify sort details. The availability of sorting depends on the partitioning method chosen.
  - **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
  - **Stable**. Select this if you want to preserve previously sorted data sets. The default is stable.
  - **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

    You can also specify sort direction, case sensitivity, whether sorted as EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu. The availability of the sort options depends on the type of data in the column, whether it is nullable or not, and the partitioning method chosen.

    If you have NLS enabled, the sorting box has an additional button. Click this to open the NLS Locales tab of the Sort Properties dialog box. This lets you view the current default collate convention, and select a different one for the stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated, for example, the character Ä follows A in Germany, but follows Z in Sweden. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

    If you require a more complex sort operation, you should use the Sort stage (see “Sort stage” on page 271).

DB2 partition properties

This dialog box appears when you select a **Partition type** of DB2 and click the properties button. It allows you to specify the DB2 table whose partitioning method is to be replicated.

Range partition properties

This dialog box appears when you select a **Partition type** of **Range** and click the properties button. It allows you to specify the range map that is to be used to determine the partitioning (you create a range map file using the Write Range Map stage - see “Write Range Map stage” on page 540). Type in a pathname or browse for a file.

Format tab

Stages that write to certain types of file (for example, the Sequential File stage) also have a Format tab which allows you to specify the format of the file or files being written to.

The Format tab is similar in structure to the Properties tab. A flat file has a number of properties that you can set different attributes for. Select the property in the tree and select the attributes you want to set from the **Available properties to add** box, it will then appear as a dependent property in the property tree and you can set its value as required. This tab sets the format information for the file at row level. You can override the settings for individual columns using the Edit Column Metadata dialog box (see page Field Level).

If you click the **Load** button you can load the format information from a table definition in the Repository.

The shortcut menu from the property tree gives access to the following functions:

- **Format as**. This applies a predefined template of properties. Choose from the following:
  - Delimited/quoted
  - Fixed-width records
  - UNIX line terminator
  - DOS line terminator
  - No terminator (fixed width)
  - Mainframe (COBOL)
- **Add sub-property**. Gives access to a list of dependent properties for the currently selected property (visible only if the property has dependents).
- **Set to default**. Appears if the currently selected property has been set to a non-default value, allowing you to re-select the default.
- **Remove**. Removes the currently selected property. This is disabled if the current property is mandatory.
- **Remove all**. Removes all the non-mandatory properties.

Columns Tab

The Input page always has a Columns tab. This displays the column meta data for the selected input link in a grid.

There are various ways of populating the grid:

- If the other end of the link has meta data specified for it, this will be displayed in the Columns tab (meta data is associated with, and travels with, a link).
- You can type the required meta data into the grid. When you have done this, you can click the **Save...** button to save the meta data as a table definition in the Repository for subsequent reuse.
- You can load an existing table definition from the Repository. Click the **Load...** button to be offered a choice of table definitions to load. Note that when you load in this way you bring in the columns definitions, not any formatting information associated with them (to load that, go to the Format tab).
- You can drag a table definition from the Repository Window on the Designer onto a link on the canvas. This transfers both the column definitions and the associated format information.

If you select the options in the Grid Properties dialog box (see *InfoSphere DataStage Designer Client Guide*), the Columns tab will also display two extra fields: **Table Definition Reference** and **Column Definition Reference**. These show the table definition and individual columns that the columns on the tab were derived from.

If you click in a row and select **Edit Row...** from the shortcut menu, the Edit Column Meta Data dialog box appears, which allows you edit the row details in a dialog box format. It also has a Parallel tab which allows you to specify properties that are peculiar to parallel job column definitions. The dialog box only shows those properties that are relevant for the current link.

The Parallel tab enables you to specify properties that give more detail about each column, and properties that are specific to the data type. Where you are specifying complex data types, you can specify a level number, which causes the Level Number field to appear in the grid on the Columns page.

If you have NLS enabled, and the column has an underlying string type, you can specify that the column contains Unicode data by selecting the Extended (Unicode) check box. Where you can enter a character for any property, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled).

Some table definitions need format information. This occurs where data is being written to a file where InfoSphere DataStage needs additional information in order to be able to locate columns and rows. Properties for the table definition at row level are set on the Format tab of the relevant stage editor, but you can override the settings for individual columns using the Parallel tab. The settings are made in a properties tree under the following categories:

Field level

This has the following properties:

- **Bytes to Skip**. Skip the specified number of bytes from the end of the previous column to the beginning of this column.
- **Delimiter**. Specifies the trailing delimiter of the column. Type an ASCII character or select one of whitespace, end, none, null, comma, or tab.
  - **whitespace**. The last column of each record will not include any trailing white spaces found at the end of the record.
  - **end**. The end of a field is taken as the delimiter, that is, there is no separate delimiter. This is not the same as a setting of `None' which is used for fields with fixed-width columns.
  - **none**. No delimiter (used for fixed-width).
  - **null**. ASCII Null character is used.
  - **comma**. ASCII comma character used.
  - **tab**. ASCII tab character used.
- **Delimiter string**. Specify a string to be written at the end of the column. Enter one or more characters. This is mutually exclusive with Delimiter, which is the default. For example, specifying `, ` (comma space - you do not need to enter the inverted commas) would have the column delimited by `, `.
- **Drop on input**. Select this property when you must fully define the meta data for a data set, but do not want the column actually read into the data set.
- **Prefix bytes**. Specifies that this column is prefixed by 1, 2, or 4 bytes containing, as a binary value, either the column's length or the tag value for a tagged column. 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. InfoSphere DataStage inserts the prefix before each field.

  This property is mutually exclusive with the Delimiter, Quote, and Final Delimiter properties, which are used by default.
- **Print field**. This property is intended for use when debugging jobs. Set it to have InfoSphere DataStage produce a message for each of the columns it reads. The message has the format:

  `Importing N: D`

  where:
  - *N* is the column name.
  - *D* is the imported data of the column. Non-printable characters contained in *D* are prefixed with an escape character and written as C string literals; if the column contains binary data, it is output in octal format.
- **Quote**. Specifies that variable length columns are enclosed in single quotes, double quotes, or another ASCII character or pair of ASCII characters. Choose **Single** or **Double**, or enter a character.
- **Start position**. Specifies the starting position of a column in the record. The starting position can be either an absolute byte offset from the first record position (0) or the starting position of another column.
- **Tag case value**. Explicitly specifies the tag value corresponding to a subfield in a tagged subrecord. By default the fields are numbered 0 to *N*-1, where *N* is the number of fields. (A tagged subrecord is a column whose type can vary. The subfields of the tagged subrecord are the possible types. The tag case value of the tagged subrecord selects which of those types is used to interpret the column's value for the record.)

String type

This has the following properties:

- **Character Set**. Choose from ASCII or EBCDIC (not available for ustring type (Unicode)).
- **Default**. The default value for a column. This is used for data written by a Generate stage. It also supplies the value to substitute for a column that causes an error (whether written or read).
- **Export EBCDIC as ASCII**. Select this to specify that EBCDIC characters are written as ASCII characters (not available for ustring type (Unicode)).
- **Is link field**. Selected to indicate that a column holds the length of another, variable-length column of the record or of the tag value of a tagged record field.
- **Import ASCII as EBCDIC**. Select this to specify that ASCII characters are read as EBCDIC characters (not available for ustring type (Unicode)).
- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.
- **Field width**. The number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.
- **Pad char**. Specifies the pad character used when strings or numeric values are written to an external string representation. Enter a character (single-byte for strings, can be multi-byte for ustrings) or choose null or space. The pad character is used when the external string representation is larger than required to hold the written field. In this case, the external string is filled with the pad character to its full length. Space is the default. Applies to string, ustring, and numeric data types and record, subrec, or tagged types if they contain at least one field of this type.

Date type

- **Byte order.** Specifies how multiple byte data types are ordered. Choose from:
  - little-endian. The high byte is on the right.
  - big-endian. The high byte is on the left.
  - native-endian. As defined by the native format of the machine.
- **Character Set**. Choose from ASCII or EBCDIC.
- **Days since**. Dates are written as a signed integer containing the number of days since the specified date. Enter a date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system.
- **Data Format**. Specifies the data representation format of a column. Choose from:
  - binary
  - text

    For dates, binary is equivalent to specifying the julian property for the date field, text specifies that the data to be written contains a text-based date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system.
- **Default**. The default value for a column. This is used for data written by a Generate stage. It also supplies the value to substitute for a column that causes an error (whether written or read).
- **Format string**. The string format of a date. By default this is *%yyyy-%mm-%dd*. For details about the format, see "Date formats" on page 31.

  If this format string does not include a day, it 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.

- **Is Julian**. Select this to specify that dates are written as a numeric value containing the Julian day. A Julian day specifies the date as the number of days from 4713 BCE January 1, 12:00 hours (noon) GMT.

Time type

- **Byte order**. Specifies how multiple byte data types are ordered. Choose from:
  - little-endian. The high byte is on the right.
  - big-endian. The high byte is on the left.
  - native-endian. As defined by the native format of the machine.
- **Character Set**. Choose from ASCII or EBCDIC.
- **Default**. The default value for a column. This is used for data written by a Generate stage. It also supplies the value to substitute for a column that causes an error (whether written or read).
- **Data Format**. Specifies the data representation format of a column. Choose from:
  - binary
  - text

    For time, binary is equivalent to midnight_seconds, text specifies that the field represents time in the text-based form *%hh:%nn:%ss* or in the default date format if you have defined a new one on an NLS system.
- **Format string**. Specifies the format of columns representing time as a string. By default this is %hh-%mm-%ss. For details about the format, see "Time formats" on page 35
- **Is midnight seconds**. Select this to specify that times are written as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.

Timestamp type

- **Byte order**. Specifies how multiple byte data types are ordered. Choose from:
  - little-endian. The high byte is on the right.
  - big-endian. The high byte is on the left.
  - native-endian. As defined by the native format of the machine.
- **Character Set**. Choose from ASCII or EBCDIC.
- **Data Format**. Specifies the data representation format of a column. Choose from:
  - binary
  - text

    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. A binary timestamp specifies that two 32-but integers are written. Text specifies a text-based timestamp in the form *%yyyy-%mm-%dd %hh:%nn:%ss* or in the default date format if you have defined a new one on an NLS system.
- **Default**. The default value for a column. This is used for data written by a Generate stage. It also supplies the value to substitute for a column that causes an error (whether written or read).
- **Format string**. Specifies the format of a column representing a timestamp as a string. Defaults to *%yyyy-%mm-%dd %hh:%nn:%ss*. The format combines the format for date strings and time strings. See "Date formats" on page 31 and "Time formats" on page 35.

Integer type

- **Byte order**. Specifies how multiple byte data types are ordered. Choose from:
  - little-endian. The high byte is on the right.
  - big-endian. The high byte is on the left.
  - native-endian. As defined by the native format of the machine.
- **Character Set**. Choose from ASCII or EBCDIC.

- **C_format**. Perform non-default conversion of data from a string to integer data. This property specifies a C-language format string used for reading/writing integer strings. This is passed to *sscanf()* or *sprintf().*
- **Default**. The default value for a column. This is used for data written by a Generate stage. It also supplies the value to substitute for a column that causes an error (whether written or read).
- **Data Format**. Specifies the data representation format of a column. Choose from:
  - binary
  - text
- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.
- **Field width**. The number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.
- **In_format**. Format string used for conversion of data from string to integer. This is passed to *sscanf*(). By default, InfoSphere DataStage invokes the C *sscanf*() function to convert a numeric field formatted as a string to either integer or floating point data. If this function does not output data in a satisfactory format, you can specify the in_format property to pass formatting arguments to *sscanf*().
- **Is link field**. Selected to indicate that a column holds the length of another, variable-length column of the record or of the tag value of a tagged record field.
- **Out_format**. Format string used for conversion of data from integer to a string. This is passed to *sprintf*(). By default, InfoSphere DataStage invokes the C *sprintf*() function to convert a numeric field formatted as integer data to a string. If this function does not output data in a satisfactory format, you can specify the out_format property to pass formatting arguments to *sprintf*().
- **Pad char**. Specifies the pad character used when the integer is written to an external string representation. Enter a character (single-bye for strings, can be multi-byte for ustrings) or choose null or space. The pad character is used when the external string representation is larger than required to hold the written field. In this case, the external string is filled with the pad character to its full length. Space is the default.

Decimal type

- **Allow all zeros**. Specifies whether to treat a packed decimal column containing all zeros (which is normally illegal) as a valid representation of zero. Select **Yes** or **No**.
- **Character Set**. Choose from ASCII or EBCDIC.
- **Decimal separator.** Specify the character that acts as the decimal separator (period by default).
- **Default**. The default value for a column. This is used for data written by a Generate stage. It also supplies the value to substitute for a column that causes an error (whether written or read).
- **Data Format**. Specifies the data representation format of a column. Choose from:
  - binary
  - text

    For decimals, binary means packed. Text 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. The destination string format is: [+ | -]ddd.[ddd] and any precision and scale arguments are ignored.
- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set,

calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.

- **Field width**. The number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.
- **Packed**. Select an option to specify what the decimal columns contain, choose from:
  - **Yes** to specify that the decimal columns contain data in packed decimal format (the default). This has the following sub-properties:

    Check. Select **Yes** to verify that data is packed, or **No** to not verify.

    Signed. Select **Yes** to use the existing sign when writing decimal columns. Select **No** to write a positive sign (0xf) regardless of the columns' actual sign value.
  - **No (separate)** to specify that they contain unpacked decimal with a separate sign byte. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - **No (zoned)** to specify that they contain an unpacked decimal in either ASCII or EBCDIC text. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - **No (overpunch)** to specify that the field has a leading or end 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 has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
- **Precision**. Specifies the precision where a decimal column is represented in text format. Enter a number. When a decimal is written to a string representation, InfoSphere DataStage 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. When they are defined, InfoSphere DataStage truncates or pads the source decimal to fit the size of the destination string. If you have also specified the field width property, InfoSphere DataStage truncates or pads the source decimal to fit the size specified by field width.
- **Rounding**. Specifies how to round the source field to fit into the destination decimal when reading a source field to a decimal. Choose from:
  - up (ceiling). Truncate source column towards positive infinity. This mode corresponds to the IEEE 754 Round Up mode. For example, 1.4 becomes 2, -1.6 becomes -1.
  - down (floor). Truncate source column towards negative infinity. This mode corresponds to the IEEE 754 Round Down mode. For example, 1.6 becomes 1, -1.4 becomes -2.
  - nearest value. Round the source column towards the nearest representable value. This mode corresponds to the COBOL ROUNDED mode. For example, 1.4 becomes 1, 1.5 becomes 2, -1.4 becomes -1, -1.5 becomes -2.
  - truncate towards zero. This is the default. Discard fractional digits to the right of the right-most fractional digit supported by 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. Using this method 1.6 becomes 1, -1.6 becomes -1.
- **Scale**. Specifies how to round a source decimal when its precision and scale are greater than those of the destination. By default, when the InfoSphere DataStage writes 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, InfoSphere DataStage truncates or pads the source decimal to fit the size of the destination string. If you have also specified the field width property, InfoSphere DataStage

truncates or pads the source decimal to fit the size specified by field width. Specifies how to round a source decimal when its precision and scale are greater than those of the destination.

Float type

- **C_format**. Perform non-default conversion of data from a string to floating-point data. This property specifies a C-language format string used for reading floating point strings. This is passed to *sscanf()*.
- **Character Set**. Choose from ASCII or EBCDIC.
- **Default**. The default value for a column. This is used for data written by a Generate stage. It also supplies the value to substitute for a column that causes an error (whether written or read).
- **Data Format**. Specifies the data representation format of a column. Choose from:
  - binary
  - text
- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.
- **Field width**. The number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.
- **In_format**. Format string used for conversion of data from string to floating point. This is passed to *sscanf*(). By default, InfoSphere DataStage invokes the C *sscanf*() function to convert a numeric field formatted as a string to floating point data. If this function does not output data in a satisfactory format, you can specify the in_format property to pass formatting arguments to *sscanf*().
- **Is link field**. Selected to indicate that a column holds the length of a another, variable-length column of the record or of the tag value of a tagged record field.
- **Out_format**. Format string used for conversion of data from floating point to a string. This is passed to *sprintf*(). By default, InfoSphere DataStage invokes the C *sprintf*() function to convert a numeric field formatted as floating point data to a string. If this function does not output data in a satisfactory format, you can specify the out_format property to pass formatting arguments to *sprintf*().
- **Pad char**. Specifies the pad character used when the floating point number is written to an external string representation. Enter a character (single-bye for strings, can be multi-byte for ustrings) or choose null or space. The pad character is used when the external string representation is larger than required to hold the written field. In this case, the external string is filled with the pad character to its full length. Space is the default.

Nullable

This appears for nullable fields.

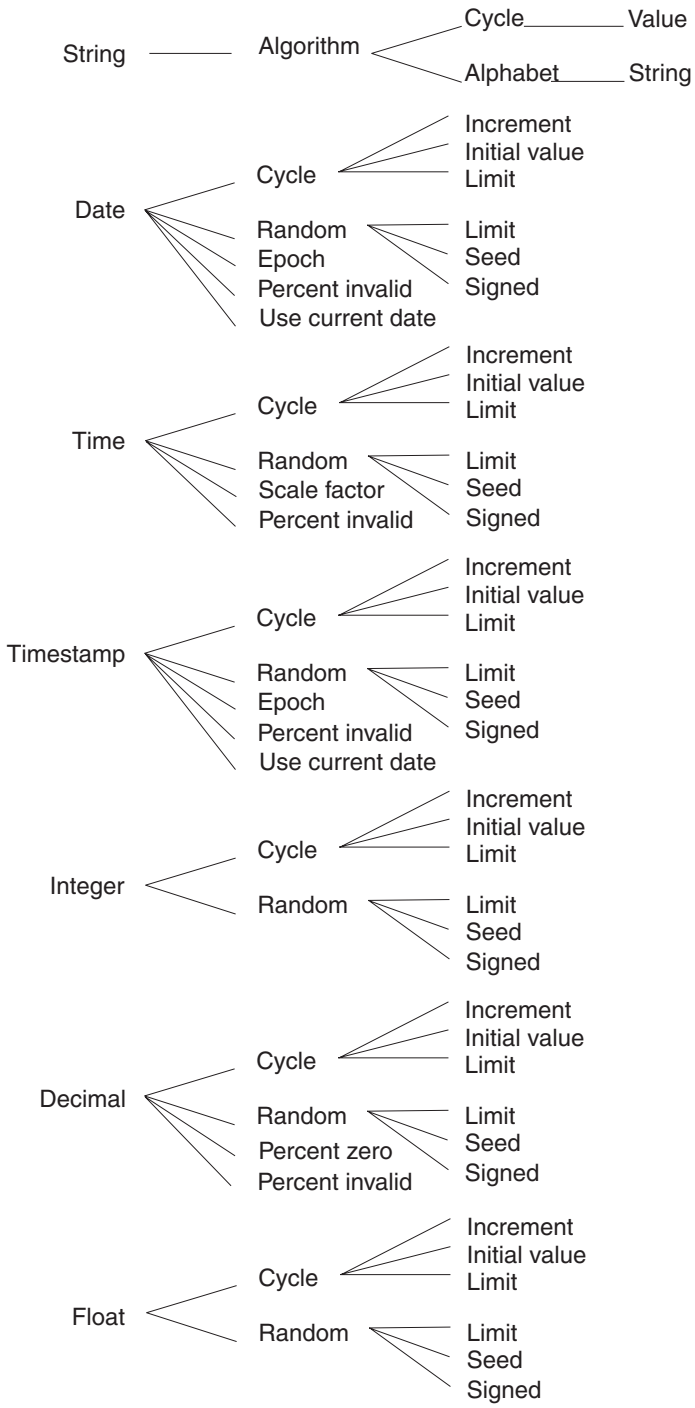
- **Actual field length**. Specifies the number of bytes to fill with the Fill character when a field is identified as null. When InfoSphere DataStage identifies a null field, it will write a field of this length full of Fill characters. This is mutually exclusive with Null field value.
- **Null field length**. The length in bytes of a variable-length field that contains a null. When a variable-length field is read, a length of null field length in the source field indicates that it contains a null. When a variable-length field is written, InfoSphere DataStage writes a length value of null field length if the field contains a null. This property is mutually exclusive with null field value.
- **Null field value**. Specifies the value given to a null field if the source is set to null. Can be a number, string, or C-type 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.

This property is mutually exclusive with Null field length and Actual length. For a fixed width data representation, you can use Pad char (from the general section of Type defaults) to specify a repeated trailing character if the value you specify is shorter than the fixed width of the field. On reading, specifies the value given to a field containing a null. On writing, specifies the value given to a field if the source is set to null. Can be a number, string, or C-type literal escape character.

Generator

If the column is being used in a Row Generator or Column Generator stage, this allows you to specify extra details about the mock data being generated. The exact fields that appear depend on the data type of the column being generated. They allow you to specify features of the data being generated, for example, for integers they allow you to specify if values are random or whether they cycle. If they cycle you can specify an initial value, an increment, and a limit. If they are random, you can specify a seed value for the random number generator, whether to include negative numbers, and a limit

The diagram below shows the Generate options available for the various data types:



All data types

All data types other than string have two **Types** of operation, cycle and random:

- **Cycle**. The cycle option generates a repeating pattern of values for a column. It has the following optional dependent properties:
  - **Increment**. The increment value added to produce the field value in the next output record. The default value is 1 (integer) or 1.0 (float).
  - **Initial value**. is the initial field value (value of the first output record). The default value is 0.
  - **Limit**. The maximum field value. When the generated field value is greater than *Limit*, it wraps back to *Initial value*. The default value of *Limit* is the maximum allowable value for the field's data type.

You can set these to `part' to use the partition number (for example, 0, 1, 2, 3 on a four node system), or `partcount' to use the total number of executing partitions (for example, 4 on a four node system).

- **Random**. The random option generates random values for a field. It has the following optional dependent properties:
  - **Limit**. Maximum generated field value. The default value of *limit* is the maximum allowable value for the field's data type.
  - **Seed**. The seed value for the random number generator used by the stage for the field. You do not have to specify *seed*. By default, the stage 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* and *+limit*). Otherwise, the operator creates values between 0 and *+limit.*

    You can *limit* and *seed* to `part' to use the partition number (for example, 0, 1, 2, 3 on a four node system), or `partcount' to use the total number of executing partitions (for example, 4 on a four node system).

Strings

By default the generator stages initialize all bytes of a string field to the same alphanumeric character. The stages use the following characters, in the following order:

```
abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

For example, the following a string with a length of 5 would produce 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.

You can also use the algorithm property to determine how string values are generated, this has two possible values: cycle and alphabet:

- **Cycle**. Values are assigned to a generated string field as a set of discrete string values to cycle through. This has the following dependent property:
  - **Values**. Repeat this property to specify the string values that the generated data cycles through.
- **Alphabet**. Values are assigned to a generated string field as a character string each of whose characters is taken in turn. This is like the default mode of operation except that you can specify the string cycled through using the dependent property **String**.

Decimal

As well as the **Type** property, decimal columns have the following properties:

- **Percent invalid**. The percentage of generated columns that will contain invalid values. Set to 10% by default.
- **Percent zero**. The percentage of generated decimal columns where all bytes of the decimal are set to binary zero (0x00). Set to 10% by default.

Date

As well as the **Type** property, date columns have the following properties:

- **Epoch**. Use this to specify the earliest generated date value, in the format *yyyy-mm-dd* (leading zeros must be supplied for all parts). The default is 1960-01-01.

- **Percent invalid**. The percentage of generated columns that will contain invalid values. Set to 10% by default.
- **Use current date**. Set this to generate today's date in this column for every row generated. If you set this all other properties are ignored.

Time

As well as the **Type** property, time columns have the following properties:

- **Percent invalid**. The percentage of generated columns that will contain invalid values. Set to 10% by default.
- **Scale factor**. Specifies a multiplier to the increment value for time. For example, a scale factor of 60 and an increment of 1 means the field increments by 60 seconds.

Timestamp

As well as the **Type** property, time columns have the following properties:

- **Epoch**. Use this to specify the earliest generated date value, in the format *yyyy-mm-dd* (leading zeros must be supplied for all parts). The default is 1960-01-01.
- **Use current date**. Set this to generate today's date in this column for every row generated. If you set this all other properties are ignored.
- **Percent invalid**. The percentage of generated columns that will contain invalid values. Set to 10% by default.
- **Scale factor**. Specifies a multiplier to the increment value for time. For example, a scale factor of 60 and an increment of 1 means the field increments by 60 seconds.

Vectors

If the row you are editing represents a column which is a variable length vector, tick the **Variable** check box. The Vector properties appear, these give the size of the vector in one of two ways:

- **Link Field Reference**. The name of a column containing the number of elements in the variable length vector. This should have an integer or float type, and have its Is Link field property set.
- **Vector prefix**. Specifies 1-, 2-, or 4-byte prefix containing the number of elements in the vector.

If the row you are editing represents a column which is a vector of known length, enter the number of elements in the **Vector Occurs** box.

Subrecords

If the row you are editing represents a column which is part of a subrecord the Level Number column indicates the level of the column within the subrecord structure.

If you specify Level numbers for columns, the column immediately preceding will be identified as a subrecord. Subrecords can be nested, so can contain further subrecords with higher level numbers (that is, level 06 is nested within level 05). Subrecord fields have a Tagged check box to indicate that this is a tagged subrecord.

Extended

For certain data types the **Extended** check box appears to allow you to modify the data type as follows:

- **Char, VarChar, LongVarChar**. Select to specify that the underlying data type is a ustring.
- **Time**. Select to indicate that the time field includes microseconds.
- **Timestamp**. Select to indicate that the timestamp field includes microseconds.

- **TinyInt**, **SmallInt**, **Integer**, **BigInt** types. Select to indicate that the underlying data type is the equivalent uint field.

Inputs page - Advanced tab

The Advanced tab allows you to specify how InfoSphere DataStage buffers data being input this stage. By default InfoSphere DataStage buffers data in such a way that no deadlocks can arise; a deadlock being the situation where a number of stages are mutually dependent, and are waiting for input from another stage and cannot output until they have received it.

The size and operation of the buffer are usually the same for all links on all stages (the default values that the settings take can be set using environment variables).

The Advanced tab allows you to specify buffer settings on a per-link basis. You should only change the settings if you fully understand the consequences of your actions (otherwise you might cause deadlock situations to arise).

Any changes you make on this tab will automatically be reflected in the Output Page Advanced tab of the stage at the other end of this link.

The settings are as follows:

- **Buffering mode**. Select one of the following from the drop-down list.
  - **(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 output data only if necessary to prevent a dataflow deadlock situation.
  - **Buffer**. This will unconditionally buffer all data output from this stage.
  - **No buffer**. Do not buffer output data under any circumstances. This could potentially lead to deadlock situations if not used carefully.

If you choose the Auto buffer or Buffer options, you can also set the values of the various buffering parameters:

- **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.
- **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.

Output page

The Output page gives information about links going out of a stage. In the case of a file or database stage an output link carries data being read from the file or database. In the case of a processing or restructure stage it carries data that the stage has processed. Where there are no output links the stage editor has no Output page.

Where it is present, the Output page contains various tabs depending on stage type. The only field the Output page itself contains is **Output name**, which gives the name of the link being edited. Where a stage has more than one output link, you can select the link you are editing from the **Output name** drop-down list.

The Output page also has a **Columns...** button. Click **Columns...** to open a window showing column names from the meta data defined for this link. You can drag these columns to various fields in the Output page tabs as required.

Certain stage types will also have a **View Data...** button. Press this to view the actual data associated with the specified data source or data target. The button is available if you have defined meta data for the link.

The Sequential File stage has a **Show File...** button, rather than **View Data...** . This shows the flat file as it has been created on disk.

General tab

The Output page always has a General tab. this allows you to enter an optional description of the link. Specifying a description for each link enhances job maintainability.

Properties tab

Some types of file and database stages can have properties that are particular to specific output links. In this case the Output page has a Properties tab. This has the same format as the Stage page Properties tab (see "Properties Tab").

Format tab

Stages that read from certain types of file (for example, the Sequential File stage) also have a Format tab which allows you to specify the format of the file or files being read from.

The Format page is similar in structure to the Properties page. A flat file has a number of properties that you can set different attributes for. Select the property in the tree and select the attributes you want to set from the **Available properties to add** window, it will then appear as a dependent property in the property tree and you can set its value as required. This tab sets the format information for the file at row level. You can override the settings for individual columns using the Edit Column Metadata dialog box (see Field Level).

Format details are also stored with table definitions, and you can use the **Load...** button to load a format from a table definition stored in the Repository.

The short-cut menu from the property tree gives access to the following functions:

- **Format as**. This applies a predefined template of properties. Choose from the following:

  - Delimited/quoted
  - Fixed-width records
  - UNIX line terminator
  - DOS line terminator
  - No terminator (fixed width)
  - Mainframe (COBOL)
- **Add sub-property**. Gives access to a list of dependent properties for the currently selected property (visible only if the property has dependents).
- **Set to default**. Appears if the currently selected property has been set to a non-default value, allowing you to re-select the default.
- **Remove**. Removes the currently selected property. This is disabled if the current property is mandatory.
- **Remove all**. Removes all the non-mandatory properties.

Columns tab

The Output page always has a Columns tab. This displays the column meta data for the selected output link in a grid.

There are various ways of populating the grid:

- If the other end of the link has meta data specified for it, this will be displayed in the Columns tab (meta data is associated with, and travels with a link).
- You can type the required meta data into the grid. When you have done this, you can click the **Save...** button to save the meta data as a table definition in the Repository for subsequent reuse.
- You can load an existing table definition from the Repository. Click the **Load...** button to be offered a choice of table definitions to load.
- If the stage you are editing is a general or restructure stage with a Mapping tab, you can drag data from the left pane to the right pane. This automatically populates the right pane and the Columns tab.

If runtime column propagation is enabled in the InfoSphere DataStage Administrator, you can select the **Runtime column propagation** option to specify that columns encountered by the stage can be used even if they are not explicitly defined in the meta data. There are some special considerations when using runtime column propagation with certain stage types:

- Sequential File
- File Set
- External Source
- External Target

See the individual stage descriptions for details of these.

If the selected output link is a reject link, the column meta data grid is read only and cannot be modified.

If you select the options in the Grid Properties dialog box (see*InfoSphere DataStage Designer Client Reference Guide*), the **Columns** tab will also display two extra fields: Table Definition Reference and Column Definition Reference. These show the table definition and individual columns that the columns on the tab were derived from.

If you click in a row and select **Edit Row...** from the shortcut menu, the Edit Column meta data dialog box appears, which allows you edit the row details in a dialog box format. It also has a Parallel tab which allows you to specify properties that are peculiar to parallel job column definitions. The properties you can specify here are the same as those specified for input links).

Mapping tab

For processing and restructure stages the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them or how they are generated.

The left pane shows the input columns or the generated columns. These are read only and cannot be modified on this tab. These columns represent the data that the stage has produced after it has processed the input data.

The right pane shows the output columns for each link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility. If you have not yet defined any output column definitions, dragging columns over will define them for you. If you have already defined output column definitions, InfoSphere DataStage performs the mapping for you as far as possible: you can do this explicitly using the auto-match facility, or implicitly by just visiting the Mapping tab and clicking **OK** (which is the equivalent of auto-matching on name).

There is also a shortcut menu which gives access to a range of column selection and editing functions, including the facilities for selecting multiple columns and editing multiple derivations (this functionality is described in the Transformer chapter).

You might choose not to map all the left hand columns, for example if your output data is a subset of your input data, but be aware that, if you have Runtime Column Propagation turned on for that link, the data you have not mapped will appear on the output link anyway.

You can also perform mapping without actually opening the stage editor. Select the stage in the Designer canvas and choose **Auto-map** from the shortcut menu.

More details about mapping operations for the different stages are given in the individual stage descriptions:

Stage	Chapter	Stage	Chapter
Aggregator	"Aggregator stage" on page 226	Change Capture	"Change Capture stage" on page 339
Join	"Join stage" on page 238	Change Apply	"Change Apply stage" on page 347
Funnel	"Funnel Stage" on page 283	Difference	"Difference stage" on page 355
Lookup	"Lookup Stage" on page 254	Column Import	"Column Import stage" on page 427
Sort	"Sort stage" on page 271	Column Export	"Column Export stage" on page 441
Merge	"Merge Stage" on page 246	Head	"Head stage" on page 502
Remove Duplicates	"Remove Duplicates Stage" on page 291	Tail	"Tail stage" on page 508
Sample	"Sample stage" on page 513	Peek	"Peek stage" on page 522
Column Generator	"Column Generator stage" on page 533	SAS	*SAS Connectivity Guide*
Copy	"Copy stage" on page 303		

A shortcut menu can be invoked from the right pane that allows you to:

- Find and replace column names.

- Validate a derivation you have entered.
- Clear an existing derivation.
- Append a new column.
- Select all columns.
- Insert a new column at the current position.
- Delete the selected column or columns.
- Cut and copy columns.
- Paste a whole column.
- Paste just the derivation from a column.

The **Find** button opens a dialog box which allows you to search for particular output columns.

The **Auto-Match** button opens a dialog box which will automatically map left pane columns onto right pane columns according to the specified criteria.

Select **Location match** to map input columns onto the output ones occupying the equivalent position. Select **Name match** to match by names. You can specify that all columns are to be mapped by name, or only the ones you have selected. You can also specify that prefixes and suffixes are ignored for input and output columns, and that case can be ignored.

Advanced tab

The Advanced tab allows you to specify how InfoSphere DataStage buffers data being output from this stage. By default InfoSphere DataStage buffers data in such a way that no deadlocks can arise; a deadlock being the situation where a number of stages are mutually dependent, and are waiting for input from another stage and cannot output until they have received it.

The size and operation of the buffer are usually the same for all links on all stages (the default values that the settings take can be set using environment variables).

The Advanced tab allows you to specify buffer settings on a per-link basis. You should only change the settings if you fully understand the consequences of your actions (otherwise you might cause deadlock situations to arise).

Any changes you make on this tab will automatically be reflected in the Input page Advanced tab of the stage at the other end of this link

The settings are as follows:

- **Buffering mode**. Select one of the following from the drop-down list.
  - **(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 output data only if necessary to prevent a dataflow deadlock situation.
  - **Buffer**. This will unconditionally buffer all data output from this stage.
  - **No buffer**. Do not buffer output data under any circumstances. This could potentially lead to deadlock situations if not used carefully.

If you choose the Auto buffer or Buffer options, you can also set the values of the various buffering parameters:

- **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.
- **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.

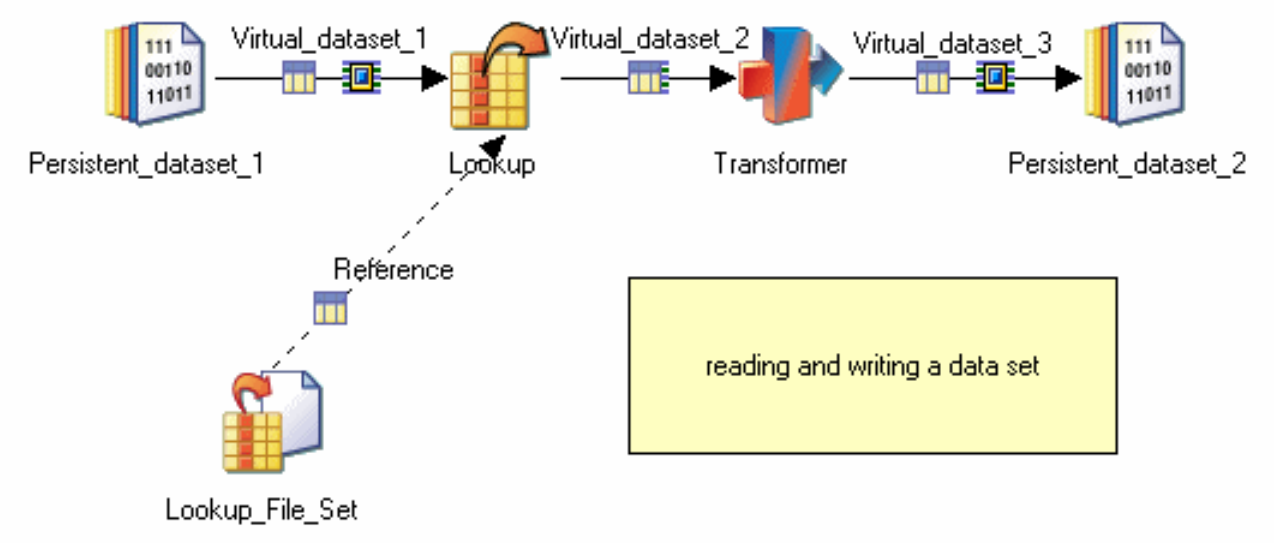
Chapter 5. Reading and Writing Files

Use the stages in the File section of the palette to read and write data from files.

Data set stage

The Data Set stage is a file stage. It allows you to read data from or write data to a data set. The stage can have a single input link or a single output link. It can be configured to execute in parallel or sequential mode.

What is a data set? parallel jobs use data sets to manage data within a job. You can think of each link in a job as carrying a data set. The Data Set stage allows you to store data being operated on in a persistent form, which can then be used by other InfoSphere DataStage jobs. Data sets are operating system files, each referred to by a control file, which by convention has the suffix .ds. Using data sets wisely can be key to good performance in a set of linked jobs. You can also manage data sets independently of a job using the Data Set Management utility, available from the InfoSphere DataStage Designer or Director.



The stage editor has up to three pages, depending on whether you are reading or writing a data set:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is present when you are writing to a data set. This is where you specify details about the data set being written to.
- **Output Page**. This is present when you are reading from a data set. This is where you specify details about the data set being read from.

Data Set stage: fast path

This section specifies the minimum steps to take to get a Data Set stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Data Set stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic methods, you will learn where the shortcuts are when you get familiar with the product.

The steps required depend on whether you are using the Data Set stage to read or write a data set.

Writing to a data set

About this task

- In the **Input Link Properties Tab** specify the pathname of the control file for the target data set. Set the Update Policy property, or accept the default setting of Overwrite.
- Ensure column meta data has been specified for the data set (this might have already been done in a preceding stage).

Reading from a data set

About this task

- In the **Output Link Properties Tab** specify the pathname of the control file for the source data set.
- Ensure column meta data has been specified for the data set.

Data Set stage: Stage page

The General tab allows you to specify an optional description of the stage. The Advanced tab allows you to specify how the stage executes.

Data Set stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the contents of the data set are processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire contents of the data set are processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. You can select **Propagate**, **Set** or **Clear**. If you select **Set** file read operations will request that the next stage preserves the partitioning as is. Propagate takes the setting of the flag from the previous stage.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Data Set stage: Input page

The Input page allows you to specify details about how the Data Set stage writes data to a data set. The Data Set stage can have only one input link.

The General tab allows you to specify an optional description of the input link. The **Properties** tab allows you to specify details of exactly what the link does. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Data Set stage properties are given in the following sections. See "Stage Editors," for a general description of the other tabs.

Data Set stage: Input link properties tab

The **Properties** tab allows you to specify properties for the input link. These dictate how incoming data is written and to what data set. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows:

Table 11. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Target/File	pathname	N/A	Y	N	N/A
Target/Update Policy	Append/Create (Error if exists)/ Overwrite/Use existing (Discard records)/Use existing (Discard records and schema)	Overwrite	Y	N	N/A

Data Set stage: Target category: File

The name of the control file for the data set. You can browse for the file or enter a job parameter. By convention, the file has the suffix `.ds`.

Update Policy

Specifies what action will be taken if the data set you are writing to already exists. Choose from:

- **Append**. Append any new data to the existing data.
- **Create** (Error if exists). InfoSphere DataStage reports an error if the data set already exists.
- **Overwrite**. Overwrites any existing data with new data.
- **Use existing (Discard records)**. Keeps existing files listed in a descriptor file (for example, `datasetname.ds` or `filesetname.fs`) but discards the old records. You receive an error if the data set with different schema already exists.
- **Use existing (Discard records and schema)**. Keeps existing files listed in a descriptor file (for example, `datasetname.ds` or `filesetname.fs`) but discards the old schema and records.

The default is **Overwrite**.

Data Set stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is written to the data set. It also allows you to specify that the data should be sorted before being written.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Data Set stage is operating in sequential mode, it will first collect the data before writing it to the file using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Data Set stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Data Set stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Data Set stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Data Set stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Data Set stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being written to the data set. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with Auto methods).

Select the check boxes as follows:

- **Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Data Set stage: Output page

The Output page allows you to specify details about how the Data Set stage reads data from a data set. The Data Set stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The **Properties** tab allows you to specify details of exactly what the link does. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Data Set stage properties and formatting are given in the following sections. See "Stage Editors," for a general description of the other tabs.

Data Set stage: Output link properties tab

The Properties tab allows you to specify properties for the output link. These dictate how incoming data is read from the data set. A Data Set stage only has one property, but this is mandatory.

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Source/File	pathname	N/A	Y	N	N/A

Data Set stage: Source category:
File

The name of the control file for the data set. You can browse for the file or enter a job parameter. By convention the file has the suffix .ds.

Data Set stage: Options category:

Use these properties to specify the options about how the Data Set stage operates.

The following property is available:

Missing Columns Mode

Use this option to specify how the stage behaves if columns defined in the stage are not present in the data set when the job runs. Select one of the following options:

Ignore The job fails. If runtime column propagation is off, the job warns at the Data Set stage. The job fails when that column is explicitly used by another stage.

Fail The job fails at the Data Set stage, regardless of whether runtime column propagation is on or off.

Default Nullable Only

The job sets any missing columns that are marked as nullable to the null value. Any missing columns marked as not nullable will cause the job to fail.

Default Non-Nullable Only

The job sets any missing columns that are marked as not nullable to the default value for that data type (for example, an integer column defaults to 0). Any missing columns marked as nullable will cause the job to fail.

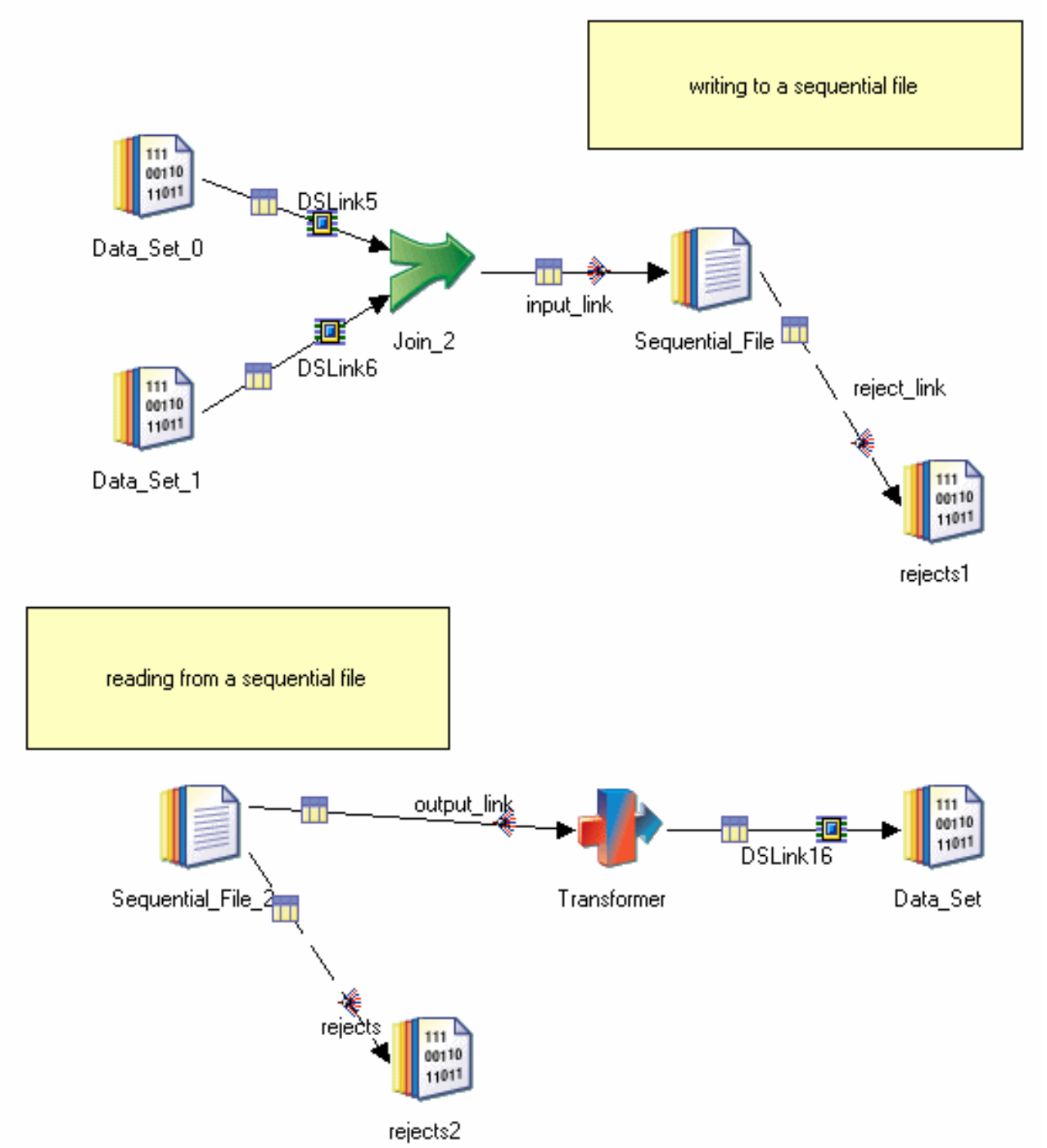
Default All

The job sets values for missing columns as follows:

- Nullable columns are set to null.
- Non-nullable columns are set to the default value for that data type (for example, an integer column defaults to 0).

Sequential file stage

The Sequential File stage is a file stage. It allows you to read data from or write data one or more flat files. The stage can have a single input link or a single output link, and a single rejects link.



When you edit a Sequential File stage, the Sequential File stage editor appears. This is based on the generic stage editor described in "Stage Editors."

The stage executes in parallel mode if reading multiple files but executes sequentially if it is only reading one file. By default a complete file will be read by a single node (although each node might read more than one file). For fixed-width files, however, you can configure the stage to behave differently:

- You can specify that single files can be read by multiple nodes. This can improve performance on cluster systems. See "Read From Multiple Nodes"

- You can specify that a number of readers run on a single node. This means, for example, that a single file can be partitioned as it is read (even though the stage is constrained to running sequentially on the conductor node). See "Number Of Readers Per Node".

(These two options are mutually exclusive.)

The stage executes in parallel if writing to multiple files, but executes sequentially if writing to a single file. Each node writes to a single file, but a node can write more than one file.

When reading or writing a flat file, InfoSphere DataStage needs to know something about the format of the file. The information required is how the file is divided into rows and how rows are divided into columns. You specify this on the Format tab. Settings for individual columns can be overridden on the Columns tab using the Edit Column Metadata dialog box.

The stage editor has up to three pages, depending on whether you are reading or writing a file:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is present when you are writing to a flat file. This is where you specify details about the file or files being written to.
- **Output Page**. This is present when you are reading from a flat file or have a reject link. This is where you specify details about the file or files being read from.

There are one or two special points to note about using runtime column propagation (RCP) with Sequential stages. See “Using RCP with Sequential File stages” on page 111 for details.

Example of writing a sequential file

In the following example, the Sequential File stage is set up to write a comma-delimited file. Here is a sample of the data as it will be written:

```
"GC13849","JON SMITH","789 LEDBURY ROAD","2/17/2007"
"GC13933","MARY GARDENER","127 BORDER ST","8/28/2009"
"GC14036","CHRIS TRAIN","1400 NEW ST","9/7/1998"
"GC14127","HUW WILLIAMS","579 DIGBETH AVENUE","6/29/2011"
"GC14263","SARA PEARS","45 ALCESTER WAY","4/12/2008"
"GC14346","LUC TEACHER","3 BIRMINGHAM ROAD","11/7/2010"
```

The metadata for the file is defined in the Columns tab as follows:

Table 12. Example metadata for writing to a sequential file

Column name	Key	SQL Type	Length	Nullable
CUSTOMER_NUMBER	yes	char	7	no
CUST_NAME		varchar	30	no
ADDR_1		varchar	30	no
SETUP_DATE		char	10	no

The Format tab is set as follows to define that the stage will write a file where each column is delimited by a comma, there is no final delimiter, and any dates in the data are expected to have the format *mm/dd/yyyy* with or without leading zeroes for month and day, rather than *yyyy-mm-dd*, which is the default format:

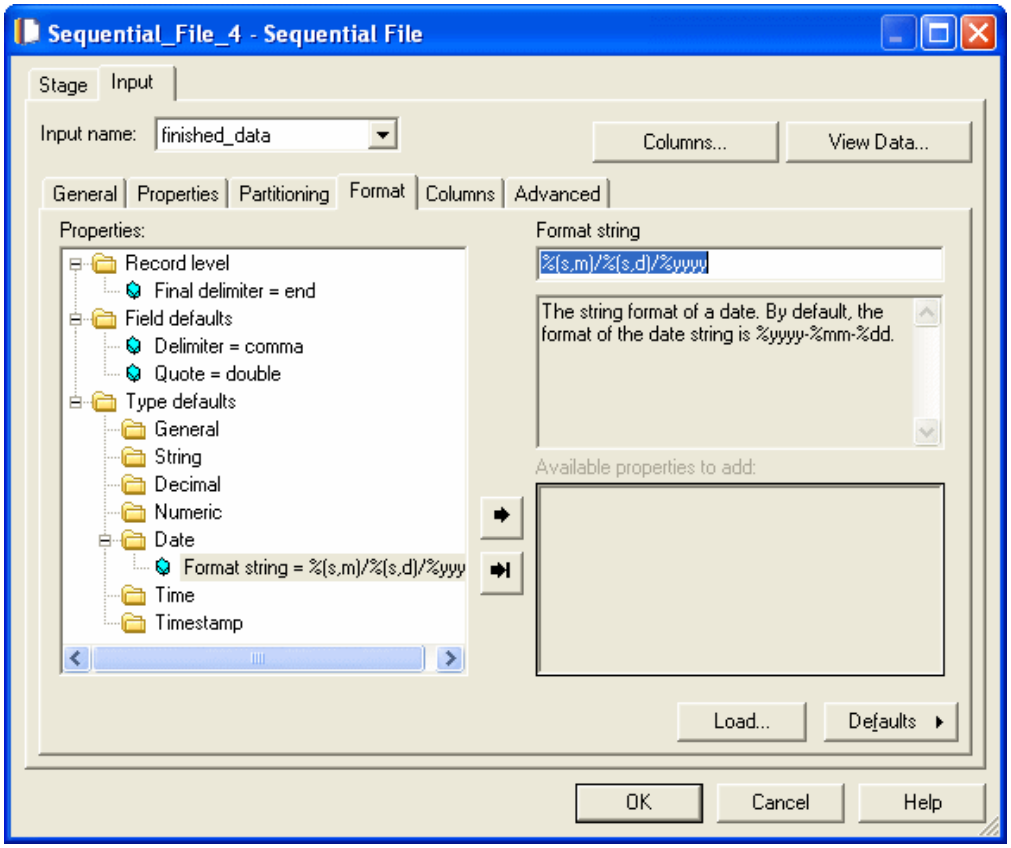


Figure 4. Format tab

Example of reading a sequential file

In the following example, the sequential file stage is set up to read a fixed width file. Here is a sample of the data in the file:

```
0136.801205/04/2001
0210.001525/04/2001
0316.803002/01/2002
0414.704002/01/2002
0517.200202/01/2002
0616.003002/01/2002
0744.001012/08/2001
0814.403002/01/2002
0950.002502/01/2002
1026.201012/08/2001
1120.701012/08/2001
1239.401012/08/2001
1310.000302/01/2002
1412.000502/01/2002
1528.800102/01/2002
1636.802021/06/2001
```

The meta data for the file is defined in the Columns tab as follows:

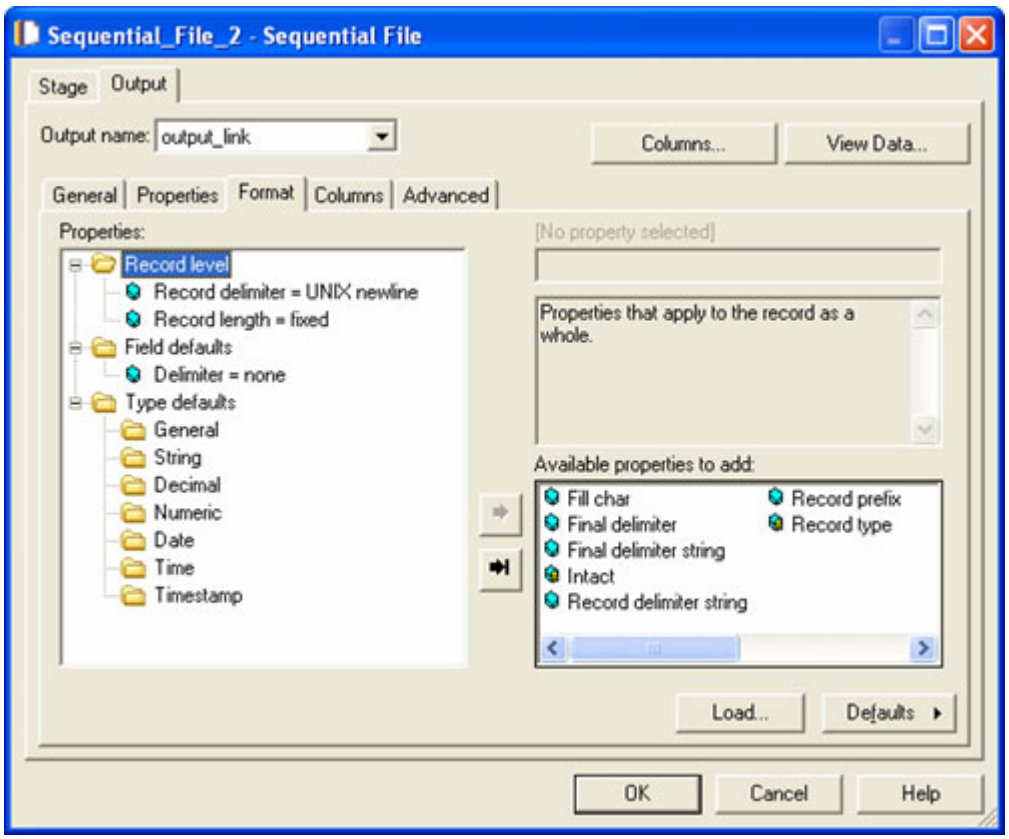
Table 13. Example metadata for reading a sequential file

Column name	Key	SQL Type	Length	Nullable
OrderID	yes	char	2	no

Table 13. Example metadata for reading a sequential file (continued)

Column name	Key	SQL Type	Length	Nullable
Price		char	5	no
Quantity		char	2	no
Order_Date		char	10	no

The Format tab is set as follows to define that the stage is reading a fixed width file where each row is delimited by a UNIX newline, and the columns have no delimiter:



Sequential File stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Sequential File stages in a job. This section specifies the minimum steps to take to get a Sequential File stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

The steps required depend on whether you are using the Sequential File stage to read or write a file.

Writing to a file

About this task

- In the **Input Link Properties Tab** specify the pathname of the file being written to (repeat this for writing to multiple files). The other properties all have default values, which you can change or not as required.

- In the **Input Link Format Tab** specify format details for the file(s) you are writing to, or accept the defaults (variable length columns enclosed in double quotes and delimited by commas, rows delimited with UNIX newlines).
- Ensure column meta data has been specified for the file(s) (this can be achieved via a schema file if required).

Reading from a file

About this task

- In the **Output Link Properties Tab**:
  - In Read Method, specify whether to read specific files (the default) or all files whose name fits a pattern.
  - If you are reading specific files, specify the pathname of the file being read from (repeat this for reading multiple files).
  - If you are reading files that fit a pattern, specify the name pattern to match.
  - Accept the default for the options or specify new settings (available options depend on the Read Method).
- In the **Output Link Format Tab** specify format details for the file(s) you are reading from, or accept the defaults (variable length columns enclosed in double quotes and delimited by commas, rows delimited with UNIX newlines).
- Ensure column meta data has been specified for the file(s) (this can be achieved via a schema file if required).

Sequential File stage: Stage page

The General tab allows you to specify an optional description of the stage. The Advanced tab allows you to specify how the stage executes. The NLS Map tab appears if you have NLS enabled on your system, it allows you to specify a character set map for the stage.

Sequential File stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. When a stage is reading or writing a single file the **Execution Mode** is sequential and you cannot change it. When a stage is reading or writing multiple files, the **Execution Mode** is parallel and you cannot change it. In parallel mode, the files are processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the **Advanced** tab. In Sequential mode the entire contents of the file are processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. You can select **Set** or **Clear**. If you select **Set** file read operations will request that the next stage preserves the partitioning as is (it is ignored for file write operations). If you set the Keep File Partitions output property this will automatically set the preserve partitioning flag.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Sequential File stage: NLS Map tab

The NLS Map tab allows you to define a character set map for the Sequential File stage. This overrides the default character set map set for the project or the job. You can specify that the map be supplied as a job parameter if required. You can also select **Allow per-column mapping**. This allows character set maps to be specified for individual columns within the data processed by the Sequential File stage. An extra property, **NLS Map**, appears in the Columns grid in the Columns tab, but note that only ustring data types allow you to set an NLS map value (see "Data Types").

Sequential File stage: Input page

The Input page allows you to specify details about how the Sequential File stage writes data to one or more flat files. The Sequential File stage can have only one input link, but this can write to multiple files.

The General tab allows you to specify an optional description of the input link. The Properties tab allows you to specify details of exactly what the link does. The Partitioning tab allows you to specify how incoming data is partitioned before being written to the file or files. The Formats tab gives information about the format of the files being written. The Columns tab specifies the column definitions of data being written. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Sequential File stage properties, partitioning, and formatting are given in the following sections. See "Stage Editors," for a general description of the other tabs.

Sequential File stage: Input link properties tab

The Properties tab allows you to specify properties for the input link. These dictate how incoming data is written and to what files. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 14. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Target/File	Pathname	N/A	Y	Y	N/A
Target/File Update Mode	Append/ Create/ Overwrite	Overwrite	Y	N	N/A
Options/Cleanup On Failure	True/False	True	Y	N	N/A
Options/“First Line is Column Names” on page 91	True/False	False	N	N	N/A
Options/Reject Mode	Continue/Fail/ Save	Continue	Y	N	N/A
Options/Filter	Command	N/A	N	N	N/A
Options/Schema File	Pathname	N/A	N	N	N/A

Sequential File stage: Target category:

File

This property defines the flat file that the incoming data will be written to. You can type in a pathname, or browse for a file. You can specify multiple files by repeating the File property. Do this by selecting the **Properties** item at the top of the tree, and clicking on **File** in the **Available properties to add** box. Do this for each extra file you want to specify.

You must specify at least one file to be written to, which must exist unless you specify a **File Update Mode** of **Create** or **Overwrite**.

File update mode

This property defines how the specified file or files are updated. The same method applies to all files being written to. Choose from **Append** to append to existing files, **Overwrite** to overwrite existing files, or **Create** to create a new file. If you specify the **Create** property for a file that already exists you will get an error at runtime.

By default this property is set to **Overwrite**.

Sequential File stage: Options category:
Cleanup on failure

This is set to **True** by default and specifies that the stage will delete any partially written files if the stage fails for any reason. Set this to **False** to specify that partially written files should be left.

First Line is Column Names

Specifies that the first line of the file contains column names. This property is false by default.

Reject mode

This specifies what happens to any data records that are not written to a file for some reason. Choose from **Continue** to continue operation and discard any rejected rows, **Fail** to cease writing if any rows are rejected, or **Save** to send rejected rows down a reject link.

Continue is set by default.

Filter

This is an optional property. You can use this to specify that the data is passed through a filter program before being written to the file or files. Specify the filter command, and any required arguments, in the Property Value box.

Schema file

This is an optional property. By default the Sequential File stage will use the column definitions defined on the **Columns** and **Format** tabs as a schema for writing to the file. You can, however, specify a file containing a schema instead (note, however, that if you have defined columns on the **Columns** tab, you should ensure these match the schema file). Type in a pathname or browse for a schema file.

Sequential File stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is written to the file or files. It also allows you to specify that the data should be sorted before being written.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Sequential File stage is operating in sequential mode, it will first collect the data before writing it to the file using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Sequential File stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Sequential File stage is set to execute in parallel (that is, is writing to multiple files), then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Sequential File stage is set to execute in sequential mode (that is, is writing to a single file), but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Sequential File stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Sequential File stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The **Partitioning** tab also allows you to specify that data arriving on the input link should be sorted before being written to the file or files. The sort is always carried out within data partitions. If the stage is

partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Sequential File stage: Input link format tab

The Format tab allows you to supply information about the format of the flat file or files to which you are writing. The tab has a similar format to the Properties tab.

If you do not alter any of the Format settings, the Sequential File stage will produce a file of the following format:

- File comprises variable length columns contained within double quotes.
- All columns are delimited by a comma, except for the final column in a row.
- Rows are delimited by a UNIX newline.

You can use the **Format As** item from the shortcut menu in the Format tab to quickly change to a fixed-width column format, using DOS newlines as row delimiters, or producing a COBOL format file.

You can use the **Defaults** button to change your default settings. Use the Format tab to specify your required settings, then click **Defaults** > **Save current as default**. All your sequential files will use your settings by default from now on. If your requirements change, you can choose **Defaults** > **Reset defaults from factory settings** to go back to the original defaults as described above. Once you have done this, you then have to click **Defaults** > **Set current from default** for the new defaults to take effect.

To change individual properties, select a property type from the main tree then add the properties you want to set to the tree structure by clicking on them in the **Available properties to addwindow**. You can then set a value for that property in the Property Value box. Pop-up help for each of the available properties appears if you hover the mouse pointer over it.

Any property that you set on this tab can be overridden at the column level by setting properties for individual columns on the Edit Column Metadata dialog box (see Columns Tab).

This description uses the terms "record" and "row" and "field" and "column" interchangeably.

The following sections list the property types and properties available for each type.

Record level

These properties define details about how data records are formatted in the flat file. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

Fill char

Specify an ASCII character or a value in the range 0 to 255. You can also choose Space or Null from a drop-down list. This character is used to fill any gaps in a written record caused by column positioning properties. Set to 0 by default (which is the NULL character). For example, to set it to space you could also type in the space character or enter 32. Note that this value is restricted to one byte, so you cannot specify a multi-byte Unicode character.

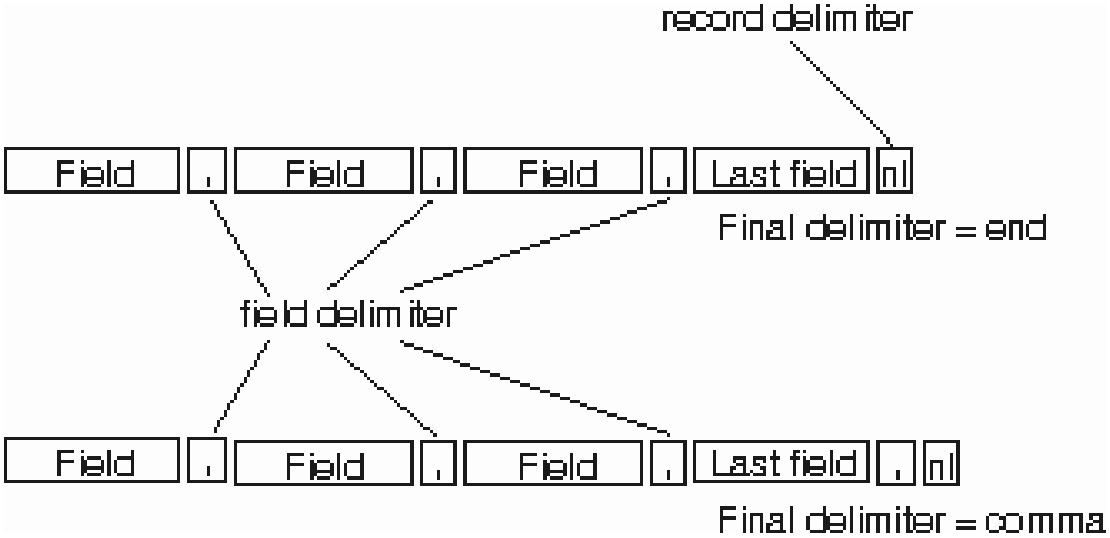
Final delimiter string

Specify a string to be written after the last column of a record in place of the column delimiter. Enter one or more characters, this precedes the record delimiter if one is used. Mutually exclusive with Final delimiter, which is the default. For example, if you set Delimiter to comma and Final delimiter string to `, ` (comma space - you do not need to enter the inverted commas) all fields are delimited by a comma, except the final field, which is delimited by a comma followed by an ASCII space character.

Final delimiter

Specify a single character to be written after the last column of a record in place of the field delimiter. Type a character or select one of whitespace, end, none, null, tab, or comma. See the following diagram for an illustration.

- **whitespace**. The last column of each record will not include any trailing white spaces found at the end of the record.
- **end**. The last column of each record does not include the field delimiter. This is the default setting.
- **none**. The last column of each record does not have a delimiter; used for fixed-width fields.
- **null**. The last column of each record is delimited by the ASCII null character.
- **comma**. The last column of each record is delimited by the ASCII comma character.
- **tab**. The last column of each record is delimited by the ASCII tab character.



When writing, a space is now inserted after every field except the last in the record. Previously, a space was inserted after every field including the last. (If you want to revert to the pre-release 7.5 behavior of inserting a space after the last field, set the APT_FINAL_DELIM_COMPATIBLE environment variable.

Intact

The intact property specifies an identifier of a partial schema. A partial schema specifies that only the column(s) named in the schema can be modified by the stage. All other columns in the row are passed through unmodified. The file containing the partial schema is specified in the **Schema File** property on the **Properties** tab. This property has a dependent property, Check intact, but this is not relevant to input links.

Record delimiter string

Specify a string to be written at the end of each record. Enter one or more characters. This is mutually exclusive with Record delimiter, which is the default, record type and record prefix.

Record delimiter

Specify a single character to be written at the end of each record. Type a character or select one of the following:

- UNIX Newline (the default)
- null

(To implement a DOS newline, use the Record delimiter string property set to "\R\N" or choose**Format as** > **DOS line terminator** from the shortcut menu.)

Note: Record delimiter is mutually exclusive with Record delimiter string, Record prefix, and Record type.

Record length

Select **Fixed** where fixed length fields are being written. InfoSphere DataStage calculates the appropriate length for the record. Alternatively specify the length of fixed records as number of bytes. This is not used by default (default files are comma-delimited). The record is padded to the specified length with either zeros or the fill character if one has been specified.

Record Prefix

Specifies that a variable-length record is prefixed by a 1-, 2-, or 4-byte length prefix. It is set to 1 by default. This is mutually exclusive with Record delimiter, which is the default, and record delimiter string and record type.

Record type

Specifies that data consists of variable-length blocked records (varying) or implicit records (implicit). If you choose the implicit property, data is written as a stream with no explicit record boundaries. The end of the record is inferred when all of the columns defined by the schema have been parsed. The varying property allows you to specify one of the following IBM blocked or spanned formats: **V**, **VB**, **VS**, **VBS**, or **VR**.

This property is mutually exclusive with Record length, Record delimiter, Record delimiter string, and Record prefix and by default is not used.

Field defaults

Defines default properties for columns written to the file or files. These are applied to all columns written, but can be overridden for individual columns from the **Columns** tab using the Edit Column Metadata dialog box. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Actual field length**. Specifies the number of bytes to fill with the Fill character when a field is identified as null. When InfoSphere DataStage identifies a null field, it will write a field of this length full of Fill characters. This is mutually exclusive with Null field value.
- **Delimiter**. Specifies the trailing delimiter of all fields in the record. Type an ASCII character or select one of whitespace, end, none, null, comma, or tab.
  - **whitespace**. Whitespace characters at the end of a column are ignored, that is, are not treated as part of the column.
  - **end**. The end of a field is taken as the delimiter, that is, there is no separate delimiter. This is not the same as a setting of `None' which is used for fields with fixed-width columns.
  - **none**. No delimiter (used for fixed-width).
  - **null**. ASCII Null character is used.
  - **comma**. ASCII comma character is used.
  - **tab**. ASCII tab character is used.
- **Delimiter string**. Specify a string to be written at the end of each field. Enter one or more characters. This is mutually exclusive with Delimiter, which is the default. For example, specifying `, ` (comma space - you do not need to enter the inverted commas) would have each field delimited by `, ` unless overridden for individual fields.
- **Null field length**. The length in bytes of a variable-length field that contains a null. When a variable-length field is written, InfoSphere DataStage writes a length value of null field length if the field contains a null. This property is mutually exclusive with null field value.
- **Null field value**. Specifies the value written to null field if the source is set to null. Can be a number, string, or C-type 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 \x*hh*, where each *h* is a hexadecimal digit 0 - F. You must use this form to encode non-printable byte values.

  This property is mutually exclusive with Null field length and Actual length. For a fixed width data representation, you can use Pad char (from the general section of Type defaults) to specify a repeated trailing character if the value you specify is shorter than the fixed width of the field.

  Null field value has a sub property named Null field value separator. This is intended for output data, and should be ignored on Format tabs belonging to input links.
- **Prefix bytes**. Specifies that each column in the data file is prefixed by 1, 2, or 4 bytes containing, as a binary value, either the column's length or the tag value for a tagged 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. InfoSphere DataStage inserts the prefix before each field.

  This property is mutually exclusive with the Delimiter, Quote, and Final Delimiter properties, which are used by default.
- **Print field**. This property is not relevant for input links.
- **Quote**. Specifies that variable length fields are enclosed in single quotes, double quotes, or another character or pair of characters. Choose **Single** or **Double**, or enter a character. This is set to double quotes by default.

  When writing, InfoSphere DataStage inserts the leading quote character, the data, and a trailing quote character. Quote characters are not counted as part of a field's length.
- **Vector prefix**. For fields that are variable length vectors, specifies a 1-, 2-, or 4-byte prefix containing the number of elements in the vector. You can override this default prefix for individual vectors.

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 this property to indicate the prefix length. InfoSphere DataStage inserts the element count as a prefix of each variable-length vector field. By default, the prefix length is assumed to be one byte.

Type defaults

These are properties that apply to all columns of a specific data type unless specifically overridden at the column level. They are divided into a number of subgroups according to data type.

General

These properties apply to several data types (unless overridden at column level):

- **Byte order**. Specifies how multiple byte data types (except string and raw data types) are ordered. Choose from:
  - little-endian. The high byte is on the right.
  - big-endian. The high byte is on the left.
  - native-endian. As defined by the native format of the machine. This is the default.
- **Data Format**. Specifies the data representation format of a field. Applies to fields of all data types except string, ustring, and raw and to record, subrec or tagged fields containing at least one field that is neither string nor raw. Choose from:
  - binary
  - text (the default)

    A setting of binary has different meanings when applied to different data types:
  - For decimals, binary means packed.
  - For other numerical data types, binary means "not text".
  - For dates, binary is equivalent to specifying the julian property for the date field.
  - For time, binary is equivalent to 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. A binary timestamp specifies that two 32-but integers are written.

    By default data is formatted as text, as follows:
  - For the date data type, text specifies that the data to be written contains a text-based date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
  - For the decimal data type: a 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. The destination string format is: [+ | -]ddd.[ddd] and any precision and scale arguments are ignored.
  - For *numeric* fields (int8, int16, int32, uint8, uint16, uint32, sfloat, and dfloat): InfoSphere DataStage assumes that numeric fields are represented as text.
  - For the time data type: text specifies that the field represents time in the text-based form %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
  - For the *timestamp* data type: text specifies a text-based timestamp in the form *%yyyy-%mm-%dd %hh:%nn:%ss* or in the default date format if you have defined a new one on an NLS system.
- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.

- **Field width**. The number of bytes in a field represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.

  If you specify neither field width nor field max width, numeric fields written 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)

  Important: If you are using Unicode character columns, you must calculate the field length in bytes and specify that value in the Field Width column property.
- **Pad char**. Specifies the pad character used when strings or numeric values are written to an external string representation. Enter a character (single-byte for strings, can be multi-byte for ustrings) or choose null or space. The pad character is used when the external string representation is larger than required to hold the written field. In this case, the external string is filled with the pad character to its full length. Space is the default. Applies to string, ustring, and numeric data types and record, subrec, or tagged types if they contain at least one field of this type.
- **Character set**. Specifies the character set. Choose from ASCII or EBCDIC. The default is ASCII. Applies to all data types except raw and ustring and record, subrec, or tagged containing no fields other than raw or ustring.

String

These properties are applied to columns with a string data type, unless overridden at column level.
- **Export EBCDIC as ASCII**. Select this to specify that EBCDIC characters are written as ASCII characters. Applies to fields of the string data type and record, subrec, or tagged fields if they contain at least one field of this type.
- **Import ASCII as EBCDIC**. Not relevant for input links.

Decimal

These properties are applied to columns with a decimal data type unless overridden at column level.
- **Allow all zeros**. Specifies whether to treat a packed decimal column containing all zeros (which is normally illegal) as a valid representation of zero. Select **Yes** or **No**. The default is **No**.
- **Decimal separator.** Specify the ASCII character that acts as the decimal separator (period by default).
- **Packed**. Select an option to specify what the decimal columns contain, choose from:
  - **Yes** to specify that the decimal columns contain data in packed decimal format (the default). This has the following sub-properties:

    Check. Select **Yes** to verify that data is packed, or **No** to not verify.

    Signed. Select **Yes** to use the existing sign when writing decimal columns. Select **No** to write a positive sign (0xf) regardless of the columns' actual sign value.
  - **No (separate)** to specify that they contain unpacked decimal with a separate sign byte. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - **No (zoned)** to specify that they contain an unpacked decimal in either ASCII or EBCDIC text. This has the following sub-property:

Sign Position. Choose leading or trailing as appropriate.

– **No (overpunch)** to specify that the field has a leading or end 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 has the following sub-property:

Sign Position. Choose leading or trailing as appropriate.

- **Precision**. Specifies the precision where a decimal column is written in text format. Enter a number. When a decimal is written to a string representation, InfoSphere DataStage 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. When they are defined, InfoSphere DataStage truncates or pads the source decimal to fit the size of the destination string. If you have also specified the field width property, InfoSphere DataStage truncates or pads the source decimal to fit the size specified by field width.
- **Rounding**. Specifies how to round a decimal column when writing it. Choose from:
  – up (ceiling). Truncate source column towards positive infinity. This mode corresponds to the IEEE 754 Round Up mode. For example, 1.4 becomes 2, -1.6 becomes -1.
  – down (floor). Truncate source column towards negative infinity. This mode corresponds to the IEEE 754 Round Down mode. For example, 1.6 becomes 1, -1.4 becomes -2.
  – nearest value. Round the source column towards the nearest representable value. This mode corresponds to the COBOL ROUNDED mode. For example, 1.4 becomes 1, 1.5 becomes 2, -1.4 becomes -1, -1.5 becomes -2.
  – truncate towards zero. This is the default. Discard fractional digits to the right of the right-most fractional digit supported by 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. Using this method 1.6 becomes 1, -1.6 becomes -1.
- **Scale**. Specifies how to round a source decimal when its precision and scale are greater than those of the destination. By default, when the InfoSphere DataStage writes 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, InfoSphere DataStage truncates or pads the source decimal to fit the size of the destination string. If you have also specified the field width property, InfoSphere DataStage truncates or pads the source decimal to fit the size specified by field width.

Numeric

These properties apply to integer and float fields unless overridden at column level.

- **C_format**. Perform non-default conversion of data from integer or floating-point data to a string. This property specifies a C-language format string used for writing integer or floating point strings. This is passed to *sprintf*(). For example, specifying a C-format of %x and a field width of 8 ensures that integers are written as 8-byte hexadecimal strings.
- **In_format**. This property is not relevant for input links..
- **Out_format**. Format string used for conversion of data from integer or floating-point data to a string. This is passed to *sprintf*(). By default, InfoSphere 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 to pass formatting arguments to *sprintf*().

Date

These properties are applied to columns with a date data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Days since**. Dates are written as a signed integer containing the number of days since the specified date. Enter a date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
- **Format string**. The string format of a date. By default this is %yyyy-%mm-%dd. For details about the format, see "Date formats" on page 31.
- **Is Julian**. Select this to specify that dates are written as a numeric value containing the Julian day. A Julian day specifies the date as the number of days from 4713 BCE January 1, 12:00 hours (noon) GMT.

Time

These properties are applied to columns with a time data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Format string**. Specifies the format of columns representing time as a string. For details about the format, see "Time formats" on page 35
- **Is midnight seconds**. Select this to specify that times are written as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.

Timestamp

These properties are applied to columns with a timestamp data type unless overridden at column level.

- **Format string**. Specifies the format of a column representing a timestamp as a string. Defaults to *%yyyy-%mm-%dd %hh:%nn:%ss*. The format combines the format for date strings and time strings. See "Date formats" on page 31 and "Time formats" on page 35.

Sequential File stage: Output page

The **Output page** allows you to specify details about how the Sequential File stage reads data from one or more flat files. The Sequential File stage can have only one output link, but this can read from multiple files.

It can also have a single reject link. This is typically used when you are writing to a file and provides a location where records that have failed to be written to a file for some reason can be sent. When you are reading files, you can use a reject link as a destination for rows that do not match the expected column definitions.

The **Output name** drop-down list allows you to choose whether you are looking at details of the main output link (the stream link) or the reject link.

The **General** tab allows you to specify an optional description of the output link. The **Properties** tab allows you to specify details of exactly what the link does. The **Formats** tab gives information about the format of the files being read. The **Columns** tab specifies the column definitions of the data. The **Advanced** tab allows you to change the default buffering settings for the output link.

Details about Sequential File stage properties and formatting are given in the following sections. See Chapter 4, "Stage editors," on page 49, "Stage Editors," for a general description of the other tabs.

Sequential File stage: Output link properties tab

The Properties tab allows you to specify properties for the output link. These dictate how incoming data is read from what files. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Source/File	pathname	N/A	Y if Read Method = Specific Files(s)	Y	N/A
Source/File Pattern	pathname	N/A	Y if Read Method = Field Pattern	N	N/A
Source/Read Method	Specific File(s)/File Pattern	Specific Files(s)	Y	N	N/A
Options/"First Line is Column Names" on page 102	True/False	False	Y	N	N/A
Options/Missing File Mode	Error/OK/ Depends	Depends	Y if File used	N	N/A
Options/Keep file Partitions	True/False	False	Y	N	N/A
Options/Reject Mode	Continue/ Fail/Save	Continue	Y	N	N/A
Options/Report Progress	Yes/No	Yes	Y	N	N/A
Options/Filter	command	N/A	N	N	N/A
Options/File Name Column	column name	N/A	N	N	N/A
Options/"Read first rows" on page 102	number	N/A	N	N	N/A
Options/"Row number column" on page 102	column name	N/A	N	N	N/A
Options/Number Of Readers Per Node	number	1	N	N	N/A
Options/Schema File	pathname	N/A	N	N	N/A
Options/"Strip BOM" on page 104	True/False	FALSE	N	N	N/A

Sequential File stage: Source category: File

This property defines the flat file that data will be read from. You can type in a pathname, or browse for a file. You can specify multiple files by repeating the File property. Do this by selecting the **Properties** item at the top of the tree, and clicking on **File** in the Available properties to add window. Do this for each extra file you want to specify.

File pattern

Specifies a group of files to import. Specify file containing a list of files or a job parameter representing the file. The file could also contain be any valid shell expression, in Bourne shell syntax, that generates a list of file names.

Read method

This property specifies whether you are reading from a specific file or files or using a file pattern to select files (for example, *.txt).

Sequential File stage: Options category:
First Line is Column Names

Specifies that the first line of the file contains column names. This property is false by default.

Missing file mode

Specifies the action to take if one of your File properties has specified a file that does not exist. Choose from **Error** to stop the job, **OK** to skip the file, or **Depends**, which means the default is **Error**, unless the file has a node name prefix of *: in which case it is **OK**. The default is **Depends**.

Keep file partitions

Set this to **True** to partition the imported data set according to the organization of the input file(s). So, for example, if you are reading three files you will have three partitions. Defaults to **False**.

Reject mode

Allows you to specify behavior if a read record does not match the expected schema. Choose from **Continue** to continue operation and discard any rejected rows, **Fail** to cease reading if any rows are rejected, or **Save** to send rejected rows down a reject link. Defaults to **Continue**.

Report progress

Choose **Yes** or **No** to enable or disable reporting. By default the stage displays a progress report at each 10% interval when it can ascertain file size. Reporting occurs only if the file is greater than 100 KB, records are fixed length, and there is no filter on the file.

Filter

This is an optional property. You can use this to specify that the data is passed through a filter program after being read from the files. Specify the filter command, and any required arguments, in the Property Value box.

File name column

This is an optional property. It adds an extra column of type VarChar to the output of the stage, containing the pathname of the file the record is read from. You should also add this column manually to the Columns definitions to ensure that the column is not dropped if you are not using runtime column propagation, or it is turned off at some point.

Read first rows

Specify a number *n* so that the stage only reads the first *n* rows from the file.

Row number column

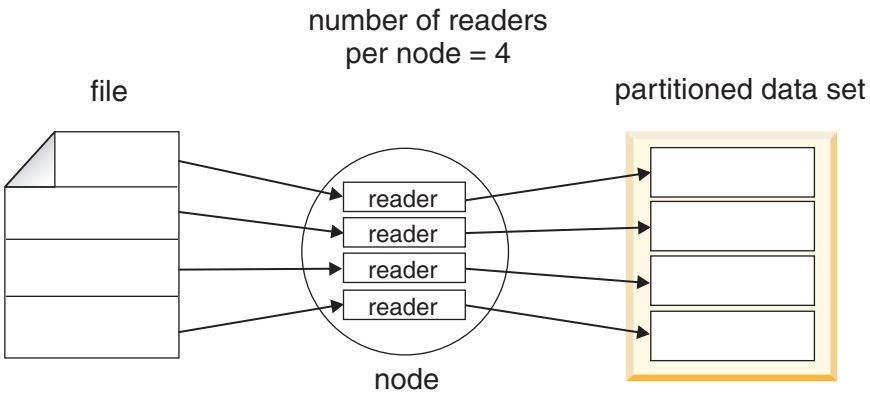
This is an optional property. It adds an extra column of type unsigned BigInt to the output of the stage, containing the row number. You must also add the column to the columns tab, unless runtime column propagation is enabled.

Number Of readers per node

This is an optional property and only applies to files containing fixed-length records, it is mutually exclusive with the Read from multiple nodes property. Specifies the number of instances of the file read operator on a processing node. The default is one operator per node per input data file. If *numReaders* is greater than one, each instance of the file read operator reads a contiguous range of records from the input file. The starting record location in the file for each operator, or *seek* location, is determined by the data file size, the record length, and the number of instances of the operator, as specified by *numReader*s.

The resulting data set contains one partition per instance of the file read operator, as determined by *numReader*s.

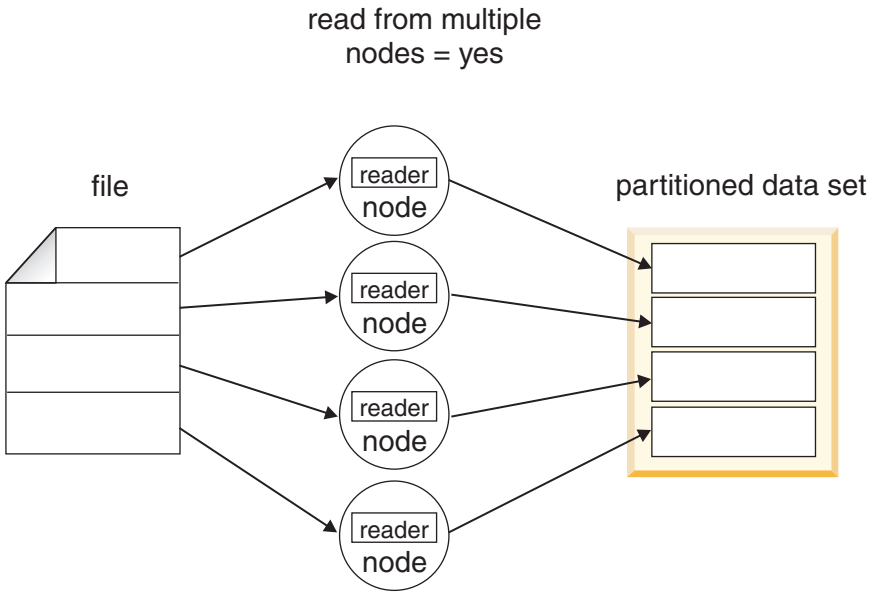
This provides a way of partitioning the data contained in a single file. Each node reads a single file, but the file can be divided according to the number of readers per node, and written to separate partitions. This method can result in better I/O performance on an SMP system.



Read from multiple nodes

This is an optional property and only applies to files containing fixed-length records, it is mutually exclusive with the Number of Readers Per Node property. Set this to Yes to allow individual files to be read by several nodes. This can improve performance on a cluster system.

InfoSphere DataStage knows the number of nodes available, and using the fixed length record size, and the actual size of the file to be read, allocates the reader on each node a separate region within the file to process. The regions will be of roughly equal size.



Schema file

This is an optional property. By default the Sequential File stage will use the column definitions defined on the **Columns** and **Format** tabs as a schema for reading the file. You can, however, specify a file containing a schema instead (note, however, that if you have defined columns on the **Columns** tab, you should ensure these match the schema file). Type in a pathname or browse for a schema file.

Strip BOM

Set this property TRUE to drop the UTF-16 Endianess byte order mark when reading data. By default, this property is set to FALSE.

Sequential File stage: Reject Links:

You cannot change the properties of a Reject link. The **Properties** tab for a reject link is blank.

Similarly, you cannot edit the column definitions for a reject link. For writing files, the link uses the column definitions for the input link. For reading files, the link uses a single column called rejected containing raw data for columns rejected after reading because they do not match the schema.

Sequential File stage: Output link format tab:

The Format tab allows you to supply information about the format of the flat file or files that you are reading. The tab has a similar format to the Properties tab.

If you do not alter any of the Format settings, the Sequential File stage will produce a file of the following format:

- File comprises variable length columns contained within double quotes.
- All columns are delimited by a comma, except for the final column in a row.
- Rows are delimited by a UNIX newline.

You can use the **Format As** item from the shortcut menu in the Format tab to quickly change to a fixed-width column format, using DOS newlines as row delimiters, or producing a COBOL format file.

You can use the **Defaults** button to change your default settings. Use the Format tab to specify your required settings, then click **Defaults** > **Save current as default**. All your sequential files will use your settings by default from now on. If your requirements change, you can choose **Defaults** > **Reset defaults from factory settings** to go back to the original defaults as described above. Once you have done this, you then have to click **Defaults** > **Set current from default** for the new defaults to take effect.

To change individual properties, select a property type from the main tree then add the properties you want to set to the tree structure by clicking on them in the **Available properties to addwindow**. You can then set a value for that property in the Property Value box. Pop-up help for each of the available properties appears if you hover the mouse pointer over it.

Any property that you set on this tab can be overridden at the column level by setting properties for individual columns on the Edit Column Metadata dialog box (see Columns Tab).

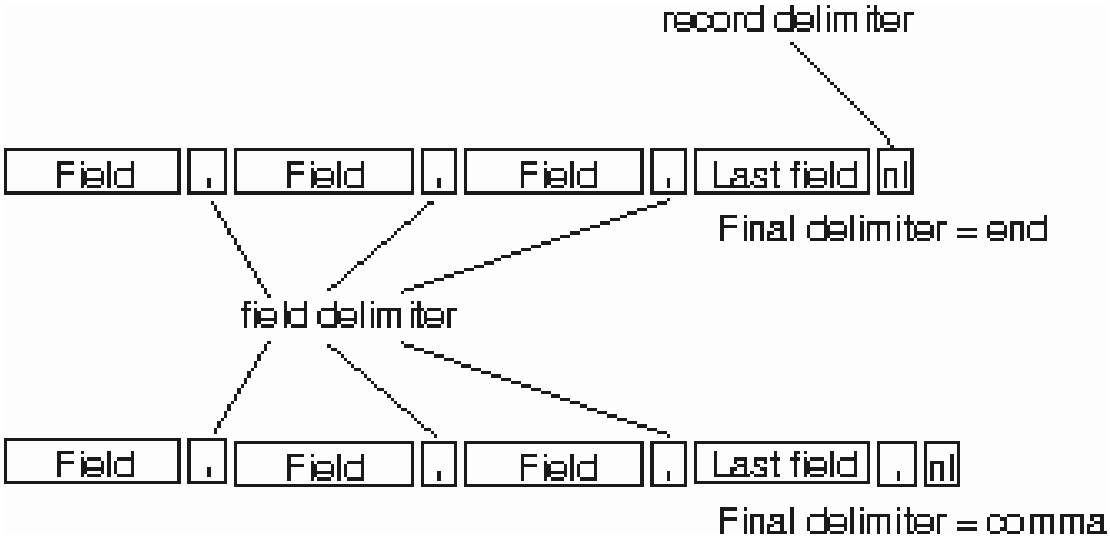
This description uses the terms "record" and "row" and "field" and "column" interchangeably.

The following sections list the property types and properties available for each type.

Record level

These properties define details about how data records are formatted in the flat file. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Fill char.** Does not apply to output links.
- **Final delimiter string**. Specify the string written after the last column of a record in place of the column delimiter. Enter one or more characters, this precedes the record delimiter if one is used. Mutually exclusive with Final delimiter, which is the default. For example, if you set Delimiter to comma and Final delimiter string to `, ` (comma space - you do not need to enter the inverted commas) all fields are delimited by a comma, except the final field, which is delimited by a comma followed by an ASCII space character. InfoSphere DataStage skips the specified delimiter string when reading the file.
- **Final delimiter**. Specify the single character written after the last column of a record in place of the field delimiter. Type a character or select one of whitespace, end, none, null, tab, or comma. InfoSphere DataStage skips the specified delimiter string when reading the file. See the following diagram for an illustration.
  - **whitespace**. The last column of each record will not include any trailing white spaces found at the end of the record.
  - **end**. The last column of each record does not include the field delimiter. This is the default setting.
  - **none**. The last column of each record does not have a delimiter, used for fixed-width fields.
  - **null**. The last column of each record is delimited by the ASCII null character.
  - **comma**. The last column of each record is delimited by the ASCII comma character.
  - **tab**. The last column of each record is delimited by the ASCII tab character.



- **Intact**. The intact property specifies an identifier of a partial schema. A partial schema specifies that only the column(s) named in the schema can be modified by the stage. All other columns in the row are passed through unmodified. The file containing the partial schema is specified in the Schema File property on the **Outputs** tab. This property has a dependent property:
  - **Check intact**. Select this to force validation of the partial schema as the file or files are imported. Note that this can degrade performance.
- **Record delimiter string**. Specify the string at the end of each record. Enter one or more characters. This is mutually exclusive with Record delimiter, which is the default, and record type and record prefix.
- **Record delimiter**. Specify the single character at the end of each record. Type a character or select one of the following:
  - UNIX Newline (the default)
  - null

    (To specify a DOS newline, use the Record delimiter string property set to "\R\N" or choose **Format as** > **DOS line terminator** from the shortcut menu.)

    Record delimiter is mutually exclusive with Record delimiter string, Record prefix, and record type.
- **Record length**. Select Fixed where fixed length fields are being read. InfoSphere DataStage calculates the appropriate length for the record. Alternatively specify the length of fixed records as number of bytes. This is not used by default (default files are comma-delimited).
- **Record Prefix**. Specifies that a variable-length record is prefixed by a 1-, 2-, or 4-byte length prefix. It is set to 1 by default. This is mutually exclusive with Record delimiter, which is the default, and record delimiter string and record type.
- **Record type**. Specifies that data consists of variable-length blocked records (varying) or implicit records (implicit). If you choose the implicit property, data is written as a stream with no explicit record boundaries. The end of the record is inferred when all of the columns defined by the schema have been parsed. The varying property allows you to specify one of the following IBM blocked or spanned formats: V, VB, VS, VBS, or VR.

  This property is mutually exclusive with Record length, Record delimiter, Record delimiter string, and Record prefix and by default is not used.

Field Defaults

Defines default properties for columns read from the file or files. These are applied to all columns, but can be overridden for individual columns from the Columns tab using the Edit Column Metadata dialog box. A common reason to override a property for an individual column occurs when reading

comma-separated values (CSV) files. CSV files often enclose fields in quotes when the fields might contain a special character, such as the field delimiter. In this case, the **Quote** property for the columns in question should be overridden.

Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Actual field length**. Specifies the actual number of bytes to skip if the field's length equals the setting of the null field length property.
- **Delimiter**. Specifies the trailing delimiter of all fields in the record. Type an ASCII character or select one of whitespace, end, none, null, comma, or tab. InfoSphere DataStage skips the delimiter when reading.
  - **whitespace**. Whitespace characters at the end of a column are ignored, that is, are not treated as part of the column.
  - **end**. The end of a field is taken as the delimiter, that is, there is no separate delimiter. This is not the same as a setting of `None' which is used for fields with fixed-width columns.
  - **none**. No delimiter (used for fixed-width).
  - **null**. ASCII Null character is used.
  - **comma**. ASCII comma character is used.
  - **tab**. ASCII tab character is used.
- **Delimiter string**. Specify the string at the end of each field. Enter one or more characters. This is mutually exclusive with Delimiter, which is the default. For example, specifying `, ` (comma space - you do not need to enter the inverted commas) specifies each field is delimited by `, ` unless overridden for individual fields. InfoSphere DataStage skips the delimiter string when reading.
- **Null field length**. The length in bytes of a variable-length field that contains a null. When a variable-length field is read, a length of null field length in the source field indicates that it contains a null. This property is mutually exclusive with null field value.
- **Null field value**. Specifies the value given to a null field if the source is set to null. Can be a number, string, or C-type 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 \x*hh*, where each *h* is a hexadecimal digit 0 - F. You must use this form to encode non-printable byte values.

  This property is mutually exclusive with Null field length and Actual length. For a fixed width data representation, you can use Pad char (from the general section of Type defaults) to specify a repeated trailing character if the value you specify is shorter than the fixed width of the field.

  You can specify a list of null values that a column could contain that represent null. To do this you specify a separator character in the dependent **Null field value separator** property, and then use this separator to delimit the null values in the **Null field value** property. For example, if you set **Null field value separator** to contain the slash character (/), then you could specify NULL/null/NUL/nul to specify that any of these strings could represent a null value in this column.
- **Null field value separator**

  This is a dependent property of **Null field value**. You can specify a separator that can be used in the **Null field value** property to specify a range of values that could represent the null value. You can specify a number, string, or C-type literal escape character (as for **Null field value**) as a separator, but a single character such as a comma (,) or slash (/) character is the best choice. You must only specify a separator if you specify multiple values in **Null field value**; specifying a separator without using it will cause a runtime error.
- **Prefix bytes**. 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. InfoSphere DataStage reads the length prefix but does not include the prefix as a separate field in the data set it reads from the file.

  This property is mutually exclusive with the Delimiter, Quote, and Final Delimiter properties, which are used by default.

- **Print field**. This property is intended for use when debugging jobs. Set it to have InfoSphere DataStage produce a message for every field it reads. The message has the format:

  `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.
- **Quote**. Specifies that variable length fields are enclosed in single quotes, double quotes, or another character or pair of characters. Choose Single or Double, or enter a character. This is set to double quotes by default.

  When reading, InfoSphere DataStage ignores the leading quote character and reads all bytes up to but not including the trailing quote character.
- **Vector prefix**. For fields that are variable length vectors, specifies that a 1-, 2-, or 4-byte prefix contains the number of elements in the vector. You can override this default prefix for individual vectors.

  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 this property to indicate the prefix length. InfoSphere DataStage reads the length prefix but does not include it as a separate field in the data set. By default, the prefix length is assumed to be one byte.

Type Defaults

These are properties that apply to all columns of a specific data type unless specifically overridden at the column level. They are divided into a number of subgroups according to data type.

General

These properties apply to several data types (unless overridden at column level):

- **Byte order**. Specifies how multiple byte data types (except string and raw data types) are ordered. Choose from:
  – little-endian. The high byte is on the right.
  – big-endian. The high byte is on the left.
  – native-endian. As defined by the native format of the machine. This is the default.
- **Data Format**. Specifies the data representation format of a field. Applies to fields of all data types except string, ustring, and raw and to record, subrec or tagged fields containing at least one field that is neither string nor raw. Choose from:
  – binary
  – text (the default)

  A setting of binary has different meanings when applied to different data types:
  – For decimals, binary means packed.
  – For other numerical data types, binary means "not text".
  – For dates, binary is equivalent to specifying the julian property for the date field.
  – For time, binary is equivalent to 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. A binary timestamp specifies that two 32-but integers are written.

  By default data is formatted as text, as follows:
  – For the date data type, text specifies that the data read, contains a text-based date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system.

  - For the decimal data type: a 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. The destination string format is: [+ | -]ddd.[ddd] and any precision and scale arguments are ignored.
  - For numeric fields (int8, int16, int32, uint8, uint16, uint32, sfloat, and dfloat): InfoSphere DataStage assumes that numeric fields are represented as text.
  - For the time data type: text specifies that the field represents time in the text-based form %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system.
  - For the timestamp data type: text specifies a text-based timestamp in the form %yyyy-%mm-%dd %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.
- **Field width**. The number of bytes in a field represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.

  If you specify neither field width nor field max width, numeric fields written 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)
- **Pad char**. This property is ignored for output links.
- **Character set**. Specifies the character set. Choose from ASCII or EBCDIC. The default is ASCII. Applies to all data types except raw and ustring and record, subrec, or tagged containing no fields other than raw or ustring.

String

These properties are applied to columns with a string data type, unless overridden at column level.

- **Export EBCDIC as ASCII**. Not relevant for output links.
- **Import ASCII as EBCDIC**. Select this to specify that ASCII characters are read as EBCDIC characters.

Decimal

These properties are applied to columns with a decimal data type unless overridden at column level.

- **Allow all zeros**. Specifies whether to treat a packed decimal column containing all zeros (which is normally illegal) as a valid representation of zero. Select Yes or No. The default is No.
- **Decimal separator**. Specify the ASCII character that acts as the decimal separator (period by default).
- **Packed**. Select an option to specify what the decimal columns contain, choose from:
  - Yes to specify that the decimal fields contain data in packed decimal format (the default). This has the following sub-properties:

    Check. Select Yes to verify that data is packed, or No to not verify.

Signed. Select Yes to use the existing sign when reading decimal fields. Select No to write a positive sign (0xf) regardless of the fields' actual sign value.

- No (separate) to specify that they contain unpacked decimal with a separate sign byte. This has the following sub-property:

  Sign Position. Choose leading or trailing as appropriate.
- No (zoned) to specify that they contain an unpacked decimal in either ASCII or EBCDIC text. This has the following sub-property:

  Sign Position. Choose leading or trailing as appropriate.
- No (overpunch) to specify that the field has a leading or end 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 has the following sub-property:

  Sign Position. Choose leading or trailing as appropriate.

- **Precision**. Specifies the precision of a packed decimal. Enter a number.
- **Rounding**. Specifies how to round the source field to fit into the destination decimal when reading a source field to a decimal. Choose from:
  - **up (ceiling)**. Truncate source column towards positive infinity. This mode corresponds to the IEEE 754 Round Up mode. For example, 1.4 becomes 2, -1.6 becomes -1.
  - **down (floor)**. Truncate source column towards negative infinity. This mode corresponds to the IEEE 754 Round Down mode. For example, 1.6 becomes 1, -1.4 becomes -2.
  - **nearest value**. Round the source column towards the nearest representable value. This mode corresponds to the COBOL ROUNDED mode. For example, 1.4 becomes 1, 1.5 becomes 2, -1.4 becomes -1, -1.5 becomes -2.
  - **truncate towards zero**. This is the default. Discard fractional digits to the right of the right-most fractional digit supported by 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. Using this method 1.6 becomes 1, -1.6 becomes -1.
- **Scale**. Specifies the scale of a source packed decimal.

Numeric

These properties apply to integer and float fields unless overridden at column level.

- **C_format**. Perform non-default conversion of data from string data to a integer or floating-point. This property specifies a C-language format string used for reading integer or floating point strings. This is passed to *sscanf*(). For example, specifying a C-format of %x and a field width of 8 ensures that a 32-bit integer is formatted as an 8-byte hexadecimal string.
- **In_format**. Format string used for conversion of data from string to integer or floating-point data This is passed to *sscanf*(). By default, InfoSphere DataStage invokes the C *sscanf*() function to convert a numeric field formatted as a string to either integer or floating point data. If this function does not output data in a satisfactory format, you can specify the in_format property to pass formatting arguments to *sscanf*().
- **Out_format**. This property is not relevant for output links.

Date

These properties are applied to columns with a date data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Days since**. Dates are written as a signed integer containing the number of days since the specified date. Enter a date in the form *%yyyy-%mm-%d*d or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).

- **Format string**. The string format of a date. By default this is %yyyy-%mm-%dd. For details about the format, see "Date formats" on page 31.
- **Is Julian**. Select this to specify that dates are written as a numeric value containing the Julian day. A Julian day specifies the date as the number of days from 4713 BCE January 1, 12:00 hours (noon) GMT.

Time

These properties are applied to columns with a time data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Format string**. Specifies the format of columns representing time as a string. By default this is %hh-%mm-%ss. For details about the format, see "Time formats" on page 35
- **Is midnight seconds**. Select this to specify that times are written as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.

Timestamp

These properties are applied to columns with a timestamp data type unless overridden at column level.

- **Format string**. Specifies the format of a column representing a timestamp as a string. The format combines the format for date strings and time strings. See "Date formats" on page 31 and "Time formats" on page 35.

Using RCP with Sequential File stages:

Runtime column propagation (RCP) allows InfoSphere DataStage to be flexible about the columns you define in a job. If RCP is enabled for a project, you can just define the columns you are interested in using in a job, but ask InfoSphere DataStage to propagate the other columns through the various stages. So such columns can be extracted from the data source and end up on your data target without explicitly being operated on in between.

Sequential files, unlike most other data sources, do not have inherent column definitions, and so InfoSphere DataStage cannot always tell where there are extra columns that need propagating. You can only use RCP on sequential files if you have used the Schema File property to specify a schema which describes all the columns in the sequential file. You need to specify the same schema file for any similar stages in the job where you want to propagate columns. Stages that will require a schema file are:

- Sequential File
- File Set
- External Source
- External Target
- Column Import
- Column Export

File set stage

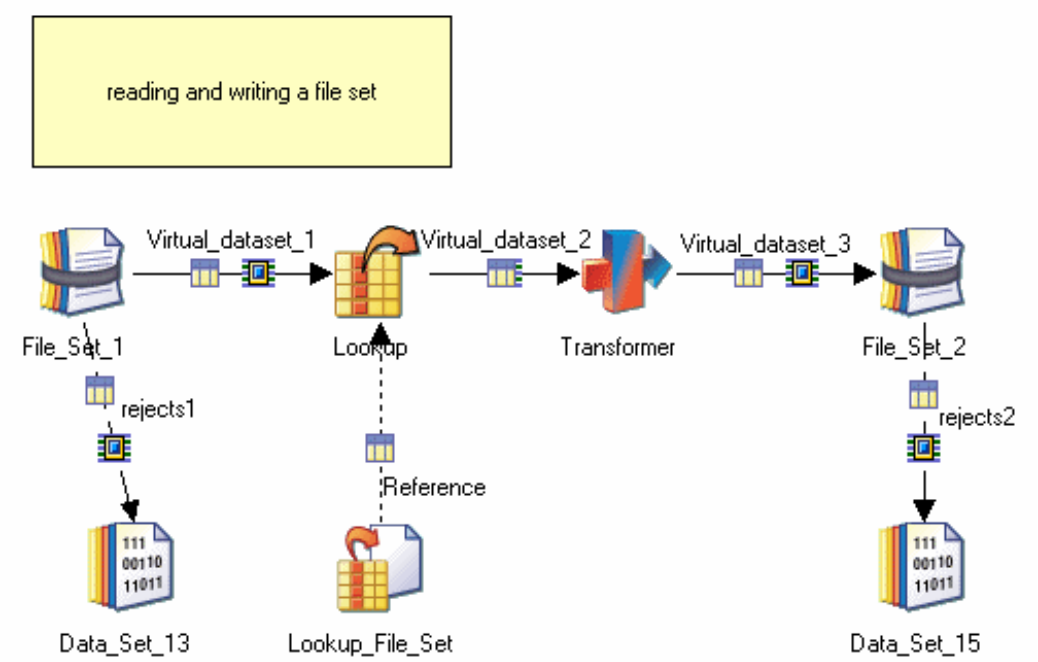
The File Set stage is a file stage. It allows you to read data from or write data to a file set. The stage can have a single input link, a single output link, and a single rejects link. It only executes in parallel mode.

What is a file set? InfoSphere DataStage can generate and name exported files, write them to their destination, and list the files it has generated in a file whose extension is, by convention, .fs. The data files and the file that lists them are called a *file set*. This capability is useful because some operating systems impose a 2 GB limit on the size of a file and you need to distribute files among nodes to prevent overruns.

The amount of data that can be stored in each destination data file is limited 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

The File Set stage enables you to create and write to file sets, and to read data back from file sets.



Unlike data sets, file sets carry formatting information that describe the format of the files to be read or written.

When you edit a File Set stage, the File Set stage editor appears. This is based on the generic stage editor described in"Stage Editors."

The stage editor has up to three pages, depending on whether you are reading or writing a file set:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is present when you are writing to a file set. This is where you specify details about the file set being written to.
- **Output Page**. This is present when you are reading from a file set. This is where you specify details about the file set being read from.

There are one or two special points to note about using runtime column propagation (RCP) with File Set stages. See "Using RCP With File Set Stages" for details.

File Set stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include File Set stages in a job. This section specifies the minimum steps to take to get a File Set stage functioning. InfoSphere

DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic methods, you will learn where the shortcuts are when you get familiar with the product.

The steps required depend on whether you are using the File Set stage to read or write a file.

Writing to a file
About this task

- In the **Input Link Properties Tab** specify the pathname of the file set being written to. The other properties all have default values, which you can change or not as required.
- In the **Input Link Format Tab** specify format details for the file set you are writing to, or accept the defaults (variable length columns enclosed in double quotes and delimited by commas, rows delimited with UNIX newlines).
- Ensure column meta data has been specified for the file set.

Reading from a file
About this task

- In the **Output Link Properties Tab** specify the pathname of the file set being read from. The other properties all have default values, which you can change or not as required.
- In the **Output Link Format Tab** specify format details for the file set you are reading from, or accept the defaults (variable length columns enclosed in double quotes and delimited by commas, rows delimited with UNIX newlines).
- Ensure column meta data has been specified for the file set (this can be achieved via a schema file if required).

File Set stage: Stage page

The General tab allows you to specify an optional description of the stage. The Advanced tab allows you to specify how the stage executes. The NLS Map tab appears if you have NLS enabled on your system, it allows you to specify a character set map for the stage.

File Set stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. This is set to parallel and cannot be changed.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. You can select **Set** or **Clear**. If you select **Set**, file set read operations will request that the next stage preserves the partitioning as is (it is ignored for file set write operations).
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

File Set stage: NLS Map tab

The NLS Map tab allows you to define a character set map for the File Set stage. This overrides the default character set map set for the project or the job. You can specify that the map be supplied as a job parameter if required. You can also select **Allow per-column mapping**. This allows character set maps to

be specified for individual columns within the data processed by the File Set stage An extra property, **NLS Map**, appears in the Columns grid in the Columns tab, but note that only ustring data types allow you to set an NLS map value (see "Data Types").

File Set stage: Input page

The Input page allows you to specify details about how the File Set stage writes data to a file set. The File Set stage can have only one input link.

The General tab allows you to specify an optional description of the input link. The Properties tab allows you to specify details of exactly what the link does. The Partitioning tab allows you to specify how incoming data is partitioned before being written to the file set. The Formats tab gives information about the format of the files being written. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about File Set stage properties, partitioning, and formatting are given in the following sections. See "Stage Editors," for a general description of the other tabs.

File Set stage: Input link properties tab

The Properties tab allows you to specify properties for the input link. These dictate how incoming data is written and to what file set. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 15. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Target/File Set	pathname	N/A	Y	N	N/A
Target/File Set Update Policy	Create (Error if exists) /Overwrite/Use Existing (Discard records)/ Use Existing (Discard schema & records)	Overwrite	Y	N	N/A
Target/File Set Schema policy	Write/Omit	Write	Y	N	N/A
Options/Cleanup on Failure	True/False	True	Y	N	N/A
Options/Single File Per Partition.	True/False	False	Y	N	N/A
Options/Reject Mode	Continue/Fail/ Save	Continue	Y	N	N/A
Options/Diskpool	string	N/A	N	N	N/A
Options/File Prefix	string	export.*username*	N	N	N/A
Options/File Suffix	string	none	N	N	N/A

Table 15. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Maximum File Size	number MB	N/A	N	N	N/A
Options/Schema File	pathname	N/A	N	N	N/A

File Set stage: Target category:
File set

This property defines the file set that the incoming data will be written to. You can type in a pathname of, or browse for a file set descriptor file (by convention ending in `.fs`).

File set update policy

Specifies what action will be taken if the file set you are writing to already exists. Choose from:

- Create (Error if exists)
- Overwrite
- Use Existing (Discard records). Keeps existing files listed in a descriptor file (for example, `datasetname.ds` or `filesetname.fs`) but discards the old records. You receive an error if the data set with different schema already exists.
- Use Existing (Discard schema & records). Keeps existing files listed in a descriptor file (for example, `datasetname.ds` or `filesetname.fs`) but discards the old schema and records.

The default is Overwrite.

File set schema policy

Specifies whether the schema should be written to the file set. Choose from **Write** or **Omit**. The default is **Write**.

File Set stage: Options category:
Cleanup on failure

This is set to **True** by default and specifies that the stage will delete any partially written files if the stage fails for any reason. Set this to **False** to specify that partially written files should be left.

Single file per partition

Set this to **True** to specify that one file is written for each partition. The default is **False**.

Reject mode

Allows you to specify behavior if a record fails to be written for some reason. Choose from **Continue** to continue operation and discard any rejected rows, **Fail** to cease reading if any rows are rejected, or **Save** to send rejected rows down a reject link. Defaults to **Continue**.

Diskpool

This is an optional property. Specify the name of the disk pool into which to write the file set. You can also specify a job parameter.

File prefix

This is an optional property. Specify a prefix for the name of the file set components. If you do not specify a prefix, the system writes the following: export.*username*, where *username* is your login. You can also specify a job parameter.

File suffix

This is an optional property. Specify a suffix for the name of the file set components. The suffix is omitted by default.

Maximum file size

This is an optional property. Specify the maximum file size in MB. The value must be equal to or greater than 1.

Schema file

This is an optional property. By default the File Set stage will use the column definitions defined on the Columns tab and formatting information from the Format tab as a schema for writing the file. You can, however, specify a file containing a schema instead (note, however, that if you have defined columns on the Columns tab, you should ensure these match the schema file). Type in a pathname or browse for a schema file.

File Set stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is written to the file set. It also allows you to specify that the data should be sorted before being written.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the File Set stage is operating in sequential mode, it will first collect the data before writing it to the file using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the File Set stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the File Set stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the File Set stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the File Set stage.
- **Entire**. Each file written to receives the entire data set.

- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default method for the File Set stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

  The **Partitioning** tab also allows you to specify that data arriving on the input link should be sorted before being written to the file or files. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

File Set stage: Input link format tab

The Format tab allows you to supply information about the format of the files in the file set to which you are writing. The tab has a similar format to the Properties tab and is described on page Format Tab.

If you do not alter any of the Format settings, the File Set stage will produce files of the following format:

- Files comprise variable length columns contained within double quotes.
- All columns are delimited by a comma, except for the final column in a row.
- Rows are delimited by a UNIX newline.

You can use the **Format As** item from the shortcut menu in the Format tab to quickly change to a fixed-width column format, using DOS newlines as row delimiters, or producing a COBOL format file.

To change individual properties, select a property type from the main tree then add the properties you want to set to the tree structure by clicking on them in the **Available properties to set window**. You can then set a value for that property in the Property Value box. Pop-up help for each of the available properties appears if you hover the mouse pointer over it.

Any property that you set on this tab can be overridden at the column level by setting properties for individual columns on the Edit Column Metadata dialog box (see page Columns Tab).

This description uses the terms "record" and "row" and "field" and "column" interchangeably.

The following sections list the Property types and properties available for each type.

Record level

These properties define details about how data records are formatted in the flat file. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

Fill char

Specify an ASCII character or a value in the range 0 to 255. You can also choose Space or Null from a drop-down list. This character is used to fill any gaps in a written record caused by column positioning properties. Set to 0 by default (which is the NULL character). For example, to set it to space you could also type in the space character or enter 32. Note that this value is restricted to one byte, so you cannot specify a multi-byte Unicode character.

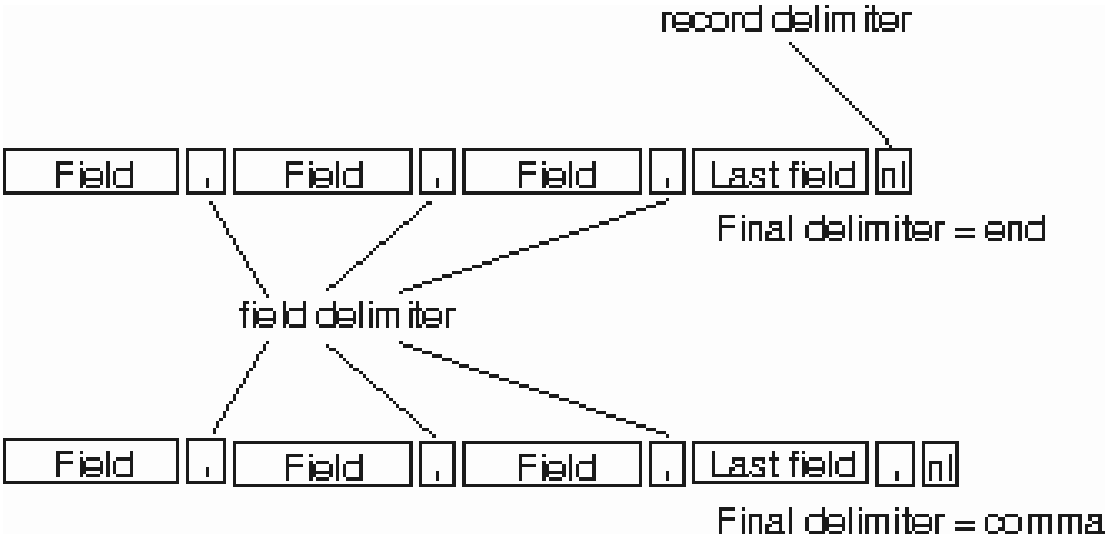
Final delimiter string

Specify a string to be written after the last column of a record in place of the column delimiter. Enter one or more characters, this precedes the record delimiter if one is used. Mutually exclusive with Final delimiter, which is the default. For example, if you set Delimiter to comma and Final delimiter string to `, ` (comma space - you do not need to enter the inverted commas) all fields are delimited by a comma, except the final field, which is delimited by a comma followed by an ASCII space character.

Final delimiter

Specify a single character to be written after the last column of a record in place of the field delimiter. Type a character or select one of whitespace, end, none, null, tab, or comma. See the following diagram for an illustration.

- **whitespace**. The last column of each record will not include any trailing white spaces found at the end of the record.
- **end**. The last column of each record does not include the field delimiter. This is the default setting.
- **none**. The last column of each record does not have a delimiter; used for fixed-width fields.
- **null**. The last column of each record is delimited by the ASCII null character.
- **comma**. The last column of each record is delimited by the ASCII comma character.
- **tab**. The last column of each record is delimited by the ASCII tab character.



When writing, a space is now inserted after every field except the last in the record. Previously, a space was inserted after every field including the last. (If you want to revert to the pre-release 7.5 behavior of inserting a space after the last field, set the APT_FINAL_DELIM_COMPATIBLE environment variable.

Intact

The intact property specifies an identifier of a partial schema. A partial schema specifies that only the column(s) named in the schema can be modified by the stage. All other columns in the row are passed through unmodified. The file containing the partial schema is specified in the **Schema File** property on the **Properties** tab. This property has a dependent property, Check intact, but this is not relevant to input links.

Record delimiter string

Specify a string to be written at the end of each record. Enter one or more characters. This is mutually exclusive with Record delimiter, which is the default, record type and record prefix.

Record delimiter

Specify a single character to be written at the end of each record. Type a character or select one of the following:

- UNIX Newline (the default)
- null

(To implement a DOS newline, use the Record delimiter string property set to "\R\N" or choose**Format as** > **DOS line terminator** from the shortcut menu.)

Note: Record delimiter is mutually exclusive with Record delimiter string, Record prefix, and Record type.

Record length

Select **Fixed** where fixed length fields are being written. InfoSphere DataStage calculates the appropriate length for the record. Alternatively specify the length of fixed records as number of bytes. This is not used by default (default files are comma-delimited). The record is padded to the specified length with either zeros or the fill character if one has been specified.

Record Prefix

Specifies that a variable-length record is prefixed by a 1-, 2-, or 4-byte length prefix. It is set to 1 by default. This is mutually exclusive with Record delimiter, which is the default, and record delimiter string and record type.

Record type

Specifies that data consists of variable-length blocked records (varying) or implicit records (implicit). If you choose the implicit property, data is written as a stream with no explicit record boundaries. The end of the record is inferred when all of the columns defined by the schema have been parsed. The varying property allows you to specify one of the following IBM blocked or spanned formats: **V**, **VB**, **VS**, **VBS**, or **VR**.

This property is mutually exclusive with Record length, Record delimiter, Record delimiter string, and Record prefix and by default is not used.

Field defaults

Defines default properties for columns written to the file or files. These are applied to all columns written, but can be overridden for individual columns from the **Columns** tab using the Edit Column Metadata dialog box. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Actual field length**. Specifies the number of bytes to fill with the Fill character when a field is identified as null. When InfoSphere DataStage identifies a null field, it will write a field of this length full of Fill characters. This is mutually exclusive with Null field value.
- **Delimiter**. Specifies the trailing delimiter of all fields in the record. Type an ASCII character or select one of whitespace, end, none, null, comma, or tab.
  - **whitespace**. Whitespace characters at the end of a column are ignored, that is, are not treated as part of the column.
  - **end**. The end of a field is taken as the delimiter, that is, there is no separate delimiter. This is not the same as a setting of `None' which is used for fields with fixed-width columns.
  - **none**. No delimiter (used for fixed-width).
  - **null**. ASCII Null character is used.
  - **comma**. ASCII comma character is used.
  - **tab**. ASCII tab character is used.
- **Delimiter string**. Specify a string to be written at the end of each field. Enter one or more characters. This is mutually exclusive with Delimiter, which is the default. For example, specifying `, ` (comma space - you do not need to enter the inverted commas) would have each field delimited by `, ` unless overridden for individual fields.
- **Null field length**. The length in bytes of a variable-length field that contains a null. When a variable-length field is written, InfoSphere DataStage writes a length value of null field length if the field contains a null. This property is mutually exclusive with null field value.
- **Null field value**. Specifies the value written to null field if the source is set to null. Can be a number, string, or C-type 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 **\x***hh*, where each *h* is a hexadecimal digit 0 - F. You must use this form to encode non-printable byte values.

  This property is mutually exclusive with Null field length and Actual length. For a fixed width data representation, you can use Pad char (from the general section of Type defaults) to specify a repeated trailing character if the value you specify is shorter than the fixed width of the field.

  Null field value has a sub property named Null field value separator. This is intended for output data, and should be ignored on Format tabs belonging to input links.

- **Prefix bytes**. Specifies that each column in the data file is prefixed by 1, 2, or 4 bytes containing, as a binary value, either the column's length or the tag value for a tagged 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. InfoSphere DataStage inserts the prefix before each field.

  This property is mutually exclusive with the Delimiter, Quote, and Final Delimiter properties, which are used by default.
- **Print field**. This property is not relevant for input links.
- **Quote**. Specifies that variable length fields are enclosed in single quotes, double quotes, or another character or pair of characters. Choose **Single** or **Double**, or enter a character. This is set to double quotes by default.

  When writing, InfoSphere DataStage inserts the leading quote character, the data, and a trailing quote character. Quote characters are not counted as part of a field's length.
- **Vector prefix**. For fields that are variable length vectors, specifies a 1-, 2-, or 4-byte prefix containing the number of elements in the vector. You can override this default prefix for individual vectors.

  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 this property to indicate the prefix length. InfoSphere DataStage inserts the element count as a prefix of each variable-length vector field. By default, the prefix length is assumed to be one byte.

Type defaults

These are properties that apply to all columns of a specific data type unless specifically overridden at the column level. They are divided into a number of subgroups according to data type.

General

These properties apply to several data types (unless overridden at column level):

- **Byte order**. Specifies how multiple byte data types (except string and raw data types) are ordered. Choose from:
  - little-endian. The high byte is on the right.
  - big-endian. The high byte is on the left.
  - native-endian. As defined by the native format of the machine. This is the default.
- **Data Format**. Specifies the data representation format of a field. Applies to fields of all data types except string, ustring, and raw and to record, subrec or tagged fields containing at least one field that is neither string nor raw. Choose from:
  - binary
  - text (the default)

    A setting of binary has different meanings when applied to different data types:
  - For decimals, binary means packed.
  - For other numerical data types, binary means "not text".
  - For dates, binary is equivalent to specifying the julian property for the date field.
  - For time, binary is equivalent to 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. A binary timestamp specifies that two 32-but integers are written.

    By default data is formatted as text, as follows:
  - For the date data type, text specifies that the data to be written contains a text-based date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).

  - For the decimal data type: a 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. The destination string format is: [+ | -]ddd.[ddd] and any precision and scale arguments are ignored.
  - For *numeric* fields (int8, int16, int32, uint8, uint16, uint32, sfloat, and dfloat): InfoSphere DataStage assumes that numeric fields are represented as text.
  - For the time data type: text specifies that the field represents time in the text-based form %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
  - For the *timestamp* data type: text specifies a text-based timestamp in the form *%yyyy-%mm-%dd %hh:%nn:%ss* or in the default date format if you have defined a new one on an NLS system.
- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.
- **Field width**. The number of bytes in a field represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.

  If you specify neither field width nor field max width, numeric fields written 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)

  Important: If you are using Unicode character columns, you must calculate the field length in bytes and specify that value in the Field Width column property.
- **Pad char**. Specifies the pad character used when strings or numeric values are written to an external string representation. Enter a character (single-byte for strings, can be multi-byte for ustrings) or choose null or space. The pad character is used when the external string representation is larger than required to hold the written field. In this case, the external string is filled with the pad character to its full length. Space is the default. Applies to string, ustring, and numeric data types and record, subrec, or tagged types if they contain at least one field of this type.
- **Character set**. Specifies the character set. Choose from ASCII or EBCDIC. The default is ASCII. Applies to all data types except raw and ustring and record, subrec, or tagged containing no fields other than raw or ustring.

String

These properties are applied to columns with a string data type, unless overridden at column level.

- **Export EBCDIC as ASCII**. Select this to specify that EBCDIC characters are written as ASCII characters. Applies to fields of the string data type and record, subrec, or tagged fields if they contain at least one field of this type.
- **Import ASCII as EBCDIC**. Not relevant for input links.

Decimal

These properties are applied to columns with a decimal data type unless overridden at column level.

- **Allow all zeros**. Specifies whether to treat a packed decimal column containing all zeros (which is normally illegal) as a valid representation of zero. Select **Yes** or **No**. The default is **No**.
- **Decimal separator.** Specify the ASCII character that acts as the decimal separator (period by default).
- **Packed**. Select an option to specify what the decimal columns contain, choose from:
  - **Yes** to specify that the decimal columns contain data in packed decimal format (the default). This has the following sub-properties:

    Check. Select **Yes** to verify that data is packed, or **No** to not verify.

    Signed. Select **Yes** to use the existing sign when writing decimal columns. Select **No** to write a positive sign (0xf) regardless of the columns' actual sign value.
  - **No (separate)** to specify that they contain unpacked decimal with a separate sign byte. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - **No (zoned)** to specify that they contain an unpacked decimal in either ASCII or EBCDIC text. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - **No (overpunch)** to specify that the field has a leading or end 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 has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
- **Precision**. Specifies the precision where a decimal column is written in text format. Enter a number. When a decimal is written to a string representation, InfoSphere DataStage 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. When they are defined, InfoSphere DataStage truncates or pads the source decimal to fit the size of the destination string. If you have also specified the field width property, InfoSphere DataStage truncates or pads the source decimal to fit the size specified by field width.
- **Rounding**. Specifies how to round a decimal column when writing it. Choose from:
  - up (ceiling). Truncate source column towards positive infinity. This mode corresponds to the IEEE 754 Round Up mode. For example, 1.4 becomes 2, -1.6 becomes -1.
  - down (floor). Truncate source column towards negative infinity. This mode corresponds to the IEEE 754 Round Down mode. For example, 1.6 becomes 1, -1.4 becomes -2.
  - nearest value. Round the source column towards the nearest representable value. This mode corresponds to the COBOL ROUNDED mode. For example, 1.4 becomes 1, 1.5 becomes 2, -1.4 becomes -1, -1.5 becomes -2.
  - truncate towards zero. This is the default. Discard fractional digits to the right of the right-most fractional digit supported by 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. Using this method 1.6 becomes 1, -1.6 becomes -1.
- **Scale**. Specifies how to round a source decimal when its precision and scale are greater than those of the destination. By default, when the InfoSphere DataStage writes 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, InfoSphere DataStage truncates or pads the source decimal to fit the size of the destination string. If you have also specified the field width property, InfoSphere DataStage truncates or pads the source decimal to fit the size specified by field width.

Numeric

These properties apply to integer and float fields unless overridden at column level.

- **C_format**. Perform non-default conversion of data from integer or floating-point data to a string. This property specifies a C-language format string used for writing integer or floating point strings. This is passed to *sprintf*(). For example, specifying a C-format of %x and a field width of 8 ensures that integers are written as 8-byte hexadecimal strings.
- **In_format**. This property is not relevant for input links..
- **Out_format**. Format string used for conversion of data from integer or floating-point data to a string. This is passed to *sprintf*(). By default, InfoSphere 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 to pass formatting arguments to *sprintf*().

Date

These properties are applied to columns with a date data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Days since**. Dates are written as a signed integer containing the number of days since the specified date. Enter a date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
- **Format string**. The string format of a date. By default this is %yyyy-%mm-%dd. For details about the format, see "Date formats" on page 31.
- **Is Julian**. Select this to specify that dates are written as a numeric value containing the Julian day. A Julian day specifies the date as the number of days from 4713 BCE January 1, 12:00 hours (noon) GMT.

Time

These properties are applied to columns with a time data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Format string**. Specifies the format of columns representing time as a string. For details about the format, see "Time formats" on page 35
- **Is midnight seconds**. Select this to specify that times are written as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.

Timestamp

These properties are applied to columns with a timestamp data type unless overridden at column level.

- **Format string**. Specifies the format of a column representing a timestamp as a string. Defaults to *%yyyy-%mm-%dd %hh:%nn:%ss*. The format combines the format for date strings and time strings. See "Date formats" on page 31 and "Time formats" on page 35.

File Set stage: Output page

The **Output page** allows you to specify details about how the File Set stage reads data from a file set. The File Set stage can have only one output link. It can also have a single reject link, where rows that have failed to be written or read for some reason can be sent. The **Output name** drop-down list allows you to choose whether you are looking at details of the main output link (the stream link) or the reject link.

The General tab allows you to specify an optional description of the output link. The Properties tab allows you to specify details of exactly what the link does. The Formats tab gives information about the format of the files being read. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

Details about File Set stage properties and formatting are given in the following sections. See Chapter 4, "Stage editors," on page 49, for a general description of the other tabs.

File Set stage: Output link properties tab

The Properties tab allows you to specify properties for the output link. These dictate how incoming data is read from files in the file set. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Source/File Set	pathname	N/A	Y	N	N/A
Options/Keep file Partitions	True/False	False	Y	N	N/A
Options/Reject Mode	Continue/Fail/ Save	Continue	Y	N	N/A
Options/Report Progress	Yes/No	Yes	Y	N	N/A
Options/Filter	command	N/A	N	N	N/A
Options/Schema File	pathname	N/A	N	N	N/A
Options/Use Schema Defined in File Set	True/False	False	Y	N	N/A
Options/File Name Column	column name	N/A	N	N	N/A
Options/"Row number column" on page 126	column name	N/A	N	N	N/A
Options/"Strip BOM" on page 126	True/False	FALSE	N	N	N/A

File Set stage: Source category:
File set

This property defines the file set that the data will be read from. You can type in a pathname of, or browse for, a file set descriptor file (by convention ending in .fs).

File Set stage: Options category:
Keep file partitions

Set this to **True** to partition the read data set according to the organization of the input file(s). So, for example, if you are reading three files you will have three partitions. Defaults to **False**.

Reject mode

Allows you to specify behavior for read rows that do not match the expected schema. Choose from **Continue** to continue operation and discard any rejected rows, **Fail** to cease reading if any rows are rejected, or **Save** to send rejected rows down a reject link. Defaults to **Continue**.

Report progress

Choose **Yes** or **No** to enable or disable reporting. By default the stage displays a progress report at each 10% interval when it can ascertain file size. Reporting occurs only if the file is greater than 100 KB, records are fixed length, and there is no filter on the file.

Filter

This is an optional property. You can use this to specify that the data is passed through a filter program after being read from the files. Specify the filter command, and any required arguments, in the Property Value box.

Schema file

This is an optional property. By default the File Set stage will use the column definitions defined on the **Columns** and **Format** tabs as a schema for reading the file. You can, however, specify a file containing a schema instead (note, however, that if you have defined columns on the **Columns** tab, you should ensure these match the schema file). Type in a pathname or browse for a schema file. This property is mutually exclusive with Use Schema Defined in File Set.

Use schema defined in file set

When you create a file set you have an option to save the schema along with it. When you read the file set you can use this schema in preference to the column definitions by setting this property to True. This property is mutually exclusive with Schema File.

File name column

This is an optional property. It adds an extra column of type VarChar to the output of the stage, containing the pathname of the file the record is read from. You should also add this column manually to the Columns definitions to ensure that the column is not dropped if you are not using runtime column propagation, or it is turned off at some point.

Row number column

This is an optional property. It adds an extra column of type unsigned BigInt to the output of the stage, containing the row number. You must also add the column to the columns tab, unless runtime column propagation is enabled.

Strip BOM

Set this property TRUE to drop the UTF-16 Endianess byte order mark when reading data. By default, this property is set to FALSE.

File Set stage: Reject link properties

You cannot change the properties of a Reject link. The Properties tab for a reject link is blank.

Similarly, you cannot edit the column definitions for a reject link. For writing file sets, the link uses the column definitions for the input link. For reading file sets, the link uses a single column called rejected containing raw data for columns rejected after reading because they do not match the schema.

File Set stage: Output link Format tab

The Format tab allows you to supply information about the format of the flat file or files that you are writing. The tab has a similar format to the Properties tab.

If you do not alter any of the Format settings, the stage will produce a file of the following format:

- File comprises variable length columns contained within double quotes.
- All columns are delimited by a comma, except for the final column in a row.
- Rows are delimited by a UNIX newline.

You can use the **Format As** item from the shortcut menu in the Format tab to quickly change to a fixed-width column format, using DOS newlines as row delimiters, or producing a COBOL format file.

You can use the **Defaults** button to change your default settings. Use the Format tab to specify your required settings, then click **Defaults** > **Save current as default**. All your sequential files will use your settings by default from now on. If your requirements change, you can choose **Defaults** > **Reset defaults from factory settings** to go back to the original defaults as described above. Once you have done this, you then have to click **Defaults** > **Set current from default** for the new defaults to take effect.

To change individual properties, select a property type from the main tree then add the properties you want to set to the tree structure by clicking on them in the **Available properties to addwindow**. You can then set a value for that property in the Property Value box. Pop-up help for each of the available properties appears if you hover the mouse pointer over it.

Any property that you set on this tab can be overridden at the column level by setting properties for individual columns on the Edit Column Metadata dialog box (see Columns Tab).

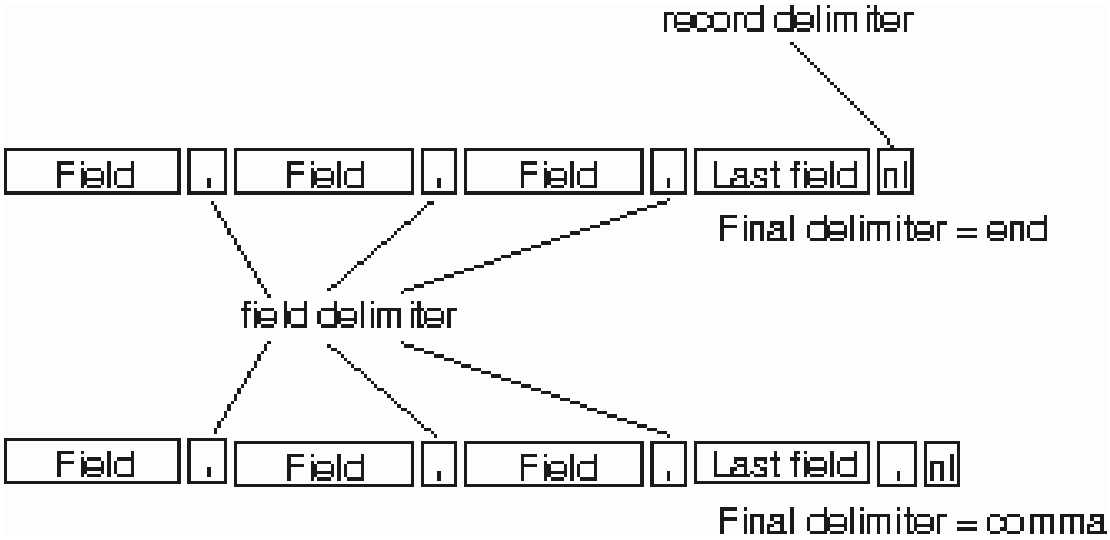
This description uses the terms "record" and "row" and "field" and "column" interchangeably.

The following sections list the property types and properties available for each type.

Record level

These properties define details about how data records are formatted in the flat file. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Fill char.** Does not apply to output links.
- **Final delimiter string**. Specify the string written after the last column of a record in place of the column delimiter. Enter one or more characters, this precedes the record delimiter if one is used. Mutually exclusive with Final delimiter, which is the default. For example, if you set Delimiter to comma and Final delimiter string to `, ` (comma space - you do not need to enter the inverted commas) all fields are delimited by a comma, except the final field, which is delimited by a comma followed by an ASCII space character. InfoSphere DataStage skips the specified delimiter string when reading the file.
- **Final delimiter**. Specify the single character written after the last column of a record in place of the field delimiter. Type a character or select one of whitespace, end, none, null, tab, or comma. InfoSphere DataStage skips the specified delimiter string when reading the file. See the following diagram for an illustration.
  - **whitespace**. The last column of each record will not include any trailing white spaces found at the end of the record.
  - **end**. The last column of each record does not include the field delimiter. This is the default setting.
  - **none**. The last column of each record does not have a delimiter, used for fixed-width fields.
  - **null**. The last column of each record is delimited by the ASCII null character.
  - **comma**. The last column of each record is delimited by the ASCII comma character.
  - **tab**. The last column of each record is delimited by the ASCII tab character.



- **Intact**. The intact property specifies an identifier of a partial schema. A partial schema specifies that only the column(s) named in the schema can be modified by the stage. All other columns in the row are passed through unmodified. The file containing the partial schema is specified in the Schema File property on the **Outputs** tab. This property has a dependent property:
  - **Check intact**. Select this to force validation of the partial schema as the file or files are imported. Note that this can degrade performance.
- **Record delimiter string**. Specify the string at the end of each record. Enter one or more characters. This is mutually exclusive with Record delimiter, which is the default, and record type and record prefix.
- **Record delimiter**. Specify the single character at the end of each record. Type a character or select one of the following:
  - UNIX Newline (the default)
  - null

  (To specify a DOS newline, use the Record delimiter string property set to "\R\N" or choose **Format as** > **DOS line terminator** from the shortcut menu.)

  Record delimiter is mutually exclusive with Record delimiter string, Record prefix, and record type.
- **Record length**. Select Fixed where fixed length fields are being read. InfoSphere DataStage calculates the appropriate length for the record. Alternatively specify the length of fixed records as number of bytes. This is not used by default (default files are comma-delimited).
- **Record Prefix**. Specifies that a variable-length record is prefixed by a 1-, 2-, or 4-byte length prefix. It is set to 1 by default. This is mutually exclusive with Record delimiter, which is the default, and record delimiter string and record type.
- **Record type**. Specifies that data consists of variable-length blocked records (varying) or implicit records (implicit). If you choose the implicit property, data is written as a stream with no explicit record boundaries. The end of the record is inferred when all of the columns defined by the schema have been parsed. The varying property allows you to specify one of the following IBM blocked or spanned formats: V, VB, VS, VBS, or VR.

  This property is mutually exclusive with Record length, Record delimiter, Record delimiter string, and Record prefix and by default is not used.

Field Defaults

Defines default properties for columns read from the file or files. These are applied to all columns, but can be overridden for individual columns from the Columns tab using the Edit Column Metadata dialog box. A common reason to override a property for an individual column occurs when reading

comma-separated values (CSV) files. CSV files often enclose fields in quotes when the fields might contain a special character, such as the field delimiter. In this case, the **Quote** property for the columns in question should be overridden.

Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Actual field length**. Specifies the actual number of bytes to skip if the field's length equals the setting of the null field length property.
- **Delimiter**. Specifies the trailing delimiter of all fields in the record. Type an ASCII character or select one of whitespace, end, none, null, comma, or tab. InfoSphere DataStage skips the delimiter when reading.
  - **whitespace**. Whitespace characters at the end of a column are ignored, that is, are not treated as part of the column.
  - **end**. The end of a field is taken as the delimiter, that is, there is no separate delimiter. This is not the same as a setting of `None' which is used for fields with fixed-width columns.
  - **none**. No delimiter (used for fixed-width).
  - **null**. ASCII Null character is used.
  - **comma**. ASCII comma character is used.
  - **tab**. ASCII tab character is used.
- **Delimiter string**. Specify the string at the end of each field. Enter one or more characters. This is mutually exclusive with Delimiter, which is the default. For example, specifying `, ` (comma space - you do not need to enter the inverted commas) specifies each field is delimited by `, ` unless overridden for individual fields. InfoSphere DataStage skips the delimiter string when reading.
- **Null field length**. The length in bytes of a variable-length field that contains a null. When a variable-length field is read, a length of null field length in the source field indicates that it contains a null. This property is mutually exclusive with null field value.
- **Null field value**. Specifies the value given to a null field if the source is set to null. Can be a number, string, or C-type 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 \x*hh*, where each *h* is a hexadecimal digit 0 - F. You must use this form to encode non-printable byte values.

  This property is mutually exclusive with Null field length and Actual length. For a fixed width data representation, you can use Pad char (from the general section of Type defaults) to specify a repeated trailing character if the value you specify is shorter than the fixed width of the field.

  You can specify a list of null values that a column could contain that represent null. To do this you specify a separator character in the dependent **Null field value separator** property, and then use this separator to delimit the null values in the **Null field value** property. For example, if you set **Null field value separator** to contain the slash character (/), then you could specify NULL/null/NUL/nul to specify that any of these strings could represent a null value in this column.
- **Null field value separator**

  This is a dependent property of **Null field value**. You can specify a separator that can be used in the **Null field value** property to specify a range of values that could represent the null value. You can specify a number, string, or C-type literal escape character (as for **Null field value**) as a separator, but a single character such as a comma (,) or slash (/) character is the best choice. You must only specify a separator if you specify multiple values in **Null field value**; specifying a separator without using it will cause a runtime error.
- **Prefix bytes**. 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. InfoSphere DataStage reads the length prefix but does not include the prefix as a separate field in the data set it reads from the file.

  This property is mutually exclusive with the Delimiter, Quote, and Final Delimiter properties, which are used by default.

- **Print field**. This property is intended for use when debugging jobs. Set it to have InfoSphere DataStage produce a message for every field it reads. The message has the format:

  `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.
- **Quote**. Specifies that variable length fields are enclosed in single quotes, double quotes, or another character or pair of characters. Choose Single or Double, or enter a character. This is set to double quotes by default.

  When reading, InfoSphere DataStage ignores the leading quote character and reads all bytes up to but not including the trailing quote character.
- **Vector prefix**. For fields that are variable length vectors, specifies that a 1-, 2-, or 4-byte prefix contains the number of elements in the vector. You can override this default prefix for individual vectors.

  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 this property to indicate the prefix length. InfoSphere DataStage reads the length prefix but does not include it as a separate field in the data set. By default, the prefix length is assumed to be one byte.

Type Defaults

These are properties that apply to all columns of a specific data type unless specifically overridden at the column level. They are divided into a number of subgroups according to data type.

General

These properties apply to several data types (unless overridden at column level):

- **Byte order**. Specifies how multiple byte data types (except string and raw data types) are ordered. Choose from:
  – little-endian. The high byte is on the right.
  – big-endian. The high byte is on the left.
  – native-endian. As defined by the native format of the machine. This is the default.
- **Data Format**. Specifies the data representation format of a field. Applies to fields of all data types except string, ustring, and raw and to record, subrec or tagged fields containing at least one field that is neither string nor raw. Choose from:
  – binary
  – text (the default)

  A setting of binary has different meanings when applied to different data types:
  – For decimals, binary means packed.
  – For other numerical data types, binary means "not text".
  – For dates, binary is equivalent to specifying the julian property for the date field.
  – For time, binary is equivalent to 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. A binary timestamp specifies that two 32-but integers are written.

  By default data is formatted as text, as follows:
  – For the date data type, text specifies that the data read, contains a text-based date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system.

  - For the decimal data type: a 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. The destination string format is: [+ | -]ddd.[ddd] and any precision and scale arguments are ignored.
  - For numeric fields (int8, int16, int32, uint8, uint16, uint32, sfloat, and dfloat): InfoSphere DataStage assumes that numeric fields are represented as text.
  - For the time data type: text specifies that the field represents time in the text-based form %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system.
  - For the timestamp data type: text specifies a text-based timestamp in the form %yyyy-%mm-%dd %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.
- **Field width**. The number of bytes in a field represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.

  If you specify neither field width nor field max width, numeric fields written 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)
- **Pad char**. This property is ignored for output links.
- **Character set**. Specifies the character set. Choose from ASCII or EBCDIC. The default is ASCII. Applies to all data types except raw and ustring and record, subrec, or tagged containing no fields other than raw or ustring.

String

These properties are applied to columns with a string data type, unless overridden at column level.

- **Export EBCDIC as ASCII**. Not relevant for output links.
- **Import ASCII as EBCDIC**. Select this to specify that ASCII characters are read as EBCDIC characters.

Decimal

These properties are applied to columns with a decimal data type unless overridden at column level.

- **Allow all zeros**. Specifies whether to treat a packed decimal column containing all zeros (which is normally illegal) as a valid representation of zero. Select Yes or No. The default is No.
- **Decimal separator**. Specify the ASCII character that acts as the decimal separator (period by default).
- **Packed**. Select an option to specify what the decimal columns contain, choose from:
  - Yes to specify that the decimal fields contain data in packed decimal format (the default). This has the following sub-properties:

    Check. Select Yes to verify that data is packed, or No to not verify.

Signed. Select Yes to use the existing sign when reading decimal fields. Select No to write a positive sign (0xf) regardless of the fields' actual sign value.

- No (separate) to specify that they contain unpacked decimal with a separate sign byte. This has the following sub-property:

  Sign Position. Choose leading or trailing as appropriate.
- No (zoned) to specify that they contain an unpacked decimal in either ASCII or EBCDIC text. This has the following sub-property:

  Sign Position. Choose leading or trailing as appropriate.
- No (overpunch) to specify that the field has a leading or end 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 has the following sub-property:

  Sign Position. Choose leading or trailing as appropriate.

- **Precision**. Specifies the precision of a packed decimal. Enter a number.
- **Rounding**. Specifies how to round the source field to fit into the destination decimal when reading a source field to a decimal. Choose from:
  - **up (ceiling)**. Truncate source column towards positive infinity. This mode corresponds to the IEEE 754 Round Up mode. For example, 1.4 becomes 2, -1.6 becomes -1.
  - **down (floor)**. Truncate source column towards negative infinity. This mode corresponds to the IEEE 754 Round Down mode. For example, 1.6 becomes 1, -1.4 becomes -2.
  - **nearest value**. Round the source column towards the nearest representable value. This mode corresponds to the COBOL ROUNDED mode. For example, 1.4 becomes 1, 1.5 becomes 2, -1.4 becomes -1, -1.5 becomes -2.
  - **truncate towards zero**. This is the default. Discard fractional digits to the right of the right-most fractional digit supported by 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. Using this method 1.6 becomes 1, -1.6 becomes -1.
- **Scale**. Specifies the scale of a source packed decimal.

Numeric

These properties apply to integer and float fields unless overridden at column level.

- **C_format**. Perform non-default conversion of data from string data to a integer or floating-point. This property specifies a C-language format string used for reading integer or floating point strings. This is passed to *sscanf*(). For example, specifying a C-format of %x and a field width of 8 ensures that a 32-bit integer is formatted as an 8-byte hexadecimal string.
- **In_format**. Format string used for conversion of data from string to integer or floating-point data This is passed to *sscanf*(). By default, InfoSphere DataStage invokes the C *sscanf*() function to convert a numeric field formatted as a string to either integer or floating point data. If this function does not output data in a satisfactory format, you can specify the in_format property to pass formatting arguments to *sscanf*().
- **Out_format**. This property is not relevant for output links.

Date

These properties are applied to columns with a date data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Days since**. Dates are written as a signed integer containing the number of days since the specified date. Enter a date in the form *%yyyy-%mm-%d*d or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).

- **Format string**. The string format of a date. By default this is %yyyy-%mm-%dd. For details about the format, see “Date formats” on page 31.
- **Is Julian**. Select this to specify that dates are written as a numeric value containing the Julian day. A Julian day specifies the date as the number of days from 4713 BCE January 1, 12:00 hours (noon) GMT.

Time

These properties are applied to columns with a time data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Format string**. Specifies the format of columns representing time as a string. By default this is %hh-%mm-%ss. For details about the format, see “Time formats” on page 35
- **Is midnight seconds**. Select this to specify that times are written as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.

Timestamp

These properties are applied to columns with a timestamp data type unless overridden at column level.

- **Format string**. Specifies the format of a column representing a timestamp as a string. The format combines the format for date strings and time strings. See “Date formats” on page 31 and “Time formats” on page 35.

Using RCP With file set stages

Runtime column propagation (RCP) allows InfoSphere DataStage to be flexible about the columns you define in a job. If RCP is enabled for a project, you can just define the columns you are interested in using in a job, but ask InfoSphere DataStage to propagate the other columns through the various stages. So such columns can be extracted from the data source and end up on your data target without explicitly being operated on in between.

The File Set stage handles a set of sequential files. Sequential files, unlike most other data sources, do not have inherent column definitions, and so InfoSphere DataStage cannot always tell where there are extra columns that need propagating. You can only use RCP on File Set stages if you have used the Schema File property (see "Schema File") to specify a schema which describes all the columns in the sequential files referenced by the stage. You need to specify the same schema file for any similar stages in the job where you want to propagate columns. Stages that will require a schema file are:

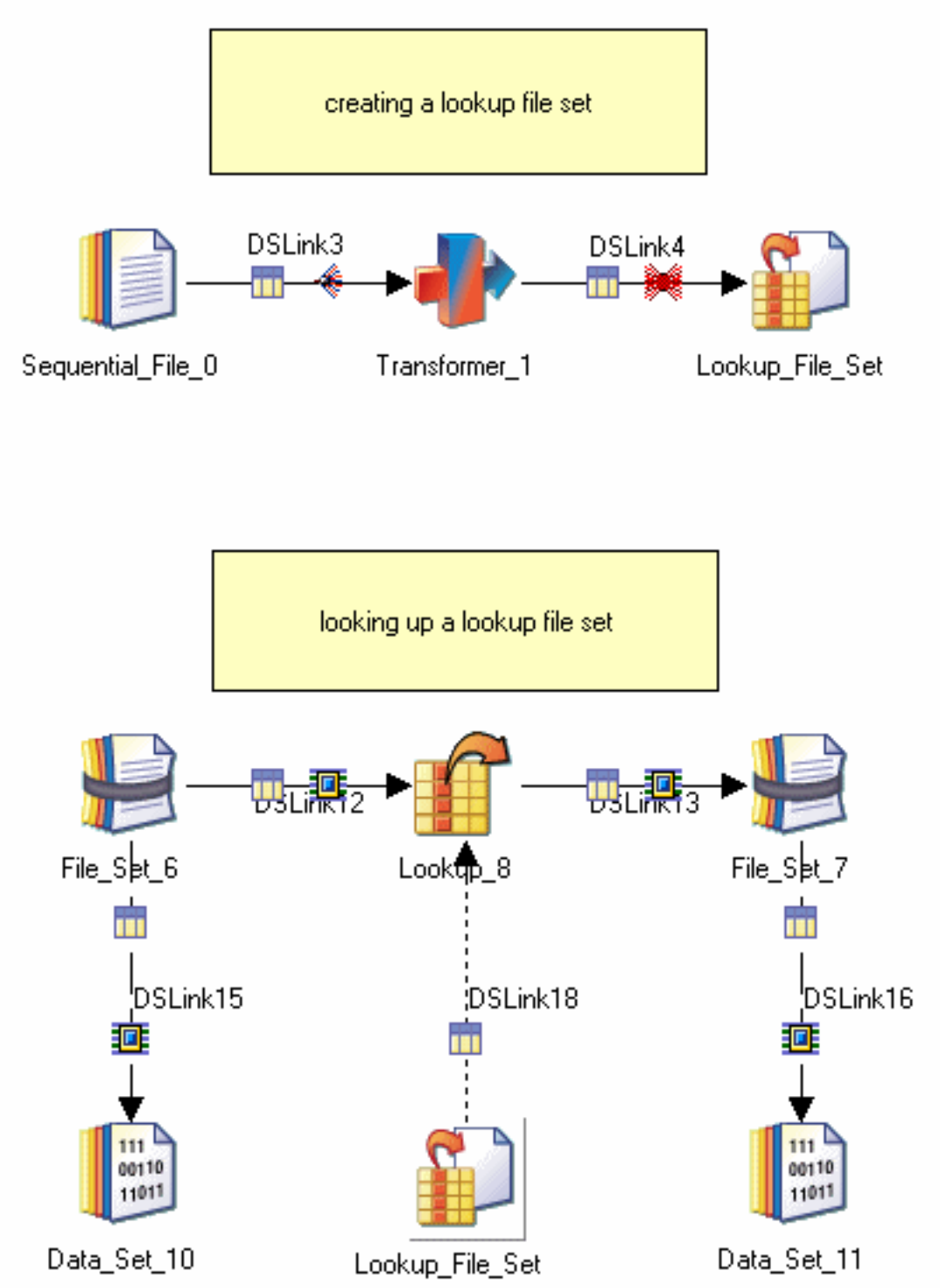
- Sequential File
- File Set
- External Source
- External Target
- Column Import
- Column Export

Lookup file set stage

The Lookup File Set stage is a file stage. It allows you to create a lookup file set or reference one for a lookup. The stage can have a single input link or a single output link. The output link must be a reference link. The stage can be configured to execute in parallel or sequential mode when used with an input link.

When creating Lookup file sets, one file will be created for each partition. The individual files are referenced by a single descriptor file, which by convention has the suffix .fs.

When performing lookups, Lookup File Set stages are used with Lookup stages. For more information about lookup operations, see "Lookup Stage."



When you use a Lookup File Set stage as a source for lookup data, there are special considerations about column naming. If you have columns of the same name in both the source and lookup data sets, the source data set column will go to the output data. If you want this column to be replaced by the column from the lookup data source, you need to drop the source data column before you perform the lookup (you could, for example, use a Modify stage to do this). See "Lookup Stage" for more details about performing lookups.

When you edit a Lookup File Set stage, the Lookup File Set stage editor appears. This is based on the generic stage editor described in "Stage Editors."

The stage editor has up to three pages, depending on whether you are creating or referencing a file set:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is present when you are creating a lookup table. This is where you specify details about the file set being created and written to.
- **Output Page**. This is present when you are reading from a lookup file set, that is, where the stage is providing a reference link to a Lookup stage. This is where you specify details about the file set being read from.

Lookup File Set stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Lookup File Set stages in a job. This section specifies the minimum steps to take to get a Lookup File Set stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

The steps required depend on whether you are using the Lookup File Set stage to create a lookup file set, or using it in conjunction with a Lookup stage.

Creating a lookup file set

You can create a lookup file set

Procedure

1. Go to the **Input Link Properties Tab**.
2. Specify the key that the lookup on this file set will ultimately be performed on. You can repeat this property to specify multiple key columns. You must specify the key when you create the file set, you cannot specify it when performing the lookup.
3. Specify the name of the Lookup File Set.
4. Specify a lookup range, or accept the default setting of **No**.
5. Set **Allow Duplicates**, or accept the default setting of **False**.
6. Ensure column meta data has been specified for the lookup file set.

Looking up a lookup file set

About this task

- In the **Output Link Properties Tab** specify the name of the lookup file set being used in the lookup.
- Ensure column meta data has been specified for the lookup file set.

Lookup File Set stage: Stage page

The General tab allows you to specify an optional description of the stage. The Advanced tab allows you to specify how the stage executes. The NLS Map tab, which appears if you have NLS enabled on your system, allows you to specify a character set map for the stage.

Lookup File Set stage: Advanced tab

This tab only appears when you are using the stage to create a reference file set (that is, where the stage has an input link). Use this tab to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the contents of the table are processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In sequential mode the entire contents of the table are processed by the conductor node.
- **Combinability mode**. This is **Auto** by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pools or pools specified in the grid. The grid allows you to make choices from lists populated from the Configuration file.

Lookup File Set stage: NLS Map tab

The NLS Map tab allows you to define a character set map for the Lookup File Set stage. This overrides the default character set map set for the project or the job. You can specify that the map be supplied as a job parameter if required. You can also select **Allow per-column mapping**. This allows character set maps to be specified for individual columns within the data processed by the External Source stage. An extra property, NLS Map, appears in the Columns grid in the Columns tab, but note that only ustring data types allow you to set an NLS map value (see "Data Types") .

Lookup File Set stage: Input page

The Input page allows you to specify details about how the Lookup File Set stage writes data to a file set. The Lookup File Set stage can have only one input link.

The General tab allows you to specify an optional description of the input link. The Properties tab allows you to specify details of exactly what the link does. The Partitioning tab allows you to specify how incoming data is partitioned before being written to the file set. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Lookup File Set stage properties and partitioning are given in the following sections. See "Stage Editors" for a general description of the other tabs.

Lookup File Set stage: Input link properties tab

The Properties tab allows you to specify properties for the input link. These dictate how incoming data is written to the file set. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 16. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Lookup Keys/Key	Input column	N/A	Y	Y	N/A

Table 16. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Lookup Keys/Case Sensitive	True/False	True	N	N	Key
Lookup Keys/Keep	True/False	False	Y	N	Key
Target/Lookup File Set	pathname	N/A	Y	N	N/A
Lookup Range/Range Lookup	Yes/No	No	Y	Y	N/A
Lookup Range/Bounded Column	Input column	N/A	N	N	Range Lookup
Lookup Range/Case Sensitive	True/False	True	N	N	Bounded Column
Lookup Range/Keep	True/False	False	N	N	Bounded Column
Lookup Range/Lower Bound	Input column	N/A	N	N	Range Lookup
Lookup Range/Case Sensitive	True/False	True	N	N	Lower Bound
Lookup Range/Keep	True/False	False	N	N	Lower Bound
Lookup Range/Upper Bound	Input column	N/A	N	N	Range Lookup
Lookup Range/Case Sensitive	True/False	True	N	N	Upper Bound
Lookup Range/Keep	True/False	False	N	N	Upper Bound
Options/Allow Duplicates	True/False	False	Y	N	N/A
Options/Diskpool	string	N/A	N	N	N/A

Lookup File Set stage: Lookup keys category: Key

Specifies the name of a lookup key column. The **Key** property can be repeated if there are multiple key columns. The property has two dependent properties:

- **Case Sensitive**

  This is a dependent property of **Key** and specifies whether the parent key is case sensitive or not. Set to **True** by default.

- **Keep**

This specifies whether the key column needs to be propagated to the output stream after the lookup, or used in a subsequent condition expression. Set to **False** by default.

Lookup File Set stage: Target category:
Lookup file set

This property defines the file set that the incoming data will be written to. You can type in a pathname of, or browse for, a file set descriptor file (by convention ending in .fs).

Lookup File Set stage: Lookup range category:
Range Lookup

Specifies whether a range lookup is required. A range lookup compares the value of a source column to a range of values between two lookup columns. Alternatively, you can compare the value of a lookup column to a range of values between two source columns. This property has the following dependent properties:

- **Bounded Column**

  Specifies the input column to use for the lookup file set. The lookup table is built with the bounded column, which will then be compared to a range of columns from the source data set. This property is mutually exclusive with Lower Bound and Upper Bound. It has two dependent properties:

  – **Case Sensitive**

    This is an optional property. Specifies whether the column is case sensitive or not. Set to **True** by default.

  – **Keep**

    Specifies whether the column needs to be propagated to the output stream after the lookup, or used in a subsequent condition expression. Set to **False** by default.

- **Lower Bound**

  Specifies the input column to use for the lower bound of the range in the lookup table. This cannot be the same as the upper bound column and is mutually exclusive with Bounded Column. It has two dependent properties:

  – **Case Sensitive**

    This is an optional property. Specifies whether the column is case sensitive or not. Set to **True** by default.

  – **Keep**

    Specifies whether the column needs to be propagated to the output stream after the lookup, or used in a subsequent condition expression. Set to **False** by default.

- **Upper Bound**

  Specifies the input column to use for the upper bound of the range in the lookup table. This cannot be the same as the lower bound column and is mutually exclusive with Bounded Column. It has two dependent properties:

  – **Case Sensitive**

    This is an optional property. Specifies whether the column value is case sensitive or not. Set to **True** by default.

  – **Keep**

    Specifies whether the column needs to be propagated to the output stream after the lookup, or used in a subsequent condition expression. Set to **False** by default.

Lookup File Set stage: Options category:
Allow duplicates

Set this to cause multiple copies of duplicate records to be saved in the lookup table without a warning being issued. Two lookup records are duplicates when all lookup key columns have the same value in the

two records. If you do not specify this option, InfoSphere DataStage issues a warning message when it encounters duplicate records and discards all but the first of the matching records.

Diskpool

This is an optional property. Specify the name of the disk pool into which to write the file set. You can also specify a job parameter.

Lookup File Set stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is written to the lookup file set. It also allows you to specify that the data should be sorted before being written.

By default the stage will write to the file set in entire mode. The complete data set is written to each partition.

If the Lookup File Set stage is operating in sequential mode, it will first collect the data before writing it to the file using the default (auto) collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Lookup File Set stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Lookup File Set stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** list. This will override any current partitioning.

If the Lookup File Set stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** list. This will override the default auto collection method.

The following partitioning methods are available:

- **Entire**. Each file written to receives the entire data set. This is the default partitioning method for the Lookup File Set stage.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following collection methods are available:

- **(Auto)**. This is the default method for the Lookup File Set stage. Normally, when you are using **Auto** mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.

- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The **Partitioning** tab also allows you to specify that data arriving on the input link should be sorted before being written to the file or files. The sort is always carried out within data partitions. If the stage is partitioning incoming data, the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the **Auto** methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Right-click the column in the **Selected** list to invoke the shortcut menu.

Lookup File Set stage: Output page

The Output page allows you to specify details about how the Lookup File Set stage references a file set. The Lookup File Set stage can have only one output link, which is a reference link.

The General tab allows you to specify an optional description of the output link. The Properties tab allows you to specify details of exactly what the link does. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Lookup File Set stage properties are given in the following sections. See "Stage Editors," for a general description of the other tabs.

Lookup File Set stage: Output link properties tab

The Properties tab allows you to specify properties for the output link. These dictate how incoming data is read from the lookup table. There is only one output link property.

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Lookup Source/Lookup File Set	path name	N/A	Y	N	N/A

Lookup File Set stage: Lookup source category:

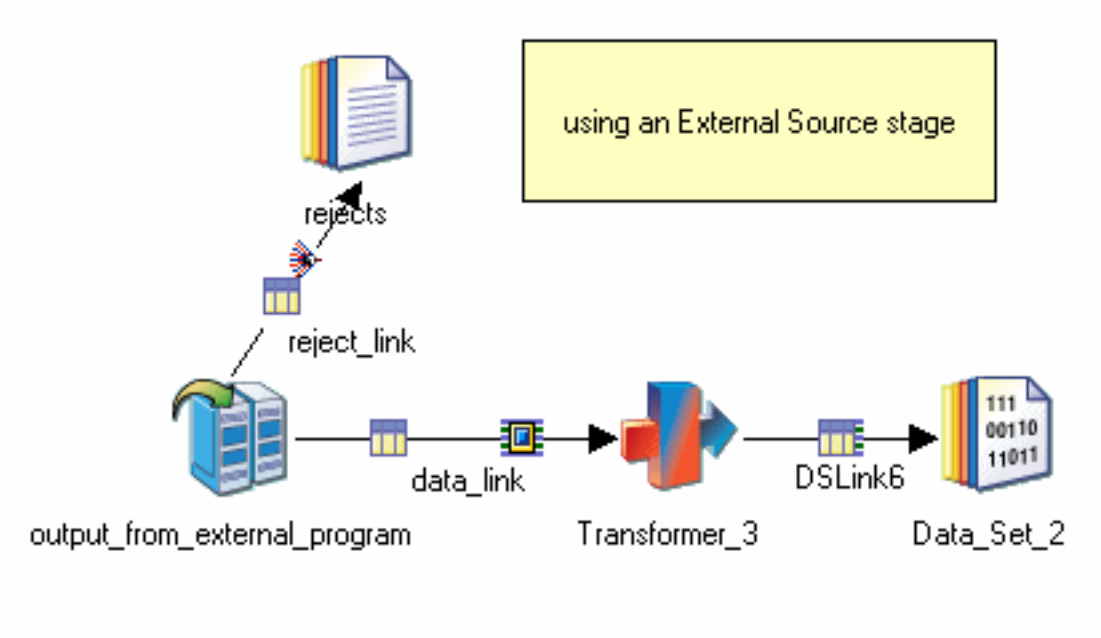
Lookup file set

This property defines the file set that the data will be referenced from. You can type in a pathname of, or browse for, a file set descriptor file (by convention ending in .fs).

External source stage

The External Source stage is a file stage. It allows you to read data that is output from one or more source programs. The stage calls the program and passes appropriate arguments. The stage can have a single output link, and a single rejects link. It can be configured to execute in parallel or sequential mode. There is also an External Target stage which allows you to write to an external program (see "External Target stage" on page 152).

The External Source stage allows you to perform actions such as interfacing with databases that are not currently supported by InfoSphere DataStage.



When reading output from a program, InfoSphere DataStage needs to know something about its format. The information required is how the data is divided into rows and how rows are divided into columns. You specify this on the Format tab. Settings for individual columns can be overridden on the Columns tab using the Edit Column Metadata dialog box.

When you edit an External Source stage, the External Source stage editor appears. This is based on the generic stage editor described in "Stage Editors."

The stage editor has two pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Output Page**. This is where you specify details about the program or programs whose output data you are reading.

There are one or two special points to note about using runtime column propagation (RCP) with External Source stages. See "Using RCP With External Source Stages" for details.

External Source stage: fast path
About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include External Source stages in a job. This section specifies the minimum steps to take to get a External Source stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use the External Source stage:

- In the **Output Link Properties Tab**:
  - Specify whether the stage is providing details of the program (the default) or whether details will be provided in a file (using the latter method you can provide a list of files and arguments).
  - If using the default source method, specify the name of the source program executable. You can also specify required arguments that InfoSphere DataStage will pass when it calls the program. Repeat this to specify multiple program calls.
  - If using the program file source method, specify the name of the file containing the list of program names and arguments.
  - Specify whether to maintain any partitions in the source data (False by default).
  - Specify how to treat rejected rows (by default the stage continues and the rows are discarded).
- In the **Format Tab** specify format details for the source data you are reading from, or accept the defaults (variable length columns enclosed in double quotes and delimited by commas, rows delimited with UNIX newlines).
- Ensure that column definitions have been specified (you can use a schema file for this if required).

External Source stage: Stage page

The General tab allows you to specify an optional description of the stage. The Advanced tab allows you to specify how the stage executes. The NLS Map tab appears if you have NLS enabled on your system, it allows you to specify a character set map for the stage.

External Source stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the data input from external programs is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode all the data from the source program is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. You can select **Set** or **Clear**. If you select **Set**, it will request that the next stage preserves the partitioning as is. **Clear** is the default.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

External Source stage: NLS Map tab

The NLS Map tab allows you to define a character set map for the External Source stage. This overrides the default character set map set for the project or the job. You can specify that the map be supplied as a job parameter if required. You can also select **Allow per-column mapping**. This allows character set maps to be specified for individual columns within the data processed by the External Source stage. An extra property, NLS Map, appears in the Columns grid in the Columns tab, but note that only ustring data types allow you to set an NLS map value (see "Data Types") .

External Source stage: Output page

The Output page allows you to specify details about how the External Source stage reads data from an external program. The External Source stage can have only one output link. It can also have a single reject link, where rows that do not match the expected schema can be sent. The **Output name** drop-down list allows you to choose whether you are looking at details of the main output link (the stream link) or the reject link.

The General tab allows you to specify an optional description of the output link. The Properties tab allows you to specify details of exactly what the link does. The Format tab gives information about the format of the files being read. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

Details about External Source stage properties and formatting are given in the following sections. See "Stage Editors," for a general description of the other tabs.

External Source stage: Output link properties tab

The Properties tab allows you to specify properties for the output link. These dictate how data is read from the external program or programs. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Source/Source Program	string	N/A	Y if Source Method = Specific Program(s)	Y	N/A
Source/Source Programs File	pathname	N/A	Y if Source Method = Program File(s)	Y	N/A
Source/Source Method	Specific Program(s)/ Program File(s)	Specific Program(s)	Y	N	N/A
Options/Keep File Partitions	True/False	False	Y	N	N/A
Options/Reject Mode	Continue/Fail/ Save	Continue	Y	N	N/A
Options/Schema File	pathname	N/A	N	N	N/A

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Source Name Column	column name	N/A	N	N	N/A
Options/"Row number column" on page 145	column name	N/A	N	N	N/A
Options/"Strip BOM" on page 145	True/False	FALSE	N	N	N/A

External Source stage: Source category:
Source program

Specifies the name of a program providing the source data. InfoSphere DataStage calls the specified program and passes to it any arguments specified. You can repeat this property to specify multiple program instances with different arguments. You can use a job parameter to supply program name and arguments.

Source programs file

Specifies a file containing a list of program names and arguments. You can browse for the file or specify a job parameter. You can repeat this property to specify multiple files.

Source method

This property specifies whether you directly specifying a program (using the Source Program property) or using a file to specify a program (using the Source Programs File property).

External Source stage: Options category:
Keep file partitions

Set this to **True** to maintain the partitioning of the read data. Defaults to **False**.

Reject mode

Allows you to specify behavior if a record fails to be read for some reason. Choose from **Continue** to continue operation and discard any rejected rows, **Fail** to cease reading if any rows are rejected, or **Save** to send rejected rows down a reject link. Defaults to **Continue**.

Schema file

This is an optional property. By default the External Source stage will use the column definitions defined on the Columns tab and Schema tab as a schema for reading the file. You can, however, specify a file containing a schema instead (note, however, that if you have defined columns on the Columns tab, you should ensure these match the schema file). Type in a pathname or browse for a schema file.

Source name column

This is an optional property. It adds an extra column of type VarChar to the output of the stage, containing the pathname of the source the record is read from. You should also add this column manually to the Columns definitions to ensure that the column is not dropped if you are not using runtime column propagation, or it is turned off at some point.

Row number column

This is an optional property. It adds an extra column of type unsigned BigInt to the output of the stage, containing the row number. You must also add the column to the columns tab, unless runtime column propagation is enabled.

Strip BOM

Set this property TRUE to drop the UTF-16 Endianess byte order mark when reading data. By default, this property is set to FALSE.

External Source stage: Output link Format tab

The Format tab allows you to supply information about the format of the source data you are reading. The tab has a similar format to the Properties tab.

If you do not alter any of the Format settings, the stage will produce a file of the following format:

- File comprises variable length columns contained within double quotes.
- All columns are delimited by a comma, except for the final column in a row.
- Rows are delimited by a UNIX newline.

You can use the **Format As** item from the shortcut menu in the Format tab to quickly change to a fixed-width column format, using DOS newlines as row delimiters, or producing a COBOL format file.

You can use the **Defaults** button to change your default settings. Use the Format tab to specify your required settings, then click **Defaults** > **Save current as default**. All your sequential files will use your settings by default from now on. If your requirements change, you can choose **Defaults** > **Reset defaults from factory settings** to go back to the original defaults as described above. Once you have done this, you then have to click **Defaults** > **Set current from default** for the new defaults to take effect.

To change individual properties, select a property type from the main tree then add the properties you want to set to the tree structure by clicking on them in the **Available properties to addwindow**. You can then set a value for that property in the Property Value box. Pop-up help for each of the available properties appears if you hover the mouse pointer over it.

Any property that you set on this tab can be overridden at the column level by setting properties for individual columns on the Edit Column Metadata dialog box (see Columns Tab).

This description uses the terms "record" and "row" and "field" and "column" interchangeably.

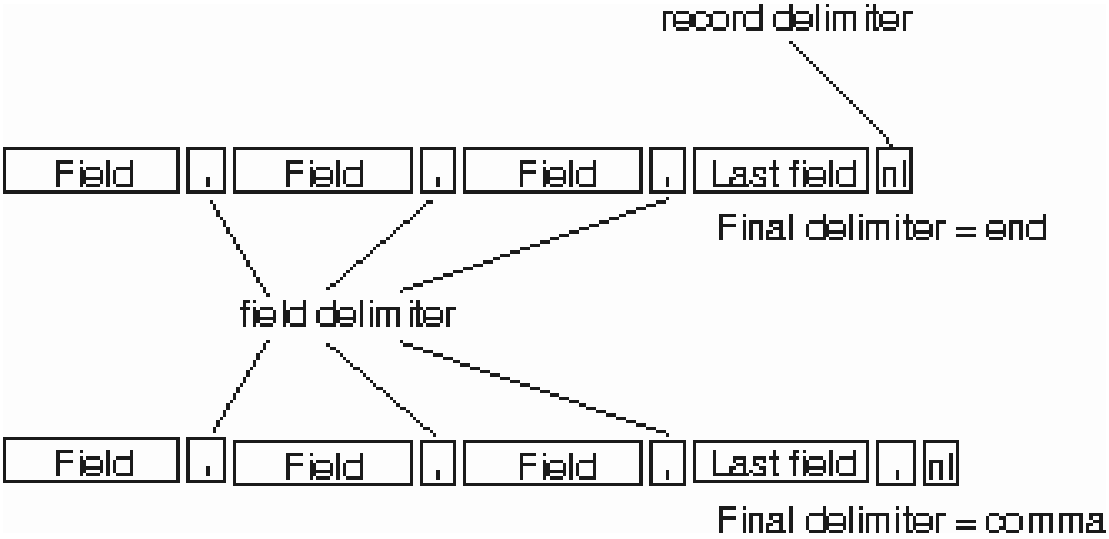
The following sections list the property types and properties available for each type.

Record level

These properties define details about how data records are formatted in the flat file. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Fill char.** Does not apply to output links.
- **Final delimiter string**. Specify the string written after the last column of a record in place of the column delimiter. Enter one or more characters, this precedes the record delimiter if one is used. Mutually exclusive with Final delimiter, which is the default. For example, if you set Delimiter to comma and Final delimiter string to `, ` (comma space - you do not need to enter the inverted commas) all fields are delimited by a comma, except the final field, which is delimited by a comma followed by an ASCII space character. InfoSphere DataStage skips the specified delimiter string when reading the file.

- **Final delimiter**. Specify the single character written after the last column of a record in place of the field delimiter. Type a character or select one of whitespace, end, none, null, tab, or comma. InfoSphere DataStage skips the specified delimiter string when reading the file. See the following diagram for an illustration.
  - **whitespace**. The last column of each record will not include any trailing white spaces found at the end of the record.
  - **end**. The last column of each record does not include the field delimiter. This is the default setting.
  - **none**. The last column of each record does not have a delimiter, used for fixed-width fields.
  - **null**. The last column of each record is delimited by the ASCII null character.
  - **comma**. The last column of each record is delimited by the ASCII comma character.
  - **tab**. The last column of each record is delimited by the ASCII tab character.



- **Intact**. The intact property specifies an identifier of a partial schema. A partial schema specifies that only the column(s) named in the schema can be modified by the stage. All other columns in the row are passed through unmodified. The file containing the partial schema is specified in the Schema File property on the **Outputs** tab. This property has a dependent property:
  - **Check intact**. Select this to force validation of the partial schema as the file or files are imported. Note that this can degrade performance.
- **Record delimiter string**. Specify the string at the end of each record. Enter one or more characters. This is mutually exclusive with Record delimiter, which is the default, and record type and record prefix.
- **Record delimiter**. Specify the single character at the end of each record. Type a character or select one of the following:
  - UNIX Newline (the default)
  - null

  (To specify a DOS newline, use the Record delimiter string property set to "\R\N" or choose **Format as** > **DOS line terminator** from the shortcut menu.)

  Record delimiter is mutually exclusive with Record delimiter string, Record prefix, and record type.
- **Record length**. Select Fixed where fixed length fields are being read. InfoSphere DataStage calculates the appropriate length for the record. Alternatively specify the length of fixed records as number of bytes. This is not used by default (default files are comma-delimited).
- **Record Prefix**. Specifies that a variable-length record is prefixed by a 1-, 2-, or 4-byte length prefix. It is set to 1 by default. This is mutually exclusive with Record delimiter, which is the default, and record delimiter string and record type.

- **Record type**. Specifies that data consists of variable-length blocked records (varying) or implicit records (implicit). If you choose the implicit property, data is written as a stream with no explicit record boundaries. The end of the record is inferred when all of the columns defined by the schema have been parsed. The varying property allows you to specify one of the following IBM blocked or spanned formats: V, VB, VS, VBS, or VR.

  This property is mutually exclusive with Record length, Record delimiter, Record delimiter string, and Record prefix and by default is not used.

Field Defaults

Defines default properties for columns read from the file or files. These are applied to all columns, but can be overridden for individual columns from the Columns tab using the Edit Column Metadata dialog box. A common reason to override a property for an individual column occurs when reading comma-separated values (CSV) files. CSV files often enclose fields in quotes when the fields might contain a special character, such as the field delimiter. In this case, the **Quote** property for the columns in question should be overridden.

Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Actual field length**. Specifies the actual number of bytes to skip if the field's length equals the setting of the null field length property.
- **Delimiter**. Specifies the trailing delimiter of all fields in the record. Type an ASCII character or select one of whitespace, end, none, null, comma, or tab. InfoSphere DataStage skips the delimiter when reading.
  - **whitespace**. Whitespace characters at the end of a column are ignored, that is, are not treated as part of the column.
  - **end**. The end of a field is taken as the delimiter, that is, there is no separate delimiter. This is not the same as a setting of `None' which is used for fields with fixed-width columns.
  - **none**. No delimiter (used for fixed-width).
  - **null**. ASCII Null character is used.
  - **comma**. ASCII comma character is used.
  - **tab**. ASCII tab character is used.
- **Delimiter string**. Specify the string at the end of each field. Enter one or more characters. This is mutually exclusive with Delimiter, which is the default. For example, specifying `, ` (comma space - you do not need to enter the inverted commas) specifies each field is delimited by `, ` unless overridden for individual fields. InfoSphere DataStage skips the delimiter string when reading.
- **Null field length**. The length in bytes of a variable-length field that contains a null. When a variable-length field is read, a length of null field length in the source field indicates that it contains a null. This property is mutually exclusive with null field value.
- **Null field value**. Specifies the value given to a null field if the source is set to null. Can be a number, string, or C-type 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 \x*hh*, where each *h* is a hexadecimal digit 0 - F. You must use this form to encode non-printable byte values.

  This property is mutually exclusive with Null field length and Actual length. For a fixed width data representation, you can use Pad char (from the general section of Type defaults) to specify a repeated trailing character if the value you specify is shorter than the fixed width of the field.

  You can specify a list of null values that a column could contain that represent null. To do this you specify a separator character in the dependent **Null field value separator** property, and then use this separator to delimit the null values in the **Null field value** property. For example, if you set **Null field value separator** to contain the slash character (/), then you could specify NULL/null/NUL/nul to specify that any of these strings could represent a null value in this column.
- **Null field value separator**

This is a dependent property of **Null field value**. You can specify a separator that can be used in the **Null field value** property to specify a range of values that could represent the null value. You can specify a number, string, or C-type literal escape character (as for **Null field value**) as a separator, but a single character such as a comma (,) or slash (/) character is the best choice. You must only specify a separator if you specify multiple values in **Null field value**; specifying a separator without using it will cause a runtime error.

- **Prefix bytes**. 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. InfoSphere DataStage reads the length prefix but does not include the prefix as a separate field in the data set it reads from the file.

  This property is mutually exclusive with the Delimiter, Quote, and Final Delimiter properties, which are used by default.
- **Print field**. This property is intended for use when debugging jobs. Set it to have InfoSphere DataStage produce a message for every field it reads. The message has the format:

  `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.
- **Quote**. Specifies that variable length fields are enclosed in single quotes, double quotes, or another character or pair of characters. Choose Single or Double, or enter a character. This is set to double quotes by default.

  When reading, InfoSphere DataStage ignores the leading quote character and reads all bytes up to but not including the trailing quote character.
- **Vector prefix**. For fields that are variable length vectors, specifies that a 1-, 2-, or 4-byte prefix contains the number of elements in the vector. You can override this default prefix for individual vectors.

  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 this property to indicate the prefix length. InfoSphere DataStage reads the length prefix but does not include it as a separate field in the data set. By default, the prefix length is assumed to be one byte.

Type Defaults

These are properties that apply to all columns of a specific data type unless specifically overridden at the column level. They are divided into a number of subgroups according to data type.

General

These properties apply to several data types (unless overridden at column level):

- **Byte order**. Specifies how multiple byte data types (except string and raw data types) are ordered. Choose from:
  - little-endian. The high byte is on the right.
  - big-endian. The high byte is on the left.
  - native-endian. As defined by the native format of the machine. This is the default.
- **Data Format**. Specifies the data representation format of a field. Applies to fields of all data types except string, ustring, and raw and to record, subrec or tagged fields containing at least one field that is neither string nor raw. Choose from:
  - binary
  - text (the default)

    A setting of binary has different meanings when applied to different data types:

- For decimals, binary means packed.
- For other numerical data types, binary means "not text".
- For dates, binary is equivalent to specifying the julian property for the date field.
- For time, binary is equivalent to 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. A binary timestamp specifies that two 32-but integers are written.

  By default data is formatted as text, as follows:
- For the date data type, text specifies that the data read, contains a text-based date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system.
- For the decimal data type: a 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. The destination string format is: [+ | -]ddd.[ddd] and any precision and scale arguments are ignored.
- For numeric fields (int8, int16, int32, uint8, uint16, uint32, sfloat, and dfloat): InfoSphere DataStage assumes that numeric fields are represented as text.
- For the time data type: text specifies that the field represents time in the text-based form %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system.
- For the timestamp data type: text specifies a text-based timestamp in the form %yyyy-%mm-%dd %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).

- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.
- **Field width**. The number of bytes in a field represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.

  If you specify neither field width nor field max width, numeric fields written 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)
- **Pad char**. This property is ignored for output links.
- **Character set**. Specifies the character set. Choose from ASCII or EBCDIC. The default is ASCII. Applies to all data types except raw and ustring and record, subrec, or tagged containing no fields other than raw or ustring.

String

These properties are applied to columns with a string data type, unless overridden at column level.

- **Export EBCDIC as ASCII**. Not relevant for output links.
- **Import ASCII as EBCDIC**. Select this to specify that ASCII characters are read as EBCDIC characters.

Decimal

These properties are applied to columns with a decimal data type unless overridden at column level.

- **Allow all zeros**. Specifies whether to treat a packed decimal column containing all zeros (which is normally illegal) as a valid representation of zero. Select Yes or No. The default is No.
- **Decimal separator**. Specify the ASCII character that acts as the decimal separator (period by default).
- **Packed**. Select an option to specify what the decimal columns contain, choose from:
  - Yes to specify that the decimal fields contain data in packed decimal format (the default). This has the following sub-properties:

    Check. Select Yes to verify that data is packed, or No to not verify.

    Signed. Select Yes to use the existing sign when reading decimal fields. Select No to write a positive sign (0xf) regardless of the fields' actual sign value.
  - No (separate) to specify that they contain unpacked decimal with a separate sign byte. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - No (zoned) to specify that they contain an unpacked decimal in either ASCII or EBCDIC text. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - No (overpunch) to specify that the field has a leading or end 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 has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
- **Precision**. Specifies the precision of a packed decimal. Enter a number.
- **Rounding**. Specifies how to round the source field to fit into the destination decimal when reading a source field to a decimal. Choose from:
  - **up (ceiling)**. Truncate source column towards positive infinity. This mode corresponds to the IEEE 754 Round Up mode. For example, 1.4 becomes 2, -1.6 becomes -1.
  - **down (floor)**. Truncate source column towards negative infinity. This mode corresponds to the IEEE 754 Round Down mode. For example, 1.6 becomes 1, -1.4 becomes -2.
  - **nearest value**. Round the source column towards the nearest representable value. This mode corresponds to the COBOL ROUNDED mode. For example, 1.4 becomes 1, 1.5 becomes 2, -1.4 becomes -1, -1.5 becomes -2.
  - **truncate towards zero**. This is the default. Discard fractional digits to the right of the right-most fractional digit supported by 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. Using this method 1.6 becomes 1, -1.6 becomes -1.
- **Scale**. Specifies the scale of a source packed decimal.

Numeric

These properties apply to integer and float fields unless overridden at column level.

- **C_format**. Perform non-default conversion of data from string data to a integer or floating-point. This property specifies a C-language format string used for reading integer or floating point strings. This is passed to *sscanf*(). For example, specifying a C-format of %x and a field width of 8 ensures that a 32-bit integer is formatted as an 8-byte hexadecimal string.
- **In_format**. Format string used for conversion of data from string to integer or floating-point data This is passed to *sscanf*(). By default, InfoSphere DataStage invokes the C *sscanf*() function to convert a numeric field formatted as a string to either integer or floating point data. If this function does not output data in a satisfactory format, you can specify the in_format property to pass formatting arguments to *sscanf*().
- **Out_format**. This property is not relevant for output links.

Date

These properties are applied to columns with a date data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Days since**. Dates are written as a signed integer containing the number of days since the specified date. Enter a date in the form *%yyyy-%mm-%d*d or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
- **Format string**. The string format of a date. By default this is %yyyy-%mm-%dd. For details about the format, see "Date formats" on page 31.
- **Is Julian**. Select this to specify that dates are written as a numeric value containing the Julian day. A Julian day specifies the date as the number of days from 4713 BCE January 1, 12:00 hours (noon) GMT.

Time

These properties are applied to columns with a time data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Format string**. Specifies the format of columns representing time as a string. By default this is %hh-%mm-%ss. For details about the format, see "Time formats" on page 35
- **Is midnight seconds**. Select this to specify that times are written as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.

Timestamp

These properties are applied to columns with a timestamp data type unless overridden at column level.

- **Format string**. Specifies the format of a column representing a timestamp as a string. The format combines the format for date strings and time strings. See "Date formats" on page 31 and "Time formats" on page 35.

Using RCP With External Source Stages

Runtime column propagation (RCP) allows InfoSphere DataStage to be flexible about the columns you define in a job. If RCP is enabled for a project, you can just define the columns you are interested in using in a job, but ask InfoSphere DataStage to propagate the other columns through the various stages. So such columns can be extracted from the data source and end up on your data target without explicitly being operated on in between.

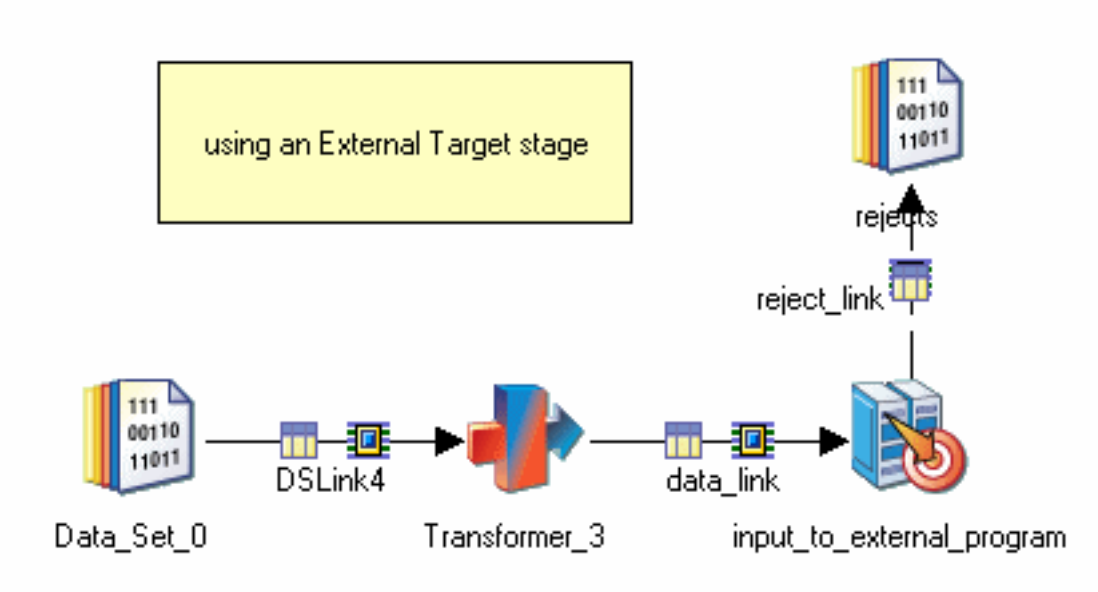
External Source stages, unlike most other data sources, do not have inherent column definitions, and so InfoSphere DataStage cannot always tell where there are extra columns that need propagating. You can only use RCP on External Source stages if you have used the Schema File property (see "Schema File" on page Schema File) to specify a schema which describes all the columns in the sequential files referenced by the stage. You need to specify the same schema file for any similar stages in the job where you want to propagate columns. Stages that will require a schema file are:

- Sequential File
- File Set
- External Source
- External Target
- Column Import
- Column Export

External Target stage

The External Target stage is a file stage. It allows you to write data to one or more source programs. The stage can have a single input link and a single rejects link. It can be configured to execute in parallel or sequential mode. There is also an External Source stage, which allows you to read from an external program (see “External source stage” on page 141)

The External Target stage allows you to perform actions such as interface with databases not currently supported by the InfoSphere DataStage Parallel Extender.



When writing to a program, InfoSphere DataStage needs to know something about how to format the data. The information required is how the data is divided into rows and how rows are divided into columns. You specify this on the **Format** tab. Settings for individual columns can be overridden on the **Columns** tab using the **Edit Column Metadata** dialog box.

When you edit an External Target stage, the External Target stage editor appears. This is based on the generic stage editor described in "Stage Editors."

The stage editor has up to three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the program or programs you are writing data to.
- **Output Page**. This appears if the stage has a rejects link.

There are one or two special points to note about using runtime column propagation (RCP) with External Target stages. See "Using RCP With External Target Stages" for details.

External Target stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include External Target stages in a job. This section specifies the minimum steps to take to get a External Target stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use the External Target stage:

- In the **Input Link Properties Tab**:
  - Specify whether the stage is providing details of the program (the default) or whether details will be provided in a file (using the latter method you can provide a list of files and arguments).
  - If using the default target method, specify the name of the target program executable. You can also specify required arguments that InfoSphere DataStage will pass when it calls the program. Repeat this to specify multiple program calls.
  - If using the program file target method, specify the name of the file containing the list of program names and arguments.
  - Specify whether to delete partially written data if the write fails for some reason (True by default).
  - Specify how to treat rejected rows (by default the stage continues and the rows are discarded).
- In the **Format Tab** specify format details for the data you are writing, or accept the defaults (variable length columns enclosed in double quotes and delimited by commas, rows delimited with UNIX newlines).
- Ensure that column definitions have been specified (this can be done in an earlier stage or in a schema file).

External Target stage: Stage page

The General tab allows you to specify an optional description of the stage. The Advanced tab allows you to specify how the stage executes. The NLS Map tab appears if you have NLS enabled on your system, it allows you to specify a character set map for the stage.

External Target stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the data output to external programs is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode all the data from the source program is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. You can select **Set** or **Clear**. If you select **Set**, it will request that the next stage preserves the partitioning as is. **Clear** is the default.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

External Target stage: NLS Map tab

The NLS Map tab allows you to define a character set map for the External Target stage. This overrides the default character set map set for the project or the job. You can specify that the map be supplied as a job parameter if required. You can also select **Allow per-column mapping**. This allows character set maps to be specified for individual columns within the data processed by the External Target stage. An extra property, **NLS Map**, appears in the Columns grid in the Columns tab, but note that only ustring data types allow you to set an NLS map value (see "Data Types").

External Target stage: Input page

The Input page allows you to specify details about how the External Target stage writes data to an external program. The External Target stage can have only one input link.

The General tab allows you to specify an optional description of the input link. The Properties tab allows you to specify details of exactly what the link does. The Partitioning tab allows you to specify how incoming data is partitioned before being written to the external program. The Format tab gives information about the format of the data being written. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about External Target stage properties, partitioning, and formatting are given in the following sections. See"Stage Editors," for a general description of the other tabs.

External Target stage: Input link properties tab

The Properties tab allows you to specify properties for the input link. These dictate how incoming data is written and to what program. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 17. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Target /Destination Program	string	N/A	Y if Source Method = Specific Program(s)	Y	N/A
Target /Destination Programs File	pathname	N/A	Y if Source Method = Program File(s)	Y	N/A
Target /Target Method	Specific Program(s)/ Program File(s)	Specific Program(s)	Y	N	N/A
Options/Reject Mode	Continue/Fail/ Save	Continue	N	N	N/A
Options/Schema File	pathname	N/A	N	N	N/A

External Target stage: Target category: Destination program

This is an optional property. Specifies the name of a program receiving data. InfoSphere DataStage calls the specified program and passes to it any arguments specified.You can repeat this property to specify multiple program instances with different arguments. You can use a job parameter to supply program name and arguments.

Destination programs file

This is an optional property. Specifies a file containing a list of program names and arguments. You can browse for the file or specify a job parameter. You can repeat this property to specify multiple files.

Target method

This property specifies whether you directly specifying a program (using the Destination Program property) or using a file to specify a program (using the Destination Programs File property).

External Target stage: Options category:
Reject mode

This is an optional property. Allows you to specify behavior if a record fails to be written for some reason. Choose from **Continue** to continue operation and discard any rejected rows, **Fail** to cease reading if any rows are rejected, or **Save** to send rejected rows down a reject link. Defaults to **Continue**.

Schema file

This is an optional property. By default the External Target stage will use the column definitions defined on the **Columns** tab as a schema for writing the file. You can, however, specify a file containing a schema instead (note, however, that if you have defined columns on the **Columns** tab, you should ensure these match the schema file). Type in a pathname or browse for a schema file.

External Target stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is written to the target program. It also allows you to specify that the data should be sorted before being written.

By default the stage will partition data in Auto mode.

If the External Target stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the External Target stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the External Target stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partitioning type** drop-down list. This will override any current partitioning.

If the External Target stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default Auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the External Target stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag columns.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.

- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default method for the External Target stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The **Partitioning** tab also allows you to specify that data arriving on the input link should be sorted before being written to the target program. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

External Target stage: Format tab

The Format tab allows you to supply information about the format of the data you are writing. The tab has a similar format to the Properties tab

If you do not alter any of the Format settings, the External Target stage will produce a file of the following format:

- Data comprises variable length columns contained within double quotes.
- All columns are delimited by a comma, except for the final column in a row.
- Rows are delimited by a UNIX newline.

You can use the **Format As** item from the shortcut menu in the Format tab to quickly change to a fixed-width column format, using DOS newlines as row delimiters, or producing a COBOL format file.

To change individual properties, select a property type from the main tree then add the properties you want to set to the tree structure by clicking on them in the **Available properties to set window**. You can then set a value for that property in the Property Value box. Pop up help for each of the available properties appears if you hover the mouse pointer over it.

This description uses the terms "record" and "row" and "field" and "column" interchangeably.

The following sections list the Property types and properties available for each type.

Record level

These properties define details about how data records are formatted in the flat file. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

Fill char

Specify an ASCII character or a value in the range 0 to 255. You can also choose Space or Null from a drop-down list. This character is used to fill any gaps in a written record caused by column positioning properties. Set to 0 by default (which is the NULL character). For example, to set it to space you could also type in the space character or enter 32. Note that this value is restricted to one byte, so you cannot specify a multi-byte Unicode character.

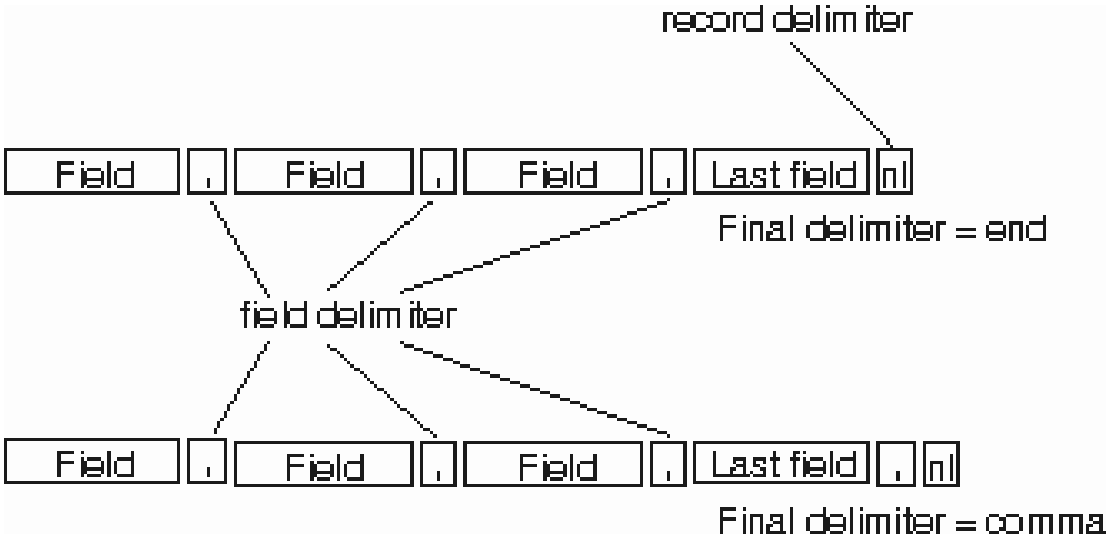
Final delimiter string

Specify a string to be written after the last column of a record in place of the column delimiter. Enter one or more characters, this precedes the record delimiter if one is used. Mutually exclusive with Final delimiter, which is the default. For example, if you set Delimiter to comma and Final delimiter string to `, ` (comma space - you do not need to enter the inverted commas) all fields are delimited by a comma, except the final field, which is delimited by a comma followed by an ASCII space character.

Final delimiter

Specify a single character to be written after the last column of a record in place of the field delimiter. Type a character or select one of whitespace, end, none, null, tab, or comma. See the following diagram for an illustration.

- **whitespace**. The last column of each record will not include any trailing white spaces found at the end of the record.
- **end**. The last column of each record does not include the field delimiter. This is the default setting.
- **none**. The last column of each record does not have a delimiter; used for fixed-width fields.
- **null**. The last column of each record is delimited by the ASCII null character.
- **comma**. The last column of each record is delimited by the ASCII comma character.
- **tab**. The last column of each record is delimited by the ASCII tab character.



When writing, a space is now inserted after every field except the last in the record. Previously, a space was inserted after every field including the last. (If you want to revert to the pre-release 7.5 behavior of inserting a space after the last field, set the APT_FINAL_DELIM_COMPATIBLE environment variable.

Intact

The intact property specifies an identifier of a partial schema. A partial schema specifies that only the column(s) named in the schema can be modified by the stage. All other columns in the row are passed through unmodified. The file containing the partial schema is specified in the **Schema File** property on the **Properties** tab. This property has a dependent property, Check intact, but this is not relevant to input links.

Record delimiter string

Specify a string to be written at the end of each record. Enter one or more characters. This is mutually exclusive with Record delimiter, which is the default, record type and record prefix.

Record delimiter

Specify a single character to be written at the end of each record. Type a character or select one of the following:

- UNIX Newline (the default)
- null

(To implement a DOS newline, use the Record delimiter string property set to "\R\N" or choose**Format as** > **DOS line terminator** from the shortcut menu.)

Note: Record delimiter is mutually exclusive with Record delimiter string, Record prefix, and Record type.

Record length

Select **Fixed** where fixed length fields are being written. InfoSphere DataStage calculates the appropriate length for the record. Alternatively specify the length of fixed records as number of bytes. This is not used by default (default files are comma-delimited). The record is padded to the specified length with either zeros or the fill character if one has been specified.

Record Prefix

Specifies that a variable-length record is prefixed by a 1-, 2-, or 4-byte length prefix. It is set to 1 by default. This is mutually exclusive with Record delimiter, which is the default, and record delimiter string and record type.

Record type

Specifies that data consists of variable-length blocked records (varying) or implicit records (implicit). If you choose the implicit property, data is written as a stream with no explicit record boundaries. The end of the record is inferred when all of the columns defined by the schema have been parsed. The varying property allows you to specify one of the following IBM blocked or spanned formats: **V**, **VB**, **VS**, **VBS**, or **VR**.

This property is mutually exclusive with Record length, Record delimiter, Record delimiter string, and Record prefix and by default is not used.

Field defaults

Defines default properties for columns written to the file or files. These are applied to all columns written, but can be overridden for individual columns from the **Columns** tab using the Edit Column Metadata dialog box. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Actual field length**. Specifies the number of bytes to fill with the Fill character when a field is identified as null. When InfoSphere DataStage identifies a null field, it will write a field of this length full of Fill characters. This is mutually exclusive with Null field value.
- **Delimiter**. Specifies the trailing delimiter of all fields in the record. Type an ASCII character or select one of whitespace, end, none, null, comma, or tab.
  - **whitespace**. Whitespace characters at the end of a column are ignored, that is, are not treated as part of the column.
  - **end**. The end of a field is taken as the delimiter, that is, there is no separate delimiter. This is not the same as a setting of `None' which is used for fields with fixed-width columns.
  - **none**. No delimiter (used for fixed-width).
  - **null**. ASCII Null character is used.
  - **comma**. ASCII comma character is used.
  - **tab**. ASCII tab character is used.
- **Delimiter string**. Specify a string to be written at the end of each field. Enter one or more characters. This is mutually exclusive with Delimiter, which is the default. For example, specifying `, ` (comma space - you do not need to enter the inverted commas) would have each field delimited by `, ` unless overridden for individual fields.
- **Null field length**. The length in bytes of a variable-length field that contains a null. When a variable-length field is written, InfoSphere DataStage writes a length value of null field length if the field contains a null. This property is mutually exclusive with null field value.
- **Null field value**. Specifies the value written to null field if the source is set to null. Can be a number, string, or C-type 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 **x**hh*, where each *h* is a hexadecimal digit 0 - F. You must use this form to encode non-printable byte values.

  This property is mutually exclusive with Null field length and Actual length. For a fixed width data representation, you can use Pad char (from the general section of Type defaults) to specify a repeated trailing character if the value you specify is shorter than the fixed width of the field.

  Null field value has a sub property named Null field value separator. This is intended for output data, and should be ignored on Format tabs belonging to input links.

- **Prefix bytes**. Specifies that each column in the data file is prefixed by 1, 2, or 4 bytes containing, as a binary value, either the column's length or the tag value for a tagged 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. InfoSphere DataStage inserts the prefix before each field.

  This property is mutually exclusive with the Delimiter, Quote, and Final Delimiter properties, which are used by default.
- **Print field**. This property is not relevant for input links.
- **Quote**. Specifies that variable length fields are enclosed in single quotes, double quotes, or another character or pair of characters. Choose **Single** or **Double**, or enter a character. This is set to double quotes by default.

  When writing, InfoSphere DataStage inserts the leading quote character, the data, and a trailing quote character. Quote characters are not counted as part of a field's length.
- **Vector prefix**. For fields that are variable length vectors, specifies a 1-, 2-, or 4-byte prefix containing the number of elements in the vector. You can override this default prefix for individual vectors.

  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 this property to indicate the prefix length. InfoSphere DataStage inserts the element count as a prefix of each variable-length vector field. By default, the prefix length is assumed to be one byte.

Type defaults

These are properties that apply to all columns of a specific data type unless specifically overridden at the column level. They are divided into a number of subgroups according to data type.

General

These properties apply to several data types (unless overridden at column level):

- **Byte order**. Specifies how multiple byte data types (except string and raw data types) are ordered. Choose from:
  - little-endian. The high byte is on the right.
  - big-endian. The high byte is on the left.
  - native-endian. As defined by the native format of the machine. This is the default.
- **Data Format**. Specifies the data representation format of a field. Applies to fields of all data types except string, ustring, and raw and to record, subrec or tagged fields containing at least one field that is neither string nor raw. Choose from:
  - binary
  - text (the default)

    A setting of binary has different meanings when applied to different data types:
  - For decimals, binary means packed.
  - For other numerical data types, binary means "not text".
  - For dates, binary is equivalent to specifying the julian property for the date field.
  - For time, binary is equivalent to 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. A binary timestamp specifies that two 32-but integers are written.

    By default data is formatted as text, as follows:
  - For the date data type, text specifies that the data to be written contains a text-based date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).

  - For the decimal data type: a 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. The destination string format is: [+ | -]ddd.[ddd] and any precision and scale arguments are ignored.
  - For *numeric* fields (int8, int16, int32, uint8, uint16, uint32, sfloat, and dfloat): InfoSphere DataStage assumes that numeric fields are represented as text.
  - For the time data type: text specifies that the field represents time in the text-based form %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
  - For the *timestamp* data type: text specifies a text-based timestamp in the form *%yyyy-%mm-%dd %hh:%nn:%ss* or in the default date format if you have defined a new one on an NLS system.
- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.
- **Field width**. The number of bytes in a field represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.

  If you specify neither field width nor field max width, numeric fields written 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)

  Important: If you are using Unicode character columns, you must calculate the field length in bytes and specify that value in the Field Width column property.
- **Pad char**. Specifies the pad character used when strings or numeric values are written to an external string representation. Enter a character (single-byte for strings, can be multi-byte for ustrings) or choose null or space. The pad character is used when the external string representation is larger than required to hold the written field. In this case, the external string is filled with the pad character to its full length. Space is the default. Applies to string, ustring, and numeric data types and record, subrec, or tagged types if they contain at least one field of this type.
- **Character set**. Specifies the character set. Choose from ASCII or EBCDIC. The default is ASCII. Applies to all data types except raw and ustring and record, subrec, or tagged containing no fields other than raw or ustring.

String

These properties are applied to columns with a string data type, unless overridden at column level.

- **Export EBCDIC as ASCII**. Select this to specify that EBCDIC characters are written as ASCII characters. Applies to fields of the string data type and record, subrec, or tagged fields if they contain at least one field of this type.
- **Import ASCII as EBCDIC**. Not relevant for input links.

Decimal

These properties are applied to columns with a decimal data type unless overridden at column level.

- **Allow all zeros**. Specifies whether to treat a packed decimal column containing all zeros (which is normally illegal) as a valid representation of zero. Select **Yes** or **No**. The default is **No**.
- **Decimal separator.** Specify the ASCII character that acts as the decimal separator (period by default).
- **Packed**. Select an option to specify what the decimal columns contain, choose from:
  - **Yes** to specify that the decimal columns contain data in packed decimal format (the default). This has the following sub-properties:

    Check. Select **Yes** to verify that data is packed, or **No** to not verify.

    Signed. Select **Yes** to use the existing sign when writing decimal columns. Select **No** to write a positive sign (0xf) regardless of the columns' actual sign value.
  - **No (separate)** to specify that they contain unpacked decimal with a separate sign byte. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - **No (zoned)** to specify that they contain an unpacked decimal in either ASCII or EBCDIC text. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - **No (overpunch)** to specify that the field has a leading or end 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 has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
- **Precision**. Specifies the precision where a decimal column is written in text format. Enter a number. When a decimal is written to a string representation, InfoSphere DataStage 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. When they are defined, InfoSphere DataStage truncates or pads the source decimal to fit the size of the destination string. If you have also specified the field width property, InfoSphere DataStage truncates or pads the source decimal to fit the size specified by field width.
- **Rounding**. Specifies how to round a decimal column when writing it. Choose from:
  - up (ceiling). Truncate source column towards positive infinity. This mode corresponds to the IEEE 754 Round Up mode. For example, 1.4 becomes 2, -1.6 becomes -1.
  - down (floor). Truncate source column towards negative infinity. This mode corresponds to the IEEE 754 Round Down mode. For example, 1.6 becomes 1, -1.4 becomes -2.
  - nearest value. Round the source column towards the nearest representable value. This mode corresponds to the COBOL ROUNDED mode. For example, 1.4 becomes 1, 1.5 becomes 2, -1.4 becomes -1, -1.5 becomes -2.
  - truncate towards zero. This is the default. Discard fractional digits to the right of the right-most fractional digit supported by 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. Using this method 1.6 becomes 1, -1.6 becomes -1.
- **Scale**. Specifies how to round a source decimal when its precision and scale are greater than those of the destination. By default, when the InfoSphere DataStage writes 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, InfoSphere DataStage truncates or pads the source decimal to fit the size of the destination string. If you have also specified the field width property, InfoSphere DataStage truncates or pads the source decimal to fit the size specified by field width.

Numeric

These properties apply to integer and float fields unless overridden at column level.

- **C_format**. Perform non-default conversion of data from integer or floating-point data to a string. This property specifies a C-language format string used for writing integer or floating point strings. This is passed to *sprintf*(). For example, specifying a C-format of %x and a field width of 8 ensures that integers are written as 8-byte hexadecimal strings.
- **In_format**. This property is not relevant for input links..
- **Out_format**. Format string used for conversion of data from integer or floating-point data to a string. This is passed to *sprintf*(). By default, InfoSphere 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 to pass formatting arguments to *sprintf*().

Date

These properties are applied to columns with a date data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Days since**. Dates are written as a signed integer containing the number of days since the specified date. Enter a date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
- **Format string**. The string format of a date. By default this is %yyyy-%mm-%dd. For details about the format, see "Date formats" on page 31.
- **Is Julian**. Select this to specify that dates are written as a numeric value containing the Julian day. A Julian day specifies the date as the number of days from 4713 BCE January 1, 12:00 hours (noon) GMT.

Time

These properties are applied to columns with a time data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Format string**. Specifies the format of columns representing time as a string. For details about the format, see "Time formats" on page 35
- **Is midnight seconds**. Select this to specify that times are written as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.

Timestamp

These properties are applied to columns with a timestamp data type unless overridden at column level.

- **Format string**. Specifies the format of a column representing a timestamp as a string. Defaults to *%yyyy-%mm-%dd %hh:%nn:%ss*. The format combines the format for date strings and time strings. See "Date formats" on page 31 and "Time formats" on page 35.

External Target stage: Output page: The Output page appears if the stage has a Reject link.

The General tab allows you to specify an optional description of the output link.

You cannot change the properties of a Reject link. The Properties tab for a reject link is blank.

Similarly, you cannot edit the column definitions for a reject link. The link uses the column definitions for the link rejecting the data records.

Using RCP with External Target stages

Runtime column propagation (RCP) allows InfoSphere DataStage to be flexible about the columns you define in a job. If RCP is enabled for a project, you can just define the columns you are interested in using in a job, but ask InfoSphere DataStage to propagate the other columns through the various stages. So such columns can be extracted from the data source and end up on your data target without explicitly being operated on in between.

External Target stages, unlike most other data targets, do not have inherent column definitions, and so InfoSphere DataStage cannot always tell where there are extra columns that need propagating. You can only use RCP on External Target stages if you have used the Schema File property (see "Schema File") to specify a schema which describes all the columns in the sequential files referenced by the stage. You need to specify the same schema file for any similar stages in the job where you want to propagate columns. Stages that will require a schema file are:

- Sequential File
- File Set
- External Source
- External Target
- Column Import
- Column Export

Complex Flat File stage

The Complex Flat File (CFF) stage is a file stage. You can use the stage to read a file or write to a file, but you cannot use the same stage to do both.

As a source, the CFF stage can have multiple output links and a single reject link. You can read data from one or more complex flat files, including MVS™ data sets with QSAM and VSAM files. You can also read data from files that contain multiple record types. The source data can contain one or more of the following clauses:

- GROUP
- REDEFINES
- OCCURS
- OCCURS DEPENDING ON

CFF source stages run in parallel mode when they are used to read multiple files, but you can configure the stage to run sequentially if it is reading only one file with a single reader.

As a target, the CFF stage can have a single input link and a single reject link. You can write data to one or more complex flat files. You cannot write to MVS data sets or to files that contain multiple record types.

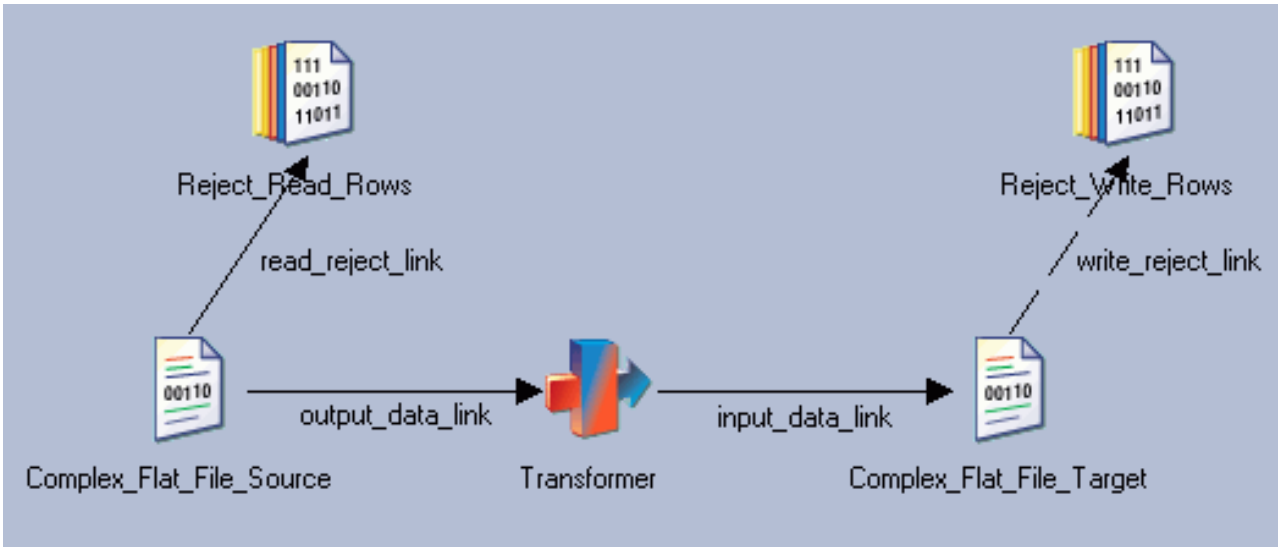


Figure 5. This job has a Complex Flat File source stage with a single reject link, and a Complex Flat File target stage with a single reject link.

Editing a Complex Flat File stage as a source

To edit a CFF stage as a source, you must provide details about the file that the stage will read, create record definitions for the data, define the column metadata, specify record ID constraints, and select output columns.

Procedure

1. Open the CFF stage editor.
2. On the Stage page, specify information about the stage data:
   a. On the File Options tab, provide details about the file that the stage will read.
   b. On the Record Options tab, describe the format of the data in the file.
   c. If the stage is reading a file that contains multiple record types, on the Records tab, create record definitions for the data.
   d. On the Records tab, create or load column definitions for the data.
   e. If the stage is reading a file that contains multiple record types, on the Records ID tab, define the record ID constraint for each record.
   f. Optional: On the Advanced tab, change the processing settings.
3. On the Output page, specify how to read data from the source file:
   a. On the Selection tab, select one or more columns for each output link.
   b. Optional: On the Constraint tab, define a constraint to filter the rows on each output link.
   c. Optional: On the Advanced tab, change the buffering settings.
4. Click **OK** to save your changes and to close the CFF stage editor.

Creating record definitions

If you are reading data from a file that contains multiple record types, you must create a separate record definition for each type.

Procedure

1. Click the **Records** tab on the Stage page.
2. Clear the **Single record** check box.

3. Right-click the default record definition RECORD_1 and select **Rename Current Record**.
4. Type a new name for the default record definition.
5. Add another record by clicking one of the buttons at the bottom of the records list. Each button offers a different insertion point. A new record is created with the default name of NEWRECORD.
6. Double-click NEWRECORD to rename it.
7. Repeat steps 3 and 4 for each new record that you need to create.
8. Right-click the master record in the list and select **Toggle Master Record**. Only one master record is permitted.

Column definitions

You must define columns to specify what data the CFF stage will read or write.

If the stage will read data from a file that contains multiple record types, you must first create record definitions on the Records tab. If the source file contains only one record type, or if the stage will write data to a target file, then the columns belong to the default record called RECORD_1.

You can load column definitions from a table in the repository, or you can type column definitions into the columns grid. You can also define columns by dragging a table definition from the Repository window to the CFF stage icon on the Designer canvas. If you drag a table to a CFF source stage that has multiple record types, the columns are added to the first record in the stage. You can then propagate the columns to one or more output links by right-clicking the stage icon and selecting **Propagate Columns** from the pop-up menu.

The columns that you define must reflect the actual layout of the file format. If you do not want to display all of the columns, you can create fillers to replace the unwanted columns by selecting the **Create fillers** check box in the Select Columns From Table window. For more information about fillers, see "Filler creation and expansion" on page 167.

After you define columns, the CFF stage projects the columns to the Selection tab on the Output page if you are using the stage as a source, or to the Columns tab on the Input page if you are using the stage as a target.

Loading columns:

The fastest way to define column metadata is to load columns from a table definition in the repository.

About this task

If you load more than one table definition, the list of columns from the subsequent tables is added to the end of the current list. If the first column of the subsequent list has a level number higher than the last column of the current list, the CFF stage inserts a 02 FILLER group item before the subsequent list is loaded. However, if the first column that is being loaded already has a level number of 02, then no 02 FILLER group item is inserted.

Procedure

1. Click the **Records** tab on the Stage page.
2. Click **Load** to open the Table Definitions window. This window displays all of the repository objects that are in the current project.
3. Select a table definition in the repository tree and click **OK**.
4. Select the columns to load in the Select Columns From Table window and click **OK**.
5. If flattening is an option for any arrays in the column structure, specify how to handle array data in the Complex File Load Option window. See "Complex file load options" on page 167 for details.

Typing columns:

You can also define column metadata by typing column definitions in the columns grid.

Procedure

1. Click the **Records** tab on the Stage page.
2. In the **Level number** field of the grid, specify the COBOL level number where the data is defined. If you do not specify a level number, a default value of 05 is used.
3. In the **Column name** field, type the name of the column.
4. In the **Native type** field, select the native data type.
5. In the **Length** field, specify the data precision.
6. In the **Scale** field, specify the data scale factor.
7. Optional: In the **Description** field, type a description of the column.

Results

You can edit column properties by selecting a property in the properties tree and making changes in the **Value** field. Use the **Available properties to add** field to add optional attributes to the properties tree.

If you need to specify COBOL attributes, open the Edit Column Meta Data window by right-clicking anywhere in the columns grid and selecting **Edit row** from the pop-up menu.

Filler creation and expansion:

When you load columns into a CFF stage, you can create fillers to save storage space and processing time.

Mainframe table definitions frequently contain hundreds of columns. If you do not want to display all of these columns in the CFF stage, you can create fillers. When you load columns into the stage, select the **Create fillers** check box in the Select Columns From Table window. This check box is selected by default and is available only when you load columns from a simple or complex flat file.

The sequences of unselected columns are collapsed into FILLER items with a storage length that is equal to the sum of the storage length of each individual column that is being replaced. The native data type is set to CHARACTER, and the name is set to FILLER_XX_YY, where XX is the start offset and YY is the end offset. Fillers for elements of a group array or an OCCURS DEPENDING ON (ODO) column have the name of FILLER_NN, where NN is the element number. The NN begins at 1 for the first unselected group element and continues sequentially. Any fillers that follow an ODO column are also numbered sequentially.

You can expand fillers on the Records tab if you want to reselect any columns. Right-click a filler either in the columns grid or the columns tree and select **Expand filler** from the pop-up menu. In the Expand Filler window, you can select some or all of the columns from the given filler. You do not need to reload the table definition and reselect the columns.

See Appendix C, “Fillers,” on page 681 for examples of how fillers are created for different COBOL structures.

Complex file load options:

If you define columns that contain arrays, you must specify how the CFF stage should process the array data.

The Complex File Load Option window opens when you load or type column definitions that contain arrays on the Records tab. This window displays the names of the column definitions and their structure. Array sizes are shown in parentheses. Select one of the following options to specify how the stage should process arrays:

- **Flatten selective arrays**. Specifies that arrays can be selected for flattening on an individual basis. This option is the default. Right-click an array in the columns list and select **Flatten** from the pop-up menu. Columns that cannot be flattened are not available for selection. The array icon changes for the arrays that will be flattened.
- **Flatten all arrays**. Flattens all arrays and creates a column for each element of the arrays.
- **As is**. Passes arrays with no changes.

If you choose to pass an array as is, the columns with arrays are loaded as is.

If you choose to flatten an array, all elements of the array will appear as separate columns in the table definition. The data is presented as one row at run time. Each array element is given a numeric suffix to make its name unique.

Consider the following complex flat file structure (in COBOL file definition format):

```
05    ID          PIC X(10)
05    NAME        PIC X(30)
05    CHILD       PIC X(30) OCCURS 5 TIMES
```

Flattening the array results in the following column definitions:

```
05    ID          PIC X(10)
05    NAME        PIC X(30)
05    CHILD       PIC X(30)
05    CHILD_2     PIC X(30)
05    CHILD_3     PIC X(30)
05    CHILD_4     PIC X(30)
05    CHILD_5     PIC X(30)
```

A parallel array is flattened in the same way.

Array columns that have redefined fields or OCCURS DEPENDING ON clauses cannot be flattened. Even if you choose to flatten all arrays in the Complex File Load Option window, these columns are passed as is.

Defining record ID constraints

If you are using the CFF stage to read data from a file that contains multiple record types, you must specify a record ID constraint to identify the format of each record.

About this task

Columns that are identified in the record ID clause must be in the same physical storage location across records. The constraint must be a simple equality expression, where a column equals a value.

Procedure

1. Click the **Records ID** tab on the Stage page.
2. Select a record from the **Records** list.
3. Select the record ID column from the **Column** list. This list displays all columns from the selected record, except the first OCCURS DEPENDING ON (ODO) column and any columns that follow it.
4. Select the = operator from the **Op** list.
5. Type the identifying value for the record ID column in the **Value** field. Character values must be enclosed in single quotation marks.

Selecting output columns

By selecting output columns, you specify which columns from the source file the CFF stage should pass to the output links.

About this task

You can select columns from multiple record types to output from the stage. If you do not select columns to output on each link, the CFF stage automatically propagates all of the stage columns except group columns to each empty output link when you click **OK** to exit the stage.

Procedure

1. Click the **Selection** tab on the Output page.
2. If you have multiple output links, select the link that you want from the **Output name** list.
3. Select columns for the output link by using one of the following methods:

Option	Description
To select individual columns	Press Ctrl and click the columns in the **Available columns** tree, then click **>** to move them to the **Selected columns** list.
To select all columns from a single record definition	Click the record name or any of its columns in the **Available columns** tree and click **>>** to move them to the **Selected columns** list. By default, group columns are not included, unless you first select the **Enable all group column selection** check box.

If you select columns out of order, they are reordered in the **Selected columns** list to match the structure of the input columns.

Array columns:

If you select an array column for output, the CFF stage passes data to the output link data in different ways depending on the type of array you select.

When you load array columns into a CFF stage, you must specify how to process the array data. You can pass the data as is, flatten all arrays on input to the stage, or flatten selected arrays on input. You choose one of these options from the Complex File Load Option window, which opens when you load column definitions on the Records tab.

If you choose to flatten arrays, the flattening is done at the time that the column metadata is loaded into the stage. All of the array elements appear as separate columns in the table. Each array column has a numeric suffix to make its name unique. You can select any or all of these columns for output.

If you choose to pass arrays as is, the array structure is preserved. The data is presented as a single row at run time for each incoming row. If the array is normalized, the incoming single row is resolved into multiple output rows.

The following examples show how the CFF stage passes data to the output link from different types of array columns, including:

- "Simple normalized array columns" on page 170
- "Nested normalized array columns" on page 170
- "Parallel normalized array columns" on page 170
- "Nested parallel denormalized array columns" on page 171

Simple normalized array columns

A simple array is a single, one-dimensional array. This example shows the result when you select all columns as output columns. For each record that is read from the input file, five rows are written to the output link. The sixth row out the link causes the second record to be read from the file, starting the process over again.

Input record:

```
05   ID              PIC X(10)
05   NAME            PIC X(30)
05   CHILD           PIC X(30) OCCURS 5 TIMES.
```

Output rows:

```
Row 1:     ID     NAME     CHILD(1)
Row 2:     ID     NAME     CHILD(2)
Row 3:     ID     NAME     CHILD(3)
Row 4:     ID     NAME     CHILD(4)
Row 5:     ID     NAME     CHILD(5)
```

Nested normalized array columns

This example shows the result when you select a nested array column as output. If you select FIELD-A, FIELD-C and FIELD-D as output columns, the CFF stage multiplies the OCCURS values at each level. In this case, 6 rows are written to the output link.

Input record:

```
05   FIELD-A          PIC X(4)
05   FIELD-B          OCCURS 2 TIMES.
     10   FIELD-C     PIC X(4)
     10   FIELD-D     PIC X(4) OCCURS 3 TIMES.
```

Output rows:

```
Row 1:     FIELD-A     FIELD-C(1)     FIELD-D(1,1)
Row 2:     FIELD-A     FIELD-C(1)     FIELD-D(1,2)
Row 3:     FIELD-A     FIELD-C(1)     FIELD-D(1,3)
Row 4:     FIELD-A     FIELD-C(2)     FIELD-D(2,1)
Row 5:     FIELD-A     FIELD-C(2)     FIELD-D(2,2)
Row 6:     FIELD-A     FIELD-C(2)     FIELD-D(2,3)
```

Parallel normalized array columns

Parallel arrays are array columns that have the same level number. The first example shows the result when you select all parallel array columns as output columns. The CFF stage determines the number of output rows by using the largest subscript. As a result, the smallest array is padded with default values and the element columns are repeated. In this case, if you select all of the input fields as output columns, four rows are written to the output link.

Input record:

```
05   FIELD-A          PIC X(4)
05   FIELD-B          PIC X(4) OCCURS 2 TIMES.
05   FIELD-C          PIC X(4)
05   FIELD-D          PIC X(4) OCCURS 3 TIMES.
05   FIELD-E          PIC X(4) OCCURS 4 TIMES.
```

Output rows:

```
Row 1:     FIELD-A     FIELD-B(1)     FIELD-C     FIELD-D(1)     FIELD-E(1)
Row 2:     FIELD-A     FIELD-B(2)     FIELD-C     FIELD-D(2)     FIELD-E(2)
Row 3:     FIELD-A                    FIELD-C     FIELD-D(3)     FIELD-E(3)
Row 4:     FIELD-A                    FIELD-C                    FIELD-E(4)
```

In the next example, only a subset of the parallel array columns are selected (FIELD-B and FIELD-E). FIELD-D is passed as is. The number of output rows is determined by the maximum size of the denormalized columns. In this case, four rows are written to the output link.

Output rows:

```
Row 1:     FIELD-A     FIELD-B(1)     FIELD-C     FIELD-D(1)     FIELD-D(2)     FIELD-D(3)     FIELD-E(1)
Row 2:     FIELD-A     FIELD-B(2)     FIELD-C     FIELD-D(1)     FIELD-D(2)     FIELD-D(3)     FIELD-E(2)
Row 3:     FIELD-A                    FIELD-C     FIELD-D(1)     FIELD-D(2)     FIELD-D(3)     FIELD-E(3)
Row 4:     FIELD-A                    FIELD-C     FIELD-D(1)     FIELD-D(2)     FIELD-D(3)     FIELD-E(4)
```

Nested parallel denormalized array columns

This complex example shows the result when you select both parallel array fields and nested array fields as output. If you select FIELD-A, FIELD-C, and FIELD-E as output columns in this example, the CFF stage determines the number of output rows by using the largest OCCURS value at each level and multiplying them. In this case, 3 is the largest OCCURS value at the outer (05) level, and 5 is the largest OCCURS value at the inner (10) level. Therefore, 15 rows are written to the output link. Some of the subscripts repeat. In particular, subscripts that are smaller than the largest OCCURS value at each level start over, including the second subscript of FIELD-C and the first subscript of FIELD-E.

```
05   FIELD-A          PIC X(10)
05   FIELD-B          OCCURS 3 TIMES.
     10   FIELD-C     PIC X(2) OCCURS 4 TIMES.
05   FIELD-D          OCCURS 2 TIMES.
     10   FIELD-E     PIC 9(3) OCCURS 5 TIMES.
```

Output rows:

```
Row 1:      FIELD-A     FIELD-C(1,1)     FIELD-E(1,1)
Row 2:      FIELD-A     FIELD-C(1,2)     FIELD-E(1,2)
Row 3:      FIELD-A     FIELD-C(1,3)     FIELD-E(1,3)
Row 4:      FIELD-A     FIELD-C(1,4)     FIELD-E(1,4)
Row 5:      FIELD-A                      FIELD-E(1,5)
Row 6:      FIELD-A     FIELD-C(2,1)     FIELD-E(2,1)
Row 7:      FIELD-A     FIELD-C(2,2)     FIELD-E(2,2)
Row 8:      FIELD-A     FIELD-C(2,3)     FIELD-E(2,3)
Row 9:      FIELD-A     FIELD-C(2,4)     FIELD-E(2,4)
Row 10:     FIELD-A                      FIELD-E(2,5)
Row 11:     FIELD-A     FIELD-C(3,1)
Row 12:     FIELD-A     FIELD-C(3,2)
Row 13:     FIELD-A     FIELD-C(3,3)
Row 14:     FIELD-A     FIELD-C(3,4)
Row 15:     FIELD-A
```

Group columns:

If you select a group column for output, the CFF stage passes data to the output link data in different ways depending on your selection.

Group columns contain elements or subgroups. When you select groups or their elements for output, the CFF stage handles them in the following manner:

- If you select a group column with any of its elements, the group column and the selected element columns are passed as group and element columns.
- If you select only elements of the group and not the group column itself, the selected element columns are treated as individual columns. Even if the selected element columns are within multiple or nested groups, all element columns are treated as top-level columns in the selection list on the Selection tab.

- You cannot select a group column without any of its elements.

Defining output link constraints

By defining a output link constraint, you can filter the data on each output link from the CFF stage.

About this task

You can set the output link constraint to match the record ID constraint for each selected output record by clicking **Default** on the Constraint tab on the Output page. The **Default** button is available only when the constraint grid is empty. For more information about record ID constraints, see "Defining record ID constraints" on page 168.

Procedure

1. Click the **Constraint** tab on the Output page.
2. In the **(** field of the grid, select an opening parenthesis if needed. You can use parentheses to specify the order in evaluating a complex constraint expression.
3. In the **Column** field, select a column or job parameter. (Group columns cannot be used in constraint expressions and are not displayed.)
4. In the **Op** field, select an operator or a logical function.
5. In the **Column/Value** field, select a column or job parameter, or double-click in the cell to type a value. Enclose character values in single quotation marks.
6. In the **)** field, select a closing parenthesis if needed.
7. If you are building a complex expression, in the **Logical** field, select **AND** or **OR** to continue the expression in the next row.
8. Click **Verify**. If errors are found, you must either correct the expression, click **Clear All** to start over, or cancel. You cannot save an incorrect constraint.

Results

The order of operators in expressions is determined by SQL standards. After you verify a constraint, any redundant parentheses might be removed.

Editing a Complex Flat File stage as a target

To edit a CFF stage as a target, you must provide details about the file that the stage will write, define the record format of the data, and define the column metadata.

Procedure

1. Open the CFF stage editor.
2. On the Stage page, specify information about the stage data:
   a. On the File Options tab, provide details about the file that the stage will write.
   b. On the Record Options tab, describe the format of the data in the file.
   c. On the Records tab, create or load column definitions for the data. See "Column definitions" on page 166 for information about how to define columns.
   d. Optional: On the Advanced tab, change the processing settings.
3. Optional: On the Input page, specify how to write data to the target file:
   a. On the Advanced tab, change the buffering settings.
   b. On the Partitioning tab, change the partitioning settings.
4. Click **OK** to save your changes and to close the CFF stage editor.

Reject links

The CFF stage can have a single reject link, whether you use the stage as a source or a target.

For CFF source stages, reject links are supported only if the source file contains a single record type without any OCCURS DEPENDING ON (ODO) columns. For CFF target stages, reject links are supported only if the target file does not contain ODO columns.

You cannot change the selection properties of a reject link. The Selection tab for a reject link is blank.

You cannot edit the column definitions for a reject link. For writing files, the reject link uses the input link column definitions. For reading files, the reject link uses a single column named "rejected" that contains raw data for the columns that were rejected after reading because they did not match the schema.

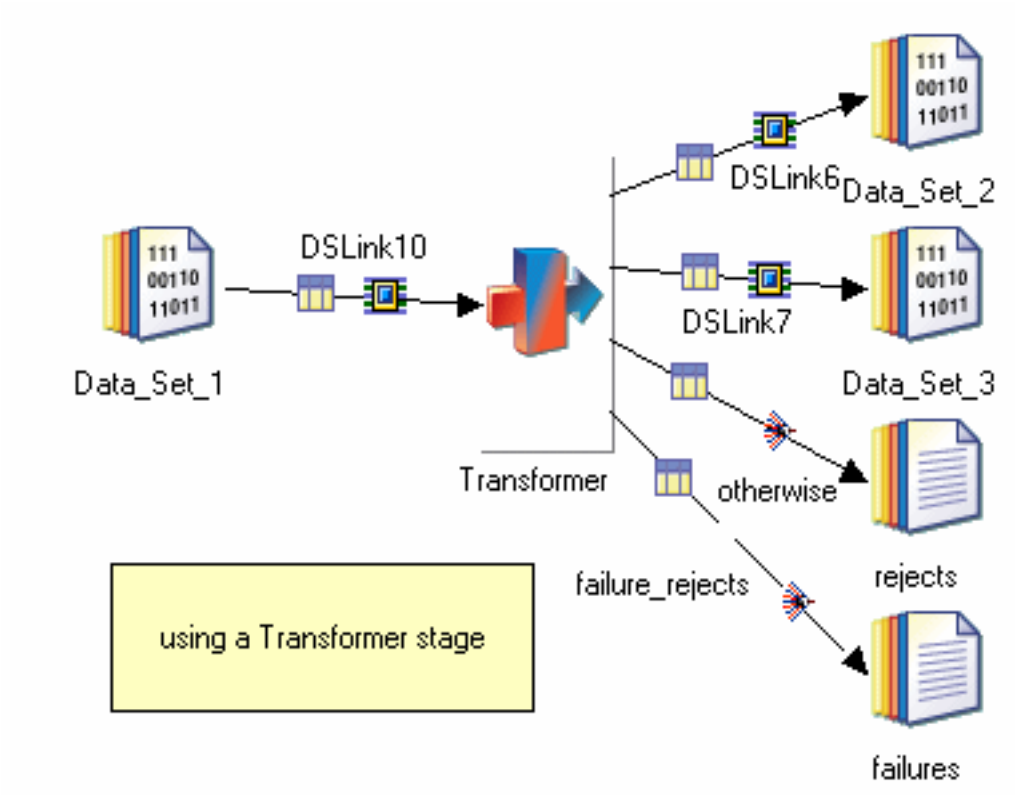
Chapter 6. Processing Data

Use the stage on the Processing section of the palette to manipulate data that you have read from a data source before writing it to a data target.

Transformer stage

The Transformer stage is a processing stage. It appears under the processing category in the tool palette. Transformer stages allow you to create transformations to apply to your data. These transformations can be simple or complex and can be applied to individual columns in your data. Transformations are specified using a set of functions. Details of available functions are given in Appendix B of IBM InfoSphere DataStage and QualityStage Parallel Job Developer's Guide.

Transformer stages can have a single input and any number of outputs. It can also have a reject link that takes any rows which have not been written to any of the outputs links by reason of a write failure or expression evaluation failure.



Unlike most of the other stages in a Parallel job, the Transformer stage has its own user interface. It does not use the generic interface as described in Chapter 4, “Stage editors,” on page 49.

When you edit a Transformer stage, the Transformer Editor appears. The left pane represents input data and the right pane, output data.

Transformer stage: fast path

About this task

This topic specifies the minimum steps to take to get a Transformer stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

- In the left pane:
  - Ensure that you have column meta data defined.
- In the right pane:
  - Ensure that you have column meta data defined for each of the output links. The easiest way to do this is to drag columns across from the input link.
  - Define the derivation for each of your output columns. You can leave this as a straight mapping from an input column, or explicitly define an expression to transform the data before it is output.
  - Optionally specify a constraint for each output link. This is an expression which input rows must satisfy before they are output on a link. Rows that are not output on any of the links can be output on the otherwise link.
  - Optionally specify one or more stage variables. This provides a method of defining expressions which can be reused in your output columns derivations (stage variables are only visible within the stage).

Transformer editor components

The Transformer Editor has the following components.

Transformer stage: Toolbar

The Transformer toolbar contains the following buttons (from left to right):

- Stage properties
- Constraints
- Show all
- Show/hide stage variables
- Cut
- Copy
- Paste
- Find/replace
- Load column definition
- Save column definition
- Column auto-match
- Input link execution order
- Output link execution order

Transformer stage: Link area

The top area displays links to and from the Transformer stage, showing their columns and the relationships between them.

The link area is where all column definitions and stage variables are defined.

The link area is divided into two panes; you can drag the splitter bar between them to resize the panes relative to one another. There is also a horizontal scroll bar, allowing you to scroll the view left or right.

The left pane shows the input link, the right pane shows output links. Output columns that have no derivation defined are shown in red.

Within the Transformer Editor, a single link might be selected at any one time. When selected, the link's title bar is highlighted, and arrowheads indicate any selected columns within that link.

Transformer stage: Metadata area

The bottom area shows the column metadata for input and output links. Again this area is divided into two panes: the left showing input link meta data and the right showing output link meta data.

The meta data for each link is shown in a grid contained within a tabbed page. Click the tab to bring the required link to the front. That link is also selected in the link area.

If you select a link in the link area, its meta data tab is brought to the front automatically.

You can edit the grids to change the column meta data on any of the links. You can also add and delete metadata.

As with column meta data grids on other stage editors, edit row in the context menu allows editing of the full metadata definitions (see "Columns Tab").

Transformer stage: Shortcut menus

The Transformer Editor shortcut menus are displayed by right-clicking the links in the links area.

There are slightly different menus, depending on whether you right-click an input link, an output link, or a stage variable. The input link menu offers you operations on input columns, the output link menu offers you operations on output columns and their derivations, and the stage variable menu offers you operations on stage variables.

The shortcut menu enables you to:

- Open the Stage Properties dialog box in order to specify stage or link properties.
- Open the Constraints dialog box to specify a constraint (only available for output links).
- Open the Column Auto Match dialog box.
- Display the Find/Replace dialog box.
- Display the Select dialog box.
- Edit, validate, or clear a derivation, or stage variable.
- Edit several derivations in one operation.
- Append a new column or stage variable to the selected link.
- Select all columns on a link.
- Insert or delete columns or stage variables.
- Cut, copy, and paste a column or a key expression or a derivation or stage variable.

If you display the menu from the links area background, you can:

- Open the Stage Properties dialog box in order to specify stage or link properties.
- Open the Constraints dialog box in order to specify a constraint for the selected output link.
- Open the Link Execution Order dialog box in order to specify the order in which links should be processed.
- Toggle between viewing link relations for all links, or for the selected link only.
- Toggle between displaying stage variables and hiding them.

Right-clicking in the meta data area of the Transformer Editor opens the standard grid editing shortcut menus.

Transformer stage basic concepts

When you first edit a Transformer stage, it is likely that you will have already defined what data is input to the stage on the input link. You will use the Transformer Editor to define the data that will be output by the stage and how it will be transformed. (You can define input data using the Transformer Editor if required.)

This section explains some of the basic concepts of using a Transformer stage.

Transformer stage: Input link

The input data source is joined to the Transformer stage via the input link.

Transformer stage: Output links

You can have any number of output links from your Transformer stage.

You might want to pass some data straight through the Transformer stage unaltered, but it's likely that you'll want to transform data from some input columns before outputting it from the Transformer stage.

You can specify such an operation by entering a transform expression. The source of an output link column is defined in that column's **Derivation** cell within the Transformer Editor. You can use the Expression Editor to enter expressions in this cell. You can also simply drag an input column to an output column's **Derivation** cell, to pass the data straight through the Transformer stage.

In addition to specifying derivation details for individual output columns, you can also specify constraints that operate on entire output links. A constraint is an expression that specifies criteria that data must meet before it can be passed to the output link. You can also specify a constraint otherwise link, which is an output link that carries all the data not output on other links, that is, columns that have not met the criteria.

Each output link is processed in turn. If the constraint expression evaluates to TRUE for an input row, the data row is output on that link. Conversely, if a constraint expression evaluates to FALSE for an input row, the data row is not output on that link.

Constraint expressions on different links are independent. If you have more than one output link, an input row might result in a data row being output from some, none, or all of the output links.

For example, if you consider the data that comes from a paint shop, it could include information about any number of different colors. If you want to separate the colors into different files, you would set up different constraints. You could output the information about green and blue paint on LinkA, red and yellow paint on LinkB, and black paint on LinkC.

When an input row contains information about yellow paint, the LinkA constraint expression evaluates to FALSE and the row is not output on LinkA. However, the input data does satisfy the constraint criterion for LinkB and the rows are output on LinkB.

If the input data contains information about white paint, this does not satisfy any constraint and the data row is not output on Links A, B or C, but will be output on the otherwise link. The otherwise link is used to route data to a table or file that is a "catch-all" for rows that are not output on any other link. The table or file containing these rows is represented by another stage in the job design.

You can also specify another output link which takes rows that have not be written to any other links because of write failure or expression evaluation failure. This is specified outside the stage by adding a

link and converting it to a reject link using the shortcut menu. This link is not shown in the Transformer meta data grid, and derives its meta data from the input link. Its column values are those in the input row that failed to be written.

If you have enabled Runtime Column Propagation for an output link (see "Output Page"), you do not have to specify meta data for that link.

Transformer stage: looping

You can use the looping mechanism in the Transformer stage to create multiple output rows from a single input row.

The following scenarios give examples of how you can use the looping facility.

Input data with multiple repeating columns

When the input data contains rows with multiple columns containing repeating data, you can use the Transformer stage to produce multiple output rows: one for each of the repeating columns.

For example, if the input row contained the following data.

Col1	Col2	Name1	Name2	Name3
abc	def	Jim	Bob	Tom

You can use the Transformer stage to flatten the input data and create multiple output rows for each input row. The data now comprises the following columns.

Col1	Col2	Name
abc	def	Jim
abc	def	Bob
abc	def	Tom

Input data with multiple repeating values in a single field

When you have data where a single column contains multiple repeating values that are separated by a delimiter, you can flatten the data to produce multiple output columns: one for each of the delimited values. You can also specify that certain values are filtered out, and not have a new row created.

For example, the input row contains the following data.

Col1	Col2	Names
abc	def	Jim/Bob/Tom

You want to flatten the name field so a new row is created for every new name indicated by the backslash (/) character. You also want to filter out the name Jim and drop the column named Col2, so that the resulting output data for the example column produces two rows with two columns.

Col1	Name
abc	Bob
abc	Tom

Value in an input row column used to generate new output rows

You can use the Transformer stage to generate new rows, based on values held in an input column.

For example, you have an input column that contains a count, and want to generate output rows based on the value of the count. The following example column has a count value of 5.

Col1	Col2	MaxCount
abc	def	5

You can generate five output rows for this one input row based on the value in the Count column.

Col1	Col2	EntryNumber
abc	def	1
abc	def	2
abc	def	3
abc	def	4
abc	def	5

Input row group aggregation included with input row data

You can save input rows to a cache area, so that you can process this data in a loop.

For example, you have input data that has a column holding a price value. You want to add a column to the output rows. The new column indicates what percentage the price value is of the total value for prices in all rows in that group. The value for the new Percentage column is calculated by the following expression.

```
(price * 100)/sum of all prices in group
```

In the example, the data is sorted and is grouped on the value in Col1.

Col1	Col2	Price
1000	abc	100.00
1000	def	20.00
1000	ghi	60.00
1000	jkl	20.00
2000	zyx	120.00
2000	wvu	110.00
2000	tsr	170.00

The percentage for each row in the group where Col1 = 1000 is calculated by the following expression.

```
(price * 100)/200
```

The percentage for each row in the group where Col1 = 2000 is calculated by the following expression.

```
(price * 100)/400
```

The output is shown in the following table.

Col1	Col2	Price	Percentage
1000	abc	100.00	50.00
1000	def	20.00	10.00
1000	ghi	60.00	30.00
1000	jkl	20.00	10.00

Col1	Col2	Price	Percentage
2000	zyx	120.00	30.00
2000	wvu	110.00	27.50
2000	tsr	170.00	42.50

This scenario uses key break facilities that are available on the Transformer stage. You can use these facilities to detect when the value of an input column changes, and so group rows as you process them.

Editing Transformer stages

About this task

The Transformer Editor enables you to perform the following operations on a Transformer stage:

- Create new columns on a link
- Delete columns from within a link
- Move columns within a link
- Edit column meta data
- Define output column derivations
- Define link constraints and handle otherwise links
- Specify the order in which links are processed
- Define local stage variables

Using drag-and-drop

Many of the Transformer stage edits can be made simpler by using the Transformer Editor's drag-and-drop functionality.

About this task

You can drag columns from any link to any other link. Common uses are:

- Copying input columns to output links
- Moving columns within a link
- Copying derivations in output links

Procedure

1. Click the source cell to select it.
2. Click the selected cell again and, without releasing the mouse button, drag the mouse pointer to the desired location within the target link. An insert point appears on the target link to indicate where the new cell will go.
3. Release the mouse button to drop the selected cell.

Results

You can drag multiple columns, key expressions, or derivations. Use the standard Explorer keys when selecting the source column cells, then proceed as for a single cell.

You can drag and drop the full column set by dragging the link title.

You can add a column to the end of an existing derivation by holding down the **Ctrl** key as you drag the column.

Find and replace facilities

About this task

If you are working on a complex job where several links, each containing several columns, go in and out of the Transformer stage, you can use the find/replace column facility to help locate a particular column or expression and change it.

The find/replace facility enables you to:

- Find and replace a column name
- Find and replace expression text
- Find the next empty expression
- Find the next expression that contains an error

To use the find/replace facilities, do one of the following:

- Click the **find/replace** button on the toolbar
- Choose **find/replace** from the link shortcut menu
- Type **Ctrl-F**

The Find and Replace dialog box appears. It has three tabs:

- **Expression Text**. Allows you to locate the occurrence of a particular string within an expression, and replace it if required. You can search up or down, and choose to match case, match whole words, or neither. You can also choose to replace all occurrences of the string within an expression.
- **Columns Names**. Allows you to find a particular column and rename it if required. You can search up or down, and choose to match case, match the whole word, or neither.
- **Expression Types**. Allows you to find the next empty expression or the next expression that contains an error. You can also press **Ctrl-M** to find the next empty expression or **Ctrl-N** to find the next erroneous expression.

  Note: The find and replace results are shown in the color specified in **Tools > Options**.

Press **F3** to repeat the last search you made without opening the Find and Replace dialog box.

Select facilities

If you are working on a complex job where several links, each containing several columns, go in and out of the Transformer stage, you can use the select column facility to select multiple columns. This facility is also available in the Mapping tabs of certain Parallel job stages.

About this task

The select facility enables you to:

- Select all columns/stage variables whose expressions contains text that matches the text specified.
- Select all column/stage variables whose name contains the text specified (and, optionally, matches a specified type).
- Select all columns/stage variable with a certain data type.
- Select all columns with missing or invalid expressions.

To use the select facilities, choose **Select** from the link shortcut menu. The Select dialog box appears. It has three tabs:

- **Expression Text**. This **Expression Text** tab allows you to select all columns/stage variables whose expressions contain text that matches the text specified. The text specified is a simple text match, taking into account the **Match case** setting.

- **Column Names**. The **Column Names** tab allows you to select all column/stage variables whose Name contains the text specified. There is an additional **Data Type** drop down list, that will limit the columns selected to those with that data type. You can use the **Data Type** drop down list on its own to select all columns of a certain data type. For example, all string columns can be selected by leaving the text field blank, and selecting String as the data type. The data types in the list are generic data types, where each of the column SQL data types belong to one of these generic types.
- **Expression Types**. The **Expression Types** tab allows you to select all columns with either empty expressions or invalid expressions.

Creating and deleting columns
About this task

You can create columns on links to the Transformer stage using any of the following methods:
- Select the link, then click the **load column definition** button in the toolbar to open the standard load columns dialog box.
- Use drag-and-drop or copy and paste functionality to create a new column by copying from an existing column on another link.
- Use the shortcut menus to create a new column definition.
- Edit the grids in the link's meta data tab to insert a new column.

When copying columns, a new column is created with the same meta data as the column it was copied from.

To delete a column from within the Transformer Editor, select the column you want to delete and click the **cut** button or choose **Delete Column** from the shortcut menu.

Moving columns within a link
About this task

You can move columns within a link using either drag-and-drop or cut and paste. Select the required column, then drag it to its new location, or cut it and paste it in its new location.

Editing column meta data
About this task

You can edit column meta data from within the grid in the bottom of the Transformer Editor. Select the tab for the link meta data that you want to edit, then use the standard InfoSphere DataStage edit grid controls.

The meta data shown does not include column derivations since these are edited in the links area.

Defining output column derivations
You can define the derivation of output columns from within the Transformer Editor in five ways.

About this task

Choose one of the following ways to define the derivation of output columns from within the Transformer Editor:
- If you require a new output column to be directly derived from an input column, with no transformations performed, then you can use drag-and-drop or copy and paste to copy an input column to an output link. The output columns will have the same names as the input columns from which they were derived.
- If the output column already exists, you can drag or copy an input column to the output column's **Derivation** field. This specifies that the column is directly derived from an input column, with no transformations performed.

- You can use the column auto-match facility to automatically set that output columns are derived from their matching input columns.
- You might need one output link column derivation to be the same as another output link column derivation. In this case you can use drag and drop or copy and paste to copy the derivation cell from one column to another.
- In many cases you will need to transform data before deriving an output column from it. For these purposes you can use the Expression Editor. To display the Expression Editor, double-click on the required output link column **Derivation** cell. (You can also invoke the Expression Editor using the shortcut menu or the shortcut keys.)

  Note: To access a vector element in a column derivation, you need to use an expression containing the vector function - see "Vector Function".

If a derivation is displayed in red (or the color defined in **Tools** > **Options**), it means that the Transformer Editor considers it incorrect.

Once an output link column has a derivation defined that contains any input link columns, then a relationship line is drawn between the input column and the output column, as shown in the following example. This is a simple example; there can be multiple relationship lines either in or out of columns. You can choose whether to view the relationships for all links, or just the relationships for the selected links, using the button in the toolbar.

Column auto-match facility:

You can specify to use the column auto-match facility, a feature that automatically set columns on an output link to be derived from matching columns on an input link.

About this task

This time-saving feature allows you to automatically set columns on an output link to be derived from matching columns on an input link. Using this feature you can fill in all the output link derivations to route data from corresponding input columns, then go back and edit individual output link columns where you want a different derivation.

Procedure

1. Do one of the following:
   - Click the **Auto-match** button in the Transformer Editor toolbar.
   - Choose **Auto-match** from the input link header or output link header shortcut menu.

   The Column Auto-Match dialog box appears.
2. Choose the output link that you want to match columns with the input link from the drop down list.
3. Click **Location match** or **Name match** from the **Match type** area.

   If you choose **Location match**, this will set output column derivations to the input link columns in the equivalent positions. It starts with the first input link column going to the first output link column, and works its way down until there are no more input columns left.
4. Click **OK** to proceed with the auto-matching.

   Note: Auto-matching does not take into account any data type incompatibility between matched columns; the derivations are set regardless.

Editing multiple derivations

About this task

You can make edits across several output column or stage variable derivations by choosing **Derivation Substitution...** from the shortcut menu. This opens the Expression Substitution dialog box.

The Expression Substitution dialog box allows you to make the same change to the expressions of all the currently selected columns within a link. For example, if you wanted to add a call to the trim() function around all the string output column expressions in a link, you could do this in two steps. First, use the Select dialog to select all the string output columns. Then use the Expression Substitution dialog to apply a trim() call around each of the existing expression values in those selected columns.

You are offered a choice between Whole expression substitution and Part of expression substitution.

Whole expression substitution:

With this option the whole existing expression for each column is replaced by the replacement value specified.

About this task

This replacement value can be a completely new value, but will usually be a value based on the original expression value. When specifying the replacement value, the existing value of the column's expression can be included in this new value by including "$1". This can be included any number of times.

For example, when adding a trim() call around each expression of the currently selected column set, having selected the required columns, you can use the following procedure.

Procedure

1. Select the **Whole expression** option.
2. Enter a replacement value of:
   `trim($1)`
3. Click **OK**

Results

Where a column's original expression was:
`DSLink3.col1`

This will be replaced by:
`trim(DSLink3.col1)`

This is applied to the expressions in each of the selected columns.

If you need to include the actual text $1 in your expression, enter it as "$$1".

Part of expression substitution:

With this option, only part of each selected expression is replaced rather than the whole expression. The part of the expression to be replaced is specified by a Regular Expression match.

About this task

It is possible that more that one part of an expression string could match the Regular Expression specified. If **Replace all occurrences** is checked, then each occurrence of a match will be updated with the replacement value specified. If it is not checked, then just the first occurrence is replaced.

When replacing part of an expression, the replacement value specified can include that part of the original expression being replaced. In order to do this, the Regular Expression specified must have round brackets around its value. "$1" in the replacement value will then represent that matched text. If the Regular Expression is not surrounded by round brackets, then "$1" will simply be the text "$1".

For complex Regular Expression usage, subsets of the Regular Expression text can be included in round brackets rather than the whole text. In this case, the entire matched part of the original expression is still replaced, but "$1", "$2" etc can be used to refer to each matched bracketed part of the Regular Expression specified.

The following is an example of the Part of expression replacement.

Suppose a selected set of columns have derivations that use input columns from `DSLink3'. For example, two of these derivations could be:

```
DSLink3.OrderCount + 1
If (DSLink3.Total > 0) Then DSLink3.Total Else -1
```

You might want to protect the usage of these input columns from null values, and use a zero value instead of the null. Use the following procedure to do this.

Procedure

1. Select the columns you want to substitute expressions for.
2. Select the **Part of expression** option.
3. Specify a Regular Expression value of:
   `(DSLink3\.[a-z,A-Z,0-9]*)`
4. Specify a replacement value of
   `NullToZero($1)`
5. Click **OK**, to apply this to all the selected column derivations.

Results

From the examples above:

```
DSLink3.OrderCount + 1
```

would become

```
NullToZero(DSLink3.OrderCount) + 1
```

and

```
If (DSLink3.Total > 0) Then DSLink3.Total Else -1
```

would become:

```
If (NullToZero(DSLink3.Total) > 0) Then DSLink3.Total Else -1
```

If the **Replace all occurrences** option is selected, the second expression will become:

```
If (NullToZero(DSLink3.Total) > 0)
Then NullToZero(DSLink3.Total)
Else -1
```

The replacement value can be any form of expression string. For example in the case above, the replacement value could have been:

```
(If (StageVar1 > 50000) Then $1 Else ($1 + 100))
```

In the first case above, the expression

```
DSLink3.OrderCount + 1
```

would become:

```
(If (StageVar1 > 50000) Then DSLink3.OrderCount
Else (DSLink3.OrderCount + 100)) + 1
```

Handling null values in input columns

If an input column that is used in the derivation expression of an output column contains a null value, then the resulting output column contains a null.

(In previous releases, if you used input columns in an output column expression, a null value in that input column caused the row to be dropped or, if a reject link had been defined, rejected. You had to specifically handle null values in the expression.)

Defining constraints and handling otherwise links

You can define a constraint to define limits for output data. You can also specify an otherwise link to catch rows that have failed to satisfy the constraints on all other output links.

About this task

You can define limits for output data by specifying a constraint. Constraints are expressions and you can specify a constraint for each output link from a Transformer stage. You can also specify that a particular link is to act as an otherwise link and catch those rows that have failed to satisfy the constraints on all other output links.

- Select an output link and click the **constraints** button.
- Double-click the output link's constraint entry field.
- Choose **Constraints** from the background or header shortcut menus.

A dialog box appears which allows you either to define constraints for any of the Transformer output links or to define a link as an otherwise link.

Define a constraint by entering an expression in the **Constraint** field for that link. Once you have done this, any constraints will appear below the link's title bar in the Transformer Editor. This constraint expression will then be checked against the row data at runtime. If the data does not satisfy the constraint, the row will not be written to that link. It is also possible to define a link which can be used to catch these rows which have been not satisfied the constraints on any previous links.

A constraint otherwise link can be defined by:

- Clicking on the **Otherwise/Log** field so a tick appears and leaving the **Constraint** fields blank. This will catch any rows that have failed to meet constraints on all the previous output links.
- Set the constraint to OTHERWISE. This will be set whenever a row is rejected on a link because the row fails to match a constraint. OTHERWISE is cleared by any output link that accepts the row.
- The otherwise link must occur *after* the output links in link order so it will catch rows that have failed to meet the constraints of all the output links. If it is not last rows might be sent down the otherwise link which satisfy a constraint on a later link and is sent down that link as well.
- Clicking on the **Otherwise/Log** field so a tick appears and defining a **Constraint**. This will result in the number of rows written to that link (that is, rows which satisfy the constraint) to be recorded in the job log as a warning message.

  Note: You can also specify a reject link which will catch rows that have not been written on any output links due to a write error or null expression error. Define this outside Transformer stage by adding a link and using the shortcut menu to convert it to a reject link.

Specifying link order

You can specify links to be in a particular order.

About this task

You can specify the order in which output links process a row.

The initial order of the links is the order in which they are added to the stage.

Procedure

1. Do one of the following:
   - Click the **output link execution order** button on the Transformer Editor toolbar.
   - Choose **output link reorder** from the background shortcut menu.

     The **Transformer Stage Properties** dialog box appears with the **Link Ordering** tab of the Stage page uppermost:.
2. Use the arrow buttons to rearrange the list of links in the execution order required.
3. When you are happy with the order, click **OK**.

Defining local stage variables

You can declare and use your own variables within a Transformer stage. Such variables are accessible only from the Transformer stage in which they are declared.

About this task

Stage variables can be used as follows:

- They can be assigned values by expressions.
- They can be used in expressions which define an output column derivation.
- Expressions evaluating a variable can include other variables or the variable being evaluated itself.

Any stage variables you declare are shown in a table in the right pane of the links area. The table looks similar to an output link. You can display or hide the table by clicking the **Stage Variable** button in the Transformer toolbar or choosing **Stage Variable** from the background shortcut menu.

Note: Stage variables are not shown in the output link meta data area at the bottom of the right pane.

The table lists the stage variables together with the expressions used to derive their values. Link lines join the stage variables with input columns used in the expressions. Links from the right side of the table link the variables to the output columns that use them.

Procedure

1. Do one of the following:
   - Select **Insert New Stage Variable** from the stage variable shortcut menu. A new variable is added to the stage variables table in the links pane. The variable is given the default name StageVar and default data type VarChar (255). You can edit these properties using the **Transformer Stage Properties** dialog box, as described in the next step.
   - Click the **Stage Properties** button on the Transformer toolbar.
   - Select **Stage Properties** from the background shortcut menu.
   - Select **Stage Variable Properties** from the stage variable shortcut menu.

     The **Transformer Stage Properties** dialog box appears.
2. Using the grid on the **Variables** page, enter the variable name, initial value, SQL type, extended information (if variable contains Unicode data), precision, scale, and an optional description. Variable names must begin with an alphabetic character (a-z, A-Z) and can only contain alphanumeric characters (a-z, A-Z, 0-9).
3. Click **OK**. The new variable appears in the stage variable table in the links pane.

Results

You perform most of the same operations on a stage variable as you can on an output column (see Defining Output Column Derivations). A shortcut menu offers the same commands. You cannot, however, paste a stage variable as a new column, or a column as a new stage variable.

Specifying loops

You can specify that the Transformer stage outputs multiple output rows for every input row by defining a loop condition.

About this task

You can specify that the Transformer stage repeats a loop multiple times for each input row that it reads. The stage can generate multiple output rows corresponding to a single input row while a particular condition is true.

You can use stage variables in the specification of this condition. Stage variables are evaluated once after each input row is read, and therefore hold the same value while the loop is executed.

You can also specify loop variables. Loop variables are evaluated each time that the loop condition is evaluated as true. Loop variables can change for each iteration of the loop, and for each output row that is generated.

The name of a variable must be unique across both stage variables and loop variables.

Procedure

1. Define any required stage variables.
2. Define a loop condition.
3. Define any required loop variables.
4. Define any output column derivations

Example

The examples show how each of the scenarios described in “Transformer stage: looping” on page 179 are implemented in a Transformer stage.

Defining a loop condition:

You specify that the Transformer stage loops when processing each input row by defining a loop condition. The loop continues to iterate while the condition is true.

About this task

You can use the @ITERATION system variable in your expression. @ITERATION holds a count of the number of times that the loop has been executed, starting at 1. @ITERATION is reset to one when a new input row is read.

Procedure

1. If required, open the Loop Condition grid by clicking the arrow on the title bar.
2. Double-click the Loop While condition, or type CTRL-D, to open the expression editor.
3. In the expression editor, specify the expression that controls your loop. The expression must return a result of true or false.

What to do next

It is possible to define a faulty loop condition that results in infinite looping, and yet still compiles successfully. To catch such events, you can specify a loop iteration warning threshold in the Loop Variable tab of the Stage Properties window. A warning is written to the job log when a loop has repeated the specified number of times, and the warning is repeated every time a multiple of that value is reached.

So, for example, if you specify a threshold of 100, warnings are written to the job log when the loop iterates 100 times, 200 times, 300 times, and so on. Setting the threshold to 0 specifies that no warnings are issued. The default threshold is 10000, which is a good starting value. You can set a limit for all jobs in your project by setting the environment variable APT_TRANSFORM_LOOP_WARNING_THRESHOLD to a threshold value.

The threshold applies to both loop iteration, and to the number of records held in the input row cache (the input row cache is used when aggregating values in input columns).

Defining loop variables:

You can declare and use loop variables within a Transformer stage. You can use the loop variables in expressions within the stage.

About this task

You can use loop variables when a loop condition is defined for the Transformer stage. When a loop is defined, the Transformer stage can output multiple rows for every row input to the stage. Loop variables are evaluated every time that the loop is iterated, and so can change their value for every output row. Such variables are accessible only from the Transformer stage in which they are declared. You cannot use a loop variable in a stage variable derivation.

Loop variables can be used as follows:

- They can be assigned values by expressions.
- They can be used in expressions which define an output column derivation.
- Expressions evaluating a variable can include other loop variables or stage variables or the variable being evaluated itself.

Any loop variables you declare are shown in a table in the right pane of the links area. The table looks like the output link table and the stage variables table. You can maximize or minimize the table by clicking the arrow in the table title bar.

The table lists the loop variables together with the expressions that are used to derive their values. Link lines join the loop variables with input columns used in the expressions. Links from the right side of the table link the variables to the output columns that use them, or to the stage variables that they use.

Procedure

1. Select **Loop Variable Properties** from the loop variable pop-up menu.
2. In the grid on the **Loop Variables** tab, enter the variable name, initial value, SQL type, extended information (if variable contains Unicode data), precision, scale, and an optional description. Variable names must begin with an alphabetic character (a-z, A-Z) and can only contain alphanumeric characters (a-z, A-Z, 0-9).
3. Click **OK**. The new loop variable appears in the loop variable table in the links pane.

   Note: You can also add a loop variable by selecting **Insert New Loop Variable** or **Append New Loop Variable** from the loop variable pop-up menu. A new variable is added to the loop variables table in the links pane. The first variable is given the default name LoopVar and default data type VarChar

(255), subsequent loop variables are named LoopVar1, LoopVar2, and so on. You can edit the variables on the **Loop Variables** tab of the Stage Properties window.

Example

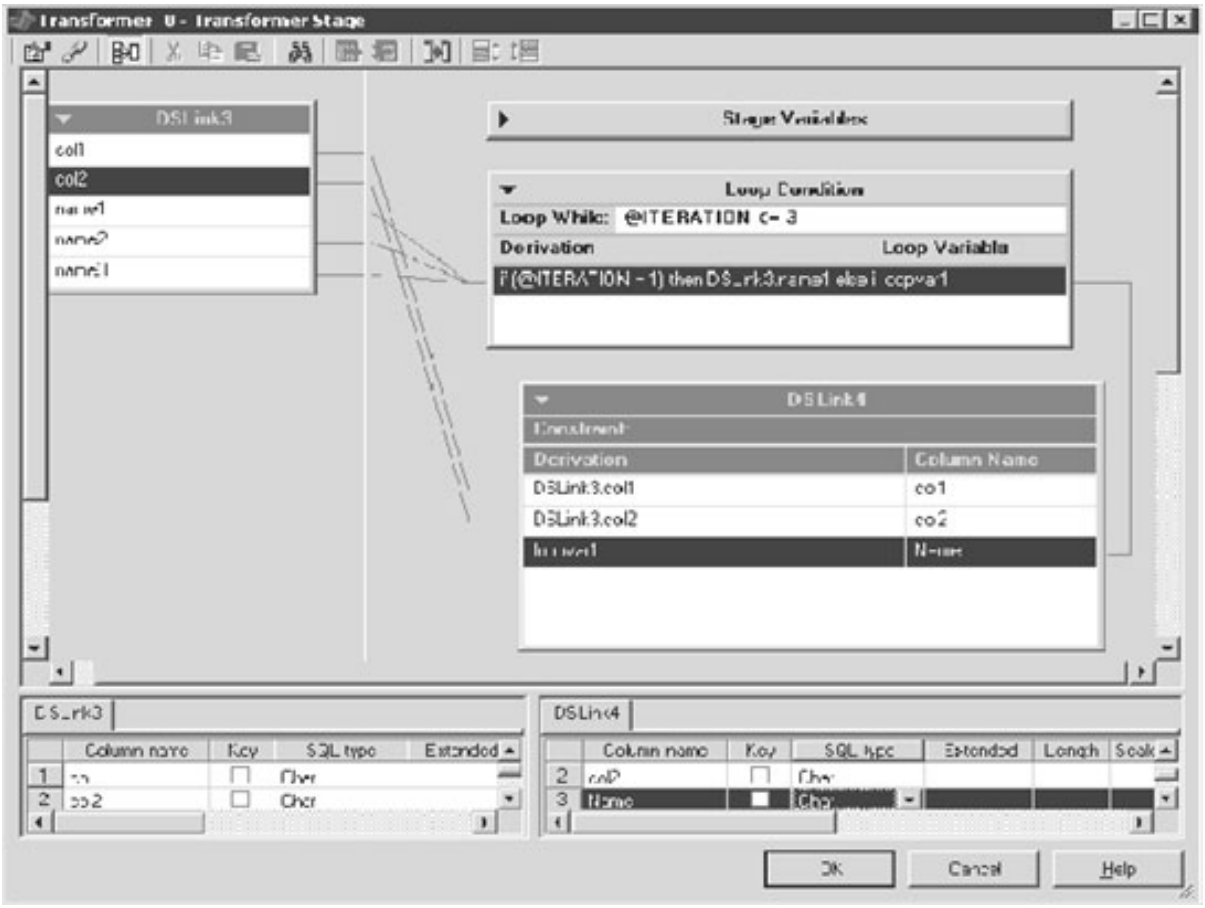


Figure 6. Example Transformer stage with loop variable defined

Loop example: converting a single row to multiple rows:

You can use the Transformer stage to convert a single row for data with repeating columns to multiple output rows.

Input data with multiple repeating columns

When the input data contains rows with multiple columns containing repeating data, you can use the Transformer stage to produce multiple output rows: one for each of the repeating columns.

For example, if the input row contained the following data.

Col1	Col2	Name1	Name2	Name3
abc	def	Jim	Bob	Tom

You can use the Transformer stage to flatten the input data and create multiple output rows for each input row. The data now comprises the following columns.

Col1	Col2	Name
abc	def	Jim
abc	def	Bob
abc	def	Tom

To implement this scenario in the Transformer stage, make the following settings:

Loop condition

Enter the following expression as the loop condition.

```
@ITERATION <= 3
```

Because each input row has three columns containing names, you need to process each input row three times and create three separate output rows.

Loop variable

Define a loop variable to supply the value for the new column Name in your output rows. The value of LoopVar1 is set by the following expression:

```
IF (@ITERATION = 1) THEN inlink.Name1
ELSE IF (@ITERATION = 2) THEN inlink.Name2
ELSE inlink.Name3
```

Output link metadata and derivations

Define the output link columns and their derivations:

- Col1 - inlink.col1
- Col2 - inlink.col2
- Name - LoopVar1

Loop example: multiple repeating values in a single field:

You can use the Transformer stage to convert a single row for data with repeating values in a single column to multiple output rows.

Input data with multiple repeating values in a single field

When you have data where a single column contains multiple repeating values that are separated by a delimiter, you can flatten the data to produce multiple output columns: one for each of the delimited values. You can also specify that certain values are filtered out, and not have a new row created.

For example, the input row contains the following data.

Col1	Col2	Names
abc	def	Jim/Bob/Tom

You want to flatten the name field so a new row is created for every new name indicated by the backslash (/) character. You also want to filter out the name Jim and drop the column named Col2, so that the resulting output data for the example column produces two rows with two columns.

Col1	Name
abc	Bob
abc	Tom

To implement this scenario in the Transformer stage, make the following settings:

Stage variable

Define a stage variable to hold a count of the fields separated by the delimiter character. The value of StageVar1 is set by the following expression:

```
DCOUNT(inlink.Names, "/")
```

Loop condition

Enter the following expression as the loop condition:

```
@ITERATION <= StageVar1
```

The loop continues to iterate for the count in the Names column.

Loop variable

Define a loop variable to supply the value for the new column Name in your output rows. The value of LoopVar1 is set by the following expression:

```
FIELD(inlink.Names, "/", @ITERATION, 1)
```

This expression extracts the substrings delimited by the slash character (/) from the input column.

Output link constraint

Define an output link constraint to filter out the name Jim. Use the following expression to define the constraint:

```
LoopVar1 <> "Jim"
```

Output link metadata and derivations

Define the output link columns and their derivations. Drop the Col2 column by not including it in the metadata.

- Col1 - inlink.col1
- Name - LoopVar1

Loop example: generating new rows:

You can use the Transformer stage to generate new rows, based on the value of a column in the input row.

Value in an input row column used to generate new output rows

You can use the Transformer stage to generate new rows, based on values held in an input column.

For example, you have an input column that contains a count, and want to generate output rows based on the value of the count. The following example column has a count value of 5.

Col1	Col2	MaxCount
abc	def	5

You can generate five output rows for this one input row based on the value in the Count column.

Col1	Col2	EntryNumber
abc	def	1
abc	def	2
abc	def	3
abc	def	4
abc	def	5

To implement this scenario in the Transformer stage, make the following settings:

Loop condition

Enter the following expression as the loop condition:

```
@ITERATION <= inlink.MaxCount
```

For each input row, the loop iterates the number of times defined by the value of the MaxCount column.

Output link metadata and derivations

Define the output link columns and their derivations:

- Col1 - inlink.Col1
- Col2 - inlink.Col2
- EntryNumber - @ITERATION

Loop example: aggregating data:

You can use the Transformer stage to add aggregated information to output rows.

Aggregation operations make use of a cache that stores input rows. You can monitor the number of entries in the cache by setting a threshold level in the Loop Variable tab of the Stage Properties window. If the threshold is reached when the job runs, a warning is issued into the log, and the job continues to run.

Input row group aggregation included with input row data

You can save input rows to a cache area, so that you can process this data in a loop.

For example, you have input data that has a column holding a price value. You want to add a column to the output rows. The new column indicates what percentage the price value is of the total value for prices in all rows in that group. The value for the new Percentage column is calculated by the following expression.

```
(price * 100)/sum of all prices in group
```

In the example, the data is sorted and is grouped on the value in Col1.

Col1	Col2	Price
1000	abc	100.00
1000	def	20.00
1000	ghi	60.00
1000	jkl	20.00
2000	zyx	120.00
2000	wvu	110.00
2000	tsr	170.00

The percentage for each row in the group where Col1 = 1000 is calculated by the following expression.

```
(price * 100)/200
```

The percentage for each row in the group where Col1 = 2000 is calculated by the following expression.

```
(price * 100)/400
```

The output is shown in the following table.

Col1	Col2	Price	Percentage
1000	abc	100.00	50.00

Col1	Col2	Price	Percentage
1000	def	20.00	10.00
1000	ghi	60.00	30.00
1000	jkl	20.00	10.00
2000	zyx	120.00	30.00
2000	wvu	110.00	27.50
2000	tsr	170.00	42.50

This scenario uses key break facilities that are available on the Transformer stage. You can use these facilities to detect when the value of an input column changes, and so group rows as you process them.

This scenario is implemented by storing the grouped rows in an input row cache and processing them when the value in a key column changes. In the example, the grouped rows are processed when the value in the column named Col1 changes from 1000 to 2000. Two functions, SaveInputRecord() and GetSavedInputRecord(), are used to add input rows to the cache and retrieve them. SaveInputRecord() is called when a stage variable is evaluated, and returns the count of rows in the cache (starting at 1 when the first row is added). GetSavedInputRecord() is called when a loop variable is evaluated.

To implement this scenario in the Transformer stage, make the following settings:

Stage variable

Define the following stage variables:

NumSavedRows

```
SaveInputRecord()
```

IsBreak

```
LastRowInGroup(inlink.Col1)
```

TotalPrice

```
IF IsBreak THEN SummingPrice + inlink.Price ELSE 0
```

SummingPrice

```
IF IsBreak THEN 0 ELSE SummingPrice + inlink.Price
```

NumRows

```
IF IsBreak THEN NumSavedRows ELSE 0
```

Loop condition

Enter the following expression as the loop condition:

```
@ITERATION <= NumRows
```

The loop continues to iterate for the count specified in the NumRows variable.

Loop variables

Define the following loop variable:

SavedRowIndex

```
GetSavedInputRecord()
```

Output link metadata and derivations

Define the output link columns and their derivations:

- Col1 - `inlink.Col1`
- Col2 - `inlink.Col2`
- Price - `inlink.Price`
- Percentage - `(inlink.Price * 100)/TotalPrice`

SaveInputRecord() is called in the first Stage Variable (NumSavedRows). SaveInputRecord() saves the current input row in the cache, and returns the count of records currently in the cache. Each input row in a group is saved until the break value is reached. At the last value in the group, NumRows is set to the number of rows stored in the input cache. The Loop Condition then loops round the number of times specified by NumRows, calling GetSavedInputRecord() each time to make the next saved input row current before re-processing each input row to create each output row. The usage of the inlink columns in the output link refers to their values in the currently retrieved input row, so will change on each output loop.

Caching selected input rows

You can call the SaveInputRecord() within an expression, so that input rows are only saved in the cache when the expression evaluates as true.

For example, you can implement the scenario described, but save only input rows where the price column is not 0. The settings are as follows:

Stage variable

Define the following stage variables:

IgnoreRow

```
IF (inlink.Price = 0) THEN 1 ELSE 0
```

NumSavedRows

```
IF IgnoreRecord THEN SavedRowSum ELSE SaveInputRecord()
```

IsBreak

```
LastRowInGroup(inlink.Col1)
```

SavedRowSum

```
 IF IsBreak THEN 0 ELSE NumSavedRows
```

TotalPrice

```
IF IsBreak THEN SummingPrice + inlink.Price ELSE 0
```

SummingPrice

```
IF IsBreak THEN 0 ELSE SummingPrice + inlink.Price
```

NumRows

```
IF IsBreak THEN NumSavedRows ELSE 0
```

Loop condition

Enter the following expression as the loop condition:

```
@ITERATION <= NumRows
```

Loop variables

Define the following loop variable:

SavedRowIndex

```
GetSavedInputRecord()
```

Output link metadata and derivations

Define the output link columns and their derivations:

- Col1 - `inlink.Col1`
- Col2 - `inlink.Col2`
- Price - `inlink.Price`
- Percentage - `(inlink.Price * 100)/TotalPrice`

This example produces output similar to the previous example, but the aggregation does not include Price values of 0, and no output rows with a Price value of 0 are produced.

Outputting additional generated rows

This example is based on the first example, but, in this case, you want to identify any input row where the Price is greater than or equal to 100. If an input row has a Price greater than or equal to 100, then a 25% discount is applied to the Price and a new additional output row is generated. The Col1 value in the new row has 1 added to it to indicate an extra discount entry. The original input row is still output as normal. Therefore any input row with a Price of greater than or equal to 100 will produce two output rows, one with the discounted price and one without.

The input data is as shown in the following table:

Col1	Col2	Price
1000	abc	100.00
1000	def	20.00
1000	ghi	60.00
1000	jkl	20.00
2000	zyx	120.00
2000	wvu	110.00
2000	tsr	170.00

The required table is shown in the following table:

Col1	Col2	Price	Percentage
1000	abc	100.00	50.00
1001	abc	75.00	50.00
1000	def	20.00	10.00
1000	ghi	60.00	30.00
1000	jkl	20.00	10.00
2000	zyx	120.00	30.00
2001	zyx	90.00	30.00
2000	wvu	110.00	27.50
2001	wvu	82.50	27.50
2000	tsr	170.00	42.50
2001	tsr	127.50	42.50

To implement this scenario in the Transformer stage, make the following settings:

Stage variable

Define the following stage variables:

NumSavedRowInt

```
SaveInputRecord()
```

AddRow

```
IF (inlink.Price >= 100) THEN 1 ELSE 0
```

NumSavedRows

```
IF AddRow THEN SaveInputRecord() ELSE NumSavedRowInt
```

IsBreak

```
LastRowInGroup(inlink.Col1)
```

TotalPrice

```
IF IsBreak THEN SummingPrice + inlink.Price ELSE 0
```

SummingPrice

```
IF IsBreak THEN 0 ELSE SummingPrice + inlink.Price
```

NumRows

```
IF IsBreak THEN NumSavedRows ELSE 0
```

Loop condition

Enter the following expression as the loop condition:

```
@ITERATION <= NumRows
```

The loop continues to iterate for the count specified in the NumRows variable.

Loop variables

Define the following loop variables:

SavedRowIndex

```
GetSavedInputRecord()
```

AddedRow

```
LastAddedRow
```

LastAddedRow

```
IF (inlink.Price < 100) THEN 0 ELSE IF (AddedRow = 0) THEN 1 ELSE 0
```

Output link metadata and derivations

Define the output link columns and their derivations:

- Col1 - `IF (inlink.Price < 100) THEN inlink.Col1 ELSE IF (AddedRow = 0) THEN inlink.Col1 ELSE inlink.Col1 + 1`
- Col2 - `inlink.Col2`
- Price - `IF (inlink.Price < 100) THEN inlink.Price ELSE IF (AddedRow = 0) THEN inlink.Price ELSE inlink.Price * 0.75`
- Percentage - `(inlink.Price * 100)/TotalPrice`

SaveInputRecord is called either once or twice depending on the value of Price. When SaveInputRecord is called twice, in addition to the normal aggregation, it produces the extra output record with the recalculated Price value. The Loop variable AddedRow is used to evaluate the output column values differently for each of the duplicate input rows.

Runtime errors

The number of calls to SaveInputRecord() and GetSavedInputRecord() must match for each loop. You can call SaveInputRecord() multiple times to add to the cache, but once you call GetSavedInputRecord(), then you must call it enough times to empty the input cache before you can call SaveInputRecord() again. The examples described can generate runtime errors in the following circumstances by not observing this rule:

- If your Transformer stage calls GetSavedInputRecord before SaveInputRecord, then a fatal error similar to the following example is reported in the job log:

  ```
  APT_CombinedOperatorController,0: Fatal Error: get_record() called on
  record 1 but only 0 records saved by save_record()
  ```

- If your Transformer stage calls GetSavedInputRecord more times than SaveInputRecord is called, then a fatal error similar to the following example is reported in the job log:

  ```
  APT_CombinedOperatorController,0: Fatal Error: get_record() called on
  record 3 but only 2 records saved by save_record()
  ```

- If your Transformer stage calls SaveInputRecord but does not call GetSavedInputRecord, then a fatal error similar to the following example is reported in the job log:

```
APT_CombinedOperatorController,0: Fatal Error: save_record() called on
record 3, but only 0 records retrieved by get_record()
```

- If your Transformer stage does not call GetSavedInputRecord as many times as SaveInputRecord, then a fatal error similar to the following example is reported in the job log:

```
APT_CombinedOperatorController,0: Fatal Error: save_record() called on
record 3, but only 2 records retrieved by get_record()
```

Key break detection

When your data is grouped on a column, and is sorted on this column, you can detect when you process the last row before the value in that column changes.

A key break can be detected on any column, as long as the input data has been sorted on that column. The data must be sorted in the job, so that the Transformer stage can detect that it is sorted. If the data is sorted before being input to the job, you can include a dummy Sort stage and set the stage's **Sort Key** property to **Don't Sort (previously sorted)** for every column on which the data was previously sorted.

You can use the following function to detect key breaks, and then process data accordingly:

`LastRowInGroup(`*`InputColumn`*`)`

You can call this function when processing an input row, and the function returns TRUE if the value in the named column changes after this row (that is, this row is the last row in a group). The function also returns TRUE if this row is the last row in the input data.

- If the input data is sorted by more than one column, then LastRowInGroup() can be called with any of those columns as its argument.
- If the argument specified is the primary sorted key, then the behavior is as described for a single column.
- If the argument specified is the secondary or other sorted key, then LastRowInGroup() returns TRUE if the value of the specified column is about to change, or if any of its higher level of key columns are about to change. For example, if the primary sorted column is Col1 and the secondary sorted column is Col2, then LastRowInGroup(Col2) returns true when either the value in Col2 is about to change, or the value in Col1 is about to change. Therefore it can return true when Col2 is not about to change, but Col1 is, because Col1 is a higher level sorted key than Col2.

Detection of end of data or end of wave

You can use a function to detect where your Transformer stage is processing the last input column in the input data, or in the wave.

You can use the following function to detect end of data or end of wave, and then process data accordingly:

`LastRow()`

You can call this function when processing an input row, and the function returns TRUE if the current row is the last row in the data, or the end of wave.

The InfoSphere DataStage expression editor

The InfoSphere DataStage Expression Editor helps you to enter correct expressions when you edit Transformer stages. The Expression Editor can:

- Facilitate the entry of expression elements
- Complete the names of frequently used variables
- Validate the expression

The Expression Editor can be opened from:

- Output link **Derivation** cells
- Stage variable **Derivation** cells
- Constraint dialog box

Expression format

The format of an expression is as follows:

```
KEY:

something_like_this                is a token
something_in_italics is a terminal, that is, does not break down any further
|          is a choice between tokens
[          is an optional part of the construction
"XXX"      is a literal token (that is, use XXX not
           including the quotes)
================================================
expression ::=    function_call |
      variable_name |
   other_name |
   constant |
   unary_expression |
   binary_expression |
   if_then_else_expression |
   substring_expression |
   "(" expression ")"

function_call ::= function_name "(" [argument_list] ")"
argument_list ::= expression | expression "," argument_list
function_name ::=    name of a built-in function |
   name of a user-defined_function
variable_name ::=       job_parameter name |
      stage_variable_name |
      link_variable name
other_name ::=    name of a built-in macro, system variable, and so on.
constant ::= numeric_constant | string_constant
numeric_constant ::= ["+" | "-"] digits ["." [digits]] ["E" | "e" ["+" | "-"] digits]
string_constant ::=    "'" [characters] "'" |
   """ [characters] """ |
   "\" [characters] "\"
unary_expression ::= unary_operator expression
unary_operator ::= "+" | "-"
binary_expression ::= expression binary_operator expression
binary_operator ::=    arithmetic_operator |
   concatenation_operator |
   matches_operator |
   relational_operator |
   logical_operator
arithmetic_operator ::= "+" | "-" | "*" | "/" | "^"
concatenation_operator ::= ":"
relational_operator ::=    "   =" |"EQ" |
      "<>" | "#" | "NE" |
      ">" | "GT" |
      ">=" | "=>" | "GE" |
      "<" | "LT" |
      "<=" | "=<" | "LE"
logical_operator ::= "AND" | "OR"
if_then_else_expression ::= "IF" expression "THEN" expression "ELSE" expression
substring_expression ::= expression "[" [expression ["," expression] "]"
field_expression ::=    expression "[" expression ","
   expression ","
   expression "]"
         /*    That is, always 3 args
```

Note: keywords like "AND" or "IF" or "EQ" might be in any case

Entering expressions
About this task

Whenever the insertion point is in an expression box, you can use the Expression Editor to suggest the next element in your expression. Do this by right-clicking the box, or by clicking the **Suggest** button to the right of the box. This opens the **Suggest Operand** or **Suggest Operator** menu. Which menu appears depends on context, that is, whether you should be entering an operand or an operator as the next expression element. The Functions available from this menu are described in Appendix B, "Parallel Transform functions," on page 645. The DS macros are described in *InfoSphere DataStage Parallel Job Advanced Developer Guide* You can also specify custom routines for use in the expression editor (see in *InfoSphere DataStage Designer Client Guide*).

Completing variable names
About this task

The Expression Editor stores variable names. When you enter a variable name you have used before, you can type the first few characters, then press **F5**. The Expression Editor completes the variable name for you.

If you enter the name of the input link followed by a period, for example, **DailySales.**, the Expression Editor displays a list of the column names of the link. If you continue typing, the list selection changes to match what you type. You can also select a column name using the mouse. Enter a selected column name into the expression by pressing **Tab** or **Enter**. Press **Esc** to dismiss the list without selecting a column name.

Validating the expression
About this task

When you have entered an expression in the Transformer Editor, press **Enter** to validate it. The Expression Editor checks that the syntax is correct and that any variable names used are acceptable to the compiler.

If there is an error, a message appears and the element causing the error is highlighted in the expression box. You can either correct the expression or close the Transformer Editor or Transform dialog box.

For any expression, selecting Validate from its shortcut menu will also validate it and show any errors in a message box.

Exiting the expression editor
There are a few ways in which you can exit the expression editor.

About this task

You can exit the Expression Editor in the following ways:

- Press **Esc** (which discards changes).
- Press **Return** (which accepts changes).
- Click outside the Expression Editor box (which accepts changes).

Configuring the expression editor
About this task

You can resize the Expression Editor window by dragging. The next time you open the expression editor in the same context (for example, editing output columns) on the same client, it will have the same size.

The Expression Editor is configured by editing the Designer options. This allows you to specify how `helpful' the expression editor is. For more information, see *InfoSphere DataStage Designer Client Guide*.

System variables

InfoSphere DataStage provides a set of variables containing useful system information that you can access from an output derivation or constraint.

Name Description

@FALSE
The value is replaced with 0.

@TRUE
The value is replaced with 1.

@INROWNUM
Input row counter.

@OUTROWNUM
Output row counter (per link).

@NUMPARTITIONS
The total number of partitions for the stage.

@PARTITIONNUM
The partition number for the particular instance.

@ITERATION
The iteration number of the current loop.

Evaluation sequences for transformer expressions, stage variables, and loop variables

To write efficient Transformer stage derivations, it helps to understand what items get evaluated and when.

The evaluation sequence for a Transformer stage is:

```
Evaluate each stage variable initial value
For each input row to process:
   Evaluate each stage variable derivation value, unless
   the derivation is empty
   For each output link:
      Evaluate constraint, if true:
         Evaluate each column derivation value
         Write the output row
   Next output link
Next input row
```

The evaluation sequence for a Transformer stage that has a loop condition defined is:

```
Evaluate each stage variable initial value
For each input row to process:
   Evaluate each stage variable derivation value (unless empty)
   Evaluate each loop variable initial value
   While the evaluated loop condition is true:
         Evaluate each loop variable derivation value (unless empty)
         For each output link:
            Evaluate constraint, if true:
               Evaluate each column derivation value
               Write the output row
         Next output link
   Loop back to While
Next input row
```

The stage variables, loop variables, and the columns within a link are evaluated in the order in which they are displayed on the parallel job canvas. Similarly, the output links are also evaluated in the order in which they are displayed.

Examples

Certain constructs are inefficient if they are included in output column derivations, because they are evaluated once for every output column that uses them. The following examples describe these constructs:

The same part of an expression is used in multiple column derivations.

For example, if you want to use the same substring of an input column in multiple columns in output links, you might use the following test in a number of output columns derivations:

```
IF (DSLINK1.col1[1,3] = "001") THEN ...
```

In this case, the evaluation of the substring of DSLINK1.col1[1,3] is repeated for each column that uses it. The evaluation can be made more efficient by moving the substring calculation into a stage variable. The substring is then evaluated once for every input row. This example has thus stage variable definition for StageVar1:

```
DSLINK1.col1[1,3]
```

Each column derivation starts with this test:

```
IF (StageVar1 = "001") THEN ...
```

This example can be improved further by also moving the string comparison into the stage variable. The stage variable is then defined as follows:

```
IF (DSLink1.col1[1,3] = "001") THEN 1 ELSE 0
```

Each column derivation starts with this test:

```
IF (StageVar1) THEN
```

The improved construct reduces both substring function evaluations and string comparisons.

An expression includes calculated constant values.

For example, a column definition might include a function call that returns a constant value:

```
Str(" ",20)
```

This function returns a string of 20 spaces. The function is evaluated every time the column derivation is evaluated. It is more efficient to calculate the constant value once. You can assign an initial value to a stage variable in the **Variables** tab of the Stage Properties window. The initial value is set to using this expression:

```
Str(" ", 20)
```

You do not supply the derivation of the stage variable on the main Transformer page. The initial value of the stage variable is evaluated once, before any input rows are processed. Because the derivation expression of the stage variable is empty, the stage variable is not re-evaluated for each input row. Change any expression that previously used the function `Str(" ", 20)` to use the stage variable instead.

The same considerations apply to any expression, or part of an expression, that generates a constant value. For example, the following expression concatenates two strings:

```
"abc" : "def"
```

The abcdef concatenation is repeated every time the column derivation is evaluated. Since the subpart of the expression is constant, this constant part of the expression can again be moved into a stage variable, using the initial value setting to perform the concatenation once.

An expression requiring a type conversion is used as a constant, or it is used in multiple places

For example, an expression might include the following code:

```
DSLink1.col1+"1"
```

In this example, the "1" is a string constant, and so must be converted from a string to an integer each time the expression is evaluated. The solution in this case is to change the constant from a string to an integer:

```
DSLink1.col1+1
```

If DSLINK1.col1 is a string field, however, then a conversion is required every time the expression is evaluated. If an input column is used in more than one expression, where it requires the same type conversion in each expression, it is more efficient to use a stage variable to perform the conversion once. You can create, for this example, an integer stage variable, specify its derivation to be DSLINK1.col1, and then use the stage variable in place of DSLink1.col1, where that conversion is required. When you use stage variables to evaluate parts of expressions, you must set the data type of the stage variable correctly for that context. Otherwise, needless conversions are required wherever that variable is used.

The advice on the use of stage variables equally applies to loop variables when you use the Transformer stage looping facilities. You add the evaluation to a stage variable if it is evaluated once per input row, or to a loop variable if it is evaluated once per looped output row when a loop condition was specified.

Transformer stage properties

The Transformer stage has a Properties dialog box which allows you to specify details about how the stage operates.

The Transform Stage Properties dialog box has three pages:

- **Stage Page**. This is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data input to the Transformer stage.
- **Output Page**. This is where you specify details about the output links from the Transformer stage.

Transformer stage: Stage page

The Stage page has up to nine tabs:

- **General**. Use this tab to enter an optional description of the stage.
- **Variables**. Use this tab to set up stage variables for use in the stage.
- **Loop Variables**. Use this tab to set up loop variables for use in the stage.
- **Surrogate Key**. Allows you to generate surrogate keys that can be used in complex lookup and update operations.
- **Advanced**. Use this tab to specify how the stage executes.
- **Link Ordering**. Use this tab to specify the order in which the output links will be processed.
- **Triggers**. Use this tab to run certain routines at certain points in the stage's execution.
- **NLS Locale**. Use this tab to select a locale other than the project default to determine collating rules.
- **Build**. Use this tab to override the default compiler and linker flags for this stag.

The Variables tab is described in "Defining Local Stage Variables". The Loop Variables tab is described in “Defining loop variables” on page 190. The Link Ordering tab is described in "Specifying Link Order".

Transformer stage: General tab:

Use the General page to provide a description of the stage, control the number of rejected row warnings, and modify the behavior of the Transformer when it encounters an unhandled null.

Whenever a row is rejected because it contains a null value, a warning is written to the job log. Potentially there could be a lot of messages, so you can use the **Maximum log reject messages** option to set limits. By default, up to 50 messages per partition are allowed, but you can increase or decrease this, or set it to -1 to allow unlimited messages.

Use the **Legacy null processing** option to return to pre-release 8.5 null handling. Before release 8.5, null values in input rows had to be explicitly handled by function calls if any processing was carried out on that row. Otherwise rows that contained nulls were dropped (this did not apply where the row was passed through the Transformer with no processing). If you do not select the **Legacy null processing** option, then a null in the input column used in the derivation causes a null to be output.

You can set the **Abort on unhandled null** option to force the job to stop when the transformer encounters a null value in a row. You can then locate the row and column that contained the null from the job log. If you set this option together with the **Legacy null processing** option, then any nulls that occur in input columns used in output column derivations that are not explicitly handled by the expression cause the job to stop. If you set the **Abort on unhandled null** option without setting the **Legacy null processing** option, then only operations such as attempting to set a non-nullable column to null cause the job to stop.

Transformer stage: Surrogate Key tab:

Use this tab to specify information about the key source if you generate surrogate keys. You can use surrogate keys in complex lookup and update operations. The key source can be a flat file or a database sequence.

You must create a derivation for the surrogate key column that uses the NextSurrogateKey function.

To generate surrogate keys by using a flat file:

1. In the **Source type** field, select **Flat File**.
2. In the **Source name** field, type the name and fully qualified path of the flat file, or click the arrow button to browse for the file or to insert a job parameter. The file must be accessible from all nodes that run the Transformer stage.
3. In the **Initial value** field, type a number or insert a job parameter. The first time the job runs, this value is used to initialize the key state. Every subsequent time the job runs, the initial value is the start value for generating surrogate keys. If the specified value is taken, the stage uses the next available value.
4. In the **New surrogate keys retrieved from state file** area, specify the block size to use for surrogate key retrieval:
   - Select **In blocks of** to set the block size manually, and type a number in the **key values** field.
   - Select **System selected block size** to have the system determine the optimal block size based on your job configuration. System-selected block sizes usually result in larger key ranges, which require less frequent state access and have better performance. However, the key sequence might have temporary gaps until the next time the job runs.

To generate surrogate keys by using a database sequence:

1. In the **Source type** field, select **DBSequence**.
2. In the **Source name** field, type the name of the database sequence, or click the arrow button to browse for the sequence or to insert a job parameter.
3. In the **Database type** field, select the type of the database.
4. In the **User name** field, type the user name for the database connection. This field is required for a remote database. If you leave this field blank and you have a local database, the stage uses the workstation login user name.

5. In the **Password** field, type the password for the database connection, This field is required for a remote database. If you leave this field blank and you have a local database, the stage uses the workstation login password.
6. In the **Server name** field, type the name of the database server.
7. If the database type is DB2, complete some additional steps:
   a. In the **Database name** field, type the name of the database. If you leave this field blank, the stage uses the name that is specified by the environment variable APT_DBNAME, or by the variable DB2DBDFT if APT_DBNAME is not set.
   b. In the **Client instance name** field, type the name of the DB2 client instance. This field is required for a remote database.
   c. In the **Client alias DB name** field, type the name of the client alias database on the remote server. This field is needed only if the alias database name is not the same as the server database name.

Transformer stage: Advanced tab:

The Advanced tab is the same as the Advanced tab of the generic stage editor as described in "Advanced Tab". This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In sequential mode the data is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is set to **Propagate** by default, this sets or clears the partitioning in accordance with what the previous stage has set. You can also select **Set** or **Clear**. If you select **Set**, the stage will request that the next stage preserves the partitioning as is.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Transformer stage: Triggers tab:

The Triggers tab allows you to choose routines to be executed at specific execution points as the transformer stage runs in a job. The execution point is per-instance, that is, if a job has two transformer stage instances running in parallel, the routine will be called twice, once for each instance.

The available execution points are Before-stage and After-stage. At this release, the only available built-in routine is SetCustomSummaryInfo. You can also define custom routines to be executed; to do this you define a C function, make it available in UNIX shared library, and then define a Parallel routine which calls it (see *InfoSphere DataStage Designer Client Guide* for details on defining a Parallel Routine). Note that the function should not return a value.

SetCustomSummaryInfo is used to collect reporting information. This information is included in any XML reports generated, and can be retrieved using a number of methods:

- DSMakeJobReport API function.
- The DSJob -Report command line command.
- DSJobReport used as an after-job subroutine.

Each item of information collected by SetCustomSummaryInfo is stored as a variable name, a description, and a value. These appear as arguments in the Triggers tab grid (variable name in Argument 1, description in Argument 2, value in Argument 3). You can supply values for them via the expression editor. You can use job parameters and stage variables but you cannot access data that is available only while the stage is running, such as columns.

Transformer stage: NLS Locale tab:

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The transformer stage uses this when it is evaluating expressions. For example, it would affect the evaluation of an expression such as "if ustring1 > ustring2". Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Transformer stage: Build tab:

This tab allows you to override the compiler and linker flags that have been set for the job or project. The flags you specify here will take effect for this stage and this stage alone. The flags available are platform and compiler-dependent.

Transformer stage: Input page

The Input page allows you to specify details about data coming into the Transformer stage. The Transformer stage can have only one input link.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned. This is the same as the Partitioning tab in the generic stage editor described in "Partitioning Tab". The Advanced tab allows you to change the default buffering settings for the input link.

Transformer stage: Partitioning tab:

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected when input to the Transformer stage. It also allows you to specify that the data should be sorted on input.

By default the Transformer stage will attempt to preserve partitioning of incoming data, or use its own partitioning method according to what the previous stage in the job dictates.

If the Transformer stage is operating in sequential mode, it will first collect the data before writing it to the file using the default collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Transformer stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partitioning type** drop-down list. This will override any current partitioning.

If the Transformer stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Transformer stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default method for the Transformer stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning method chosen.

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Transformer stage: Output page

The Output Page has a General tab which allows you to enter an optional description for each of the output links on the Transformer stage. It also allows you to switch Runtime column propagation on for this link, in which case data will automatically be propagated from the input link without you having to

specify meta data for this output link (see "Runtime Column Propagation"). The Advanced tab allows you to change the default buffering settings for the output links.

BASIC Transformer stage

The BASIC Transformer stage is a processing stage. It appears under the processing category in the tool palette in the Transformer shortcut container.

The BASIC Transformer stage is similar in appearance and function to the Transformer stage described in “Transformer stage” on page 175. It gives access to BASIC transforms and functions (BASIC is the language supported by the server engine and available in server jobs). For a description of the BASIC functions available see *InfoSphere DataStage Server Job Developer Guide*.

You can only use BASIC transformer stages on SMP systems (not on MPP or cluster systems).

Note: If you encounter a problem when running a job containing a BASIC transformer, you could try increasing the value of the DSIPC_OPEN_TIMEOUT environment variable in the Parallel ► Operator specific category of the environment variable dialog box in the DataStage Administrator (see *InfoSphere DataStage Administrator Client Guide*).

BASIC Transformer stages can have a single input and any number of outputs.

BASIC Transformer stage: fast path

About this task

This section specifies the minimum steps to take to get a BASIC Transformer stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

- In the left pane:
  - Ensure that you have column metadata defined.
- In the right pane:
  - Ensure that you have column metadata defined for each of the output links. The easiest way to do this is to drag columns across from the input link.
  - Define the derivation for each of your output columns. You can leave this as a straight mapping from an input column, or explicitly define an expression to transform the data before it is output.
  - Optionally specify a constraint for each output link. This is an expression which input rows must satisfy before they are output on a link. Rows that are not output on any of the links can be output on the otherwise link.
  - Optionally specify one or more stage variables. This provides a method of defining expressions which can be reused in your output columns derivations (stage variables are only visible within the stage).

BASIC Transformer editor components

The BASIC Transformer Editor has the following components.

BASIC Transformer stage: Toolbar

The Transformer toolbar contains the following buttons (from left to right):

- Stage properties
- Constraints
- Show all

- Show/hide stage variables
- Cut
- Copy
- Paste
- Find/replace
- Load column definition
- Save column definition
- Column auto-match
- Input link execution order
- Output link execution order

BASIC Transformer stage: Link area

The top area displays links to and from the BASIC Transformer stage, showing their columns and the relationships between them.

The link area is where all column definitions and stage variables are defined.

The link area is divided into two panes; you can drag the splitter bar between them to resize the panes relative to one another. There is also a horizontal scroll bar, allowing you to scroll the view left or right.

The left pane shows the input link, the right pane shows output links. Output columns that have no derivation defined are shown in red.

Within the Transformer Editor, a single link might be selected at any one time. When selected, the link's title bar is highlighted, and arrowheads indicate any selected columns.

BASIC Transformer stage: Metadata area

The bottom area shows the column metadata for input and output links. Again this area is divided into two panes: the left showing input link meta data and the right showing output link meta data.

The meta data for each link is shown in a grid contained within a tabbed page. Click the tab to bring the required link to the front. That link is also selected in the link area.

If you select a link in the link area, its metadata tab is brought to the front automatically.

You can edit the grids to change the column meta data on any of the links. You can also add and delete metadata.

BASIC Transformer stage: Shortcut menus

The BASIC Transformer Editor shortcut menus are displayed by right-clicking the links in the links area.

There are slightly different menus, depending on whether you right-click an input link, an output link, or a stage variable. The input link menu offers you operations on input columns, the output link menu offers you operations on output columns and their derivations, and the stage variable menu offers you operations on stage variables.

The shortcut menu enables you to:

- Open the Stage Properties dialog box in order to specify stage or link properties.
- Open the Constraints dialog box to specify a constraint (only available for output links).
- Open the Column Auto Match dialog box.

- Display the Find/Replace dialog box.
- Display the Select dialog box.
- Edit, validate, or clear a derivation or stage variable.
- Edit several derivations in one operation.
- Append a new column or stage variable to the selected link.
- Select all columns on a link.
- Insert or delete columns or stage variables.
- Cut, copy, and paste a column or a key expression or a derivation or stage variable.

If you display the menu from the links area background, you can:

- Open the Stage Properties dialog box in order to specify stage or link properties.
- Open the Constraints dialog box in order to specify a constraint for the selected output link.
- Open the Link Execution Order dialog box in order to specify the order in which links should be processed.
- Toggle between viewing link relations for all links, or for the selected link only.
- Toggle between displaying stage variables and hiding them.

Right-clicking in the meta data area of the Transformer Editor opens the standard grid editing shortcut menus.

BASIC Transformer stage basic concepts

When you first edit a Transformer stage, it is likely that you will have already defined what data is input to the stage on the input links. You will use the Transformer Editor to define the data that will be output by the stage and how it will be transformed. (You can define input data using the Transformer Editor if required.)

This section explains some of the basic concepts of using a Transformer stage.

BASIC Transformer stage: Input link

The input data source is joined to the BASIC Transformer stage via the input link.

BASIC Transformer stage: Output links

You can have any number of output links from your Transformer stage.

You might want to pass some data straight through the BASIC Transformer stage unaltered, but it's likely that you'll want to transform data from some input columns before outputting it from the BASIC Transformer stage.

You can specify such an operation by entering an expression or by selecting a transform to apply to the data. InfoSphere DataStage has many built-in transforms, or you can define your own custom transforms that are stored in the Repository and can be reused as required.

The source of an output link column is defined in that column's **Derivation** cell within the Transformer Editor. You can use the Expression Editor to enter expressions or transforms in this cell. You can also simply drag an input column to an output column's **Derivation** cell, to pass the data straight through the BASIC Transformer stage.

In addition to specifying derivation details for individual output columns, you can also specify constraints that operate on entire output links. A constraint is a BASIC expression that specifies criteria that data must meet before it can be passed to the output link. You can also specify a reject link, which is an output link that carries all the data not output on other links, that is, columns that have not met the criteria.

Each output link is processed in turn. If the constraint expression evaluates to TRUE for an input row, the data row is output on that link. Conversely, if a constraint expression evaluates to FALSE for an input row, the data row is not output on that link.

Constraint expressions on different links are independent. If you have more than one output link, an input row might result in a data row being output from some, none, or all of the output links.

For example, if you consider the data that comes from a paint shop, it could include information about any number of different colors. If you want to separate the colors into different files, you would set up different constraints. You could output the information about green and blue paint on LinkA, red and yellow paint on LinkB, and black paint on LinkC.

When an input row contains information about yellow paint, the LinkA constraint expression evaluates to FALSE and the row is not output on LinkA. However, the input data does satisfy the constraint criterion for LinkB and the rows are output on LinkB.

If the input data contains information about white paint, this does not satisfy any constraint and the data row is not output on Links A, B or C, but will be output on the reject link. The reject link is used to route data to a table or file that is a "catch-all" for rows that are not output on any other link. The table or file containing these rejects is represented by another stage in the job design.

BASIC Transformer stage: Before-stage and after-stage routines

You can specify routines to be executed before or after the stage has processed the data. For example, you might use a before-stage routine to prepare the data before processing starts. You might use an after-stage routine to send an electronic message when the stage has finished.

Editing BASIC transformer stages

About this task

The Transformer Editor enables you to perform the following operations on a BASIC Transformer stage:

- Create new columns on a link
- Delete columns from within a link
- Move columns within a link
- Edit column meta data
- Define output column derivations
- Specify before- and after-stage subroutines
- Define link constraints and handle rejects
- Specify the order in which links are processed
- Define local stage variables

Using drag-and-drop

Many of the BASIC Transformer stage edits can be made simpler by using the Transformer Editor's drag-and-drop functionality.

About this task

You can drag columns from any link to any other link. Common uses are:

- Copying input columns to output links
- Moving columns within a link
- Copying derivations in output links

Procedure

1. Click the source cell to select it.
2. Click the selected cell again and, without releasing the mouse button, drag the mouse pointer to the desired location within the target link. An insert point appears on the target link to indicate where the new cell will go.
3. Release the mouse button to drop the selected cell.

Results

You can drag multiple columns or derivations. Use the standard Explorer keys when selecting the source column cells, then proceed as for a single cell.

You can drag the full column set by dragging the link title.

You can add a column to the end of an existing derivation by holding down the **Ctrl** key as you drag the column.

Find and replace facilities

About this task

If you are working on a complex job where several links, each containing several columns, go in and out of the BASIC Transformer stage, you can use the find/replace column facility to help locate a particular column or expression and change it.

The find/replace facility enables you to:

- Find and replace a column name
- Find and replace expression text
- Find the next empty expression
- Find the next expression that contains an error

To use the find/replace facilities, do one of the following:

- Click the **find/replace** button on the toolbar
- Choose **find/replace** from the link shortcut menu
- Type **Ctrl-F**

The Find and Replace dialog box appears. It has three tabs:

- **Expression Text**. Allows you to locate the occurrence of a particular string within an expression, and replace it if required. You can search up or down, and choose to match case, match whole words, or neither. You can also choose to replace all occurrences of the string within an expression.
- **Columns Names**. Allows you to find a particular column and rename it if required. You can search up or down, and choose to match case, match the whole word, or neither.
- **Expression Types**. Allows you to find the next empty expression or the next expression that contains an error. You can also press **Ctrl-M** to find the next empty expression or **Ctrl-N** to find the next erroneous expression.

  Note: The find and replace results are shown in the color specified in **Tools** > **Options**.

Press **F3** to repeat the last search you made without opening the **Find and Replace** dialog box.

Select facilities

If you are working on a complex job where several links, each containing several columns, go in and out of the Transformer stage, you can use the select column facility to select multiple columns. This facility is also available in the **Mapping** tabs of certain Parallel job stages.

About this task

The select facility enables you to:

- Select all columns/stage variables whose expressions contains text that matches the text specified.
- Select all column/stage variables whose name contains the text specified (and, optionally, matches a specified type).
- Select all columns/stage variable with a certain data type.
- Select all columns with missing or invalid expressions.

To use the select facilities, choose **Select** from the link shortcut menu. The Select dialog box appears. It has three tabs:

- **Expression Text**. This **Expression Text** tab allows you to select all columns/stage variables whose expressions contain text that matches the text specified. The text specified is a simple text match, taking into account the **Match case** setting.
- **Column Names**. The Column Names tab allows you to select all column/stage variables whose Name contains the text specified. There is an additional **Data Type** drop down list, that will limit the columns selected to those with that data type. You can use the **Data Type** drop down list on its own to select all columns of a certain data type. For example, all string columns can be selected by leaving the text field blank, and selecting String as the data type. The data types in the list are generic data types, where each of the column SQL data types belong to one of these generic types.
- **Expression Types**. The **Expression Types** tab allows you to select all columns with either empty expressions or invalid expressions.

Creating and deleting columns

About this task

You can create columns on links to the BASIC Transformer stage using any of the following methods:

- Select the link, then click the **load column definition** button in the toolbar to open the standard load columns dialog box.
- Use drag-and-drop or copy and paste functionality to create a new column by copying from an existing column on another link.
- Use the shortcut menus to create a new column definition.
- Edit the grids in the link's meta data tab to insert a new column.

When copying columns, a new column is created with the same meta data as the column it was copied from.

To delete a column from within the Transformer Editor, select the column you want to delete and click the **cut** button or choose **Delete Column** from the shortcut menu.

Moving columns within a link

About this task

You can move columns within a link using either drag-and-drop or cut and paste. Select the required column, then drag it to its new location, or cut it and paste it in its new location.

Editing column meta data

About this task

You can edit column meta data from within the grid in the bottom of the Transformer Editor. Select the tab for the link meta data that you want to edit, then use the standard InfoSphere DataStage edit grid controls.

The meta data shown does not include column derivations since these are edited in the links area.

Defining output column derivations

You can define the derivation of output columns from within the Transformer Editor in five ways.

About this task

Choose one of the following ways to define the derivation of output columns from within the Transformer Editor:

- If you require a new output column to be directly derived from an input column, with no transformations performed, then you can use drag-and-drop or copy and paste to copy an input column to an output link. The output columns will have the same names as the input columns from which they were derived.
- If the output column already exists, you can drag or copy an input column to the output column's **Derivation** field. This specifies that the column is directly derived from an input column, with no transformations performed.
- You can use the column auto-match facility to automatically set that output columns are derived from their matching input columns.
- You might need one output link column derivation to be the same as another output link column derivation. In this case you can use drag and drop or copy and paste to copy the derivation cell from one column to another.
- In many cases you will need to transform data before deriving an output column from it. For these purposes you can use the Expression Editor. To display the Expression Editor, double-click on the required output link column **Derivation** cell. (You can also invoke the Expression Editor using the shortcut menu or the shortcut keys.)

If a derivation is displayed in red (or the color defined in **Tools** > **Options**), it means that the Transformer Editor considers it incorrect. (In some cases this might simply mean that the derivation does not meet the strict usage pattern rules of the server engine, but will actually function correctly.)

Once an output link column has a derivation defined that contains any input link columns, then a relationship line is drawn between the input column and the output column, as shown in the following example. This is a simple example; there can be multiple relationship lines either in or out of columns. You can choose whether to view the relationships for all links, or just the relationships for the selected links, using the button in the toolbar.

Column auto-match facility:
About this task

This time-saving feature allows you to automatically set columns on an output link to be derived from matching columns on an input link. Using this feature you can fill in all the output link derivations to route data from corresponding input columns, then go back and edit individual output link columns where you want a different derivation.

Procedure

1. Do one of the following:
   - Click the **Auto-match** button in the Transformer Editor toolbar.
   - Choose **Auto-match** from the input link header or output link header shortcut menu.

   TheColumn Auto-Match dialog box appears.
2. Choose the input link and output link that you want to match columns for from the drop down lists.
3. Click **Location match** or **Name match** from the **Match type** area.

If you choose **Location match**, this will set output column derivations to the input link columns in the equivalent positions. It starts with the first input link column going to the first output link column, and works its way down until there are no more input columns left.

4. Click **OK** to proceed with the auto-matching.

   Note: Auto-matching does not take into account any data type incompatibility between matched columns; the derivations are set regardless.

Editing multiple derivations

About this task

You can make edits across several output column or stage variable derivations by choosing **Derivation Substitution...** from the shortcut menu. This opens the Expression Substitution dialog box.

The Expression Substitution dialog box allows you to make the same change to the expressions of all the currently selected columns within a link. For example, if you wanted to add a call to the trim() function around all the string output column expressions in a link, you could do this in two steps. First, use the Select dialog to select all the string output columns. Then use the Expression Substitution dialog to apply a trim() call around each of the existing expression values in those selected columns.

You are offered a choice between Whole expression substitution and Part of expression substitution.

Whole expression:

With this option the whole existing expression for each column is replaced by the replacement value specified.

About this task

This replacement value can be a completely new value, but will usually be a value based on the original expression value. When specifying the replacement value, the existing value of the column's expression can be included in this new value by including "$1". This can be included any number of times.

For example, when adding a trim() call around each expression of the currently selected column set, having selected the required columns, you can use the following procedure.

Procedure

1. Select the **Whole expression** option.
2. Enter a replacement value of:
   ```
   trim($1)
   ```
3. Click **OK**

Results

Where a column's original expression was:

```
DSLink3.col1
```

This will be replaced by:

```
trim(DSLink3.col1)
```

This is applied to the expressions in each of the selected columns.

If you need to include the actual text $1 in your expression, enter it as "$$1".

Part of expression:

With this option, only part of each selected expression is replaced rather than the whole expression. The part of the expression to be replaced is specified by a Regular Expression match.

About this task

It is possible that more that one part of an expression string could match the Regular Expression specified. If **Replace all occurrences** is checked, then each occurrence of a match will be updated with the replacement value specified. If it is not checked, then just the first occurrence is replaced.

When replacing part of an expression, the replacement value specified can include that part of the original expression being replaced. In order to do this, the Regular Expression specified must have round brackets around its value. "$1" in the replacement value will then represent that matched text. If the Regular Expression is not surrounded by round brackets, then "$1" will simply be the text "$1".

For complex Regular Expression usage, subsets of the Regular Expression text can be included in round brackets rather than the whole text. In this case, the entire matched part of the original expression is still replaced, but "$1", "$2" etc can be used to refer to each matched bracketed part of the Regular Expression specified.

The following is an example of the Part of expression replacement.

Suppose a selected set of columns have derivations that use input columns from `DSLink3'. For example, two of these derivations could be:

```
DSLink3.OrderCount + 1
If (DSLink3.Total > 0) Then DSLink3.Total Else -1
```

You might want to protect the usage of these input columns from null values, and use a zero value instead of the null. Use the following procedure to do this.

Procedure

1. Select the columns you want to substitute expressions for.
2. Select the **Part of expression** option.
3. Specify a Regular Expression value of:
   ```
   (DSLink3\.[a-z,A-Z,0-9]*)
   ```
4. Specify a replacement value of
   ```
   NullToZero($1)
   ```
5. Click **OK**, to apply this to all the selected column derivations.

Results

From the examples above:

```
DSLink3.OrderCount + 1
```

would become

```
NullToZero(DSLink3.OrderCount) + 1
```

and

```
If (DSLink3.Total > 0) Then DSLink3.Total Else -1
```

would become:

```
If (NullToZero(DSLink3.Total) > 0) Then DSLink3.Total Else -1
```

If the **Replace all occurrences** option is selected, the second expression will become:

```
If (NullToZero(DSLink3.Total) > 0)
Then NullToZero(DSLink3.Total)
Else -1
```

The replacement value can be any form of expression string. For example in the case above, the replacement value could have been:

```
(If (StageVar1 > 50000) Then $1 Else ($1 + 100))
```

In the first case above, the expression

```
DSLink3.OrderCount + 1
```

would become:

```
(If (StageVar1 > 50000) Then DSLink3.OrderCount
Else (DSLink3.OrderCount + 100)) + 1
```

Specifying before-stage and after-stage subroutines

About this task

You can specify BASIC routines to be executed before or after the stage has processed the data.

To specify a routine, click the stage properties button in the toolbar to open the Stage Properties dialog box.

The General tab contains the following fields:

- **Before-stage subroutine** and **Input Value**. Contain the name (and value) of a subroutine that is executed before the stage starts to process any data.
- **After-stage subroutine** and **Input Value**. Contain the name (and value) of a subroutine that is executed after the stage has processed the data.

Choose a routine from the drop-down list box. This list box contains all the built routines defined as a **Before/After Subroutine** under the **Routines** branch in the Repository. Enter an appropriate value for the routine's input argument in the **Input Value** field.

If you choose a routine that is defined in the Repository, but which was edited but not compiled, a warning message reminds you to compile the routine when you close the **Transformer stage** dialog box.

If you installed or imported a job, the **Before-stage subroutine** or **After-stage subroutine** field might reference a routine that does not exist on your system. In this case, a warning message appears when you close the dialog box. You must install or import the "missing" routine or choose an alternative one to use.

A return code of 0 from the routine indicates success, any other code indicates failure and causes a fatal error when the job is run.

Defining constraints and handling reject links

You can define a constraint to define limits for output data. You can also specify reject links.

About this task

You can define limits for output data by specifying a constraint. Constraints are expressions and you can specify a constraint for each output link from a Transformer stage. You can also specify that a particular link is to act as a reject link. Reject links output rows that have not been written on any other output links from the Transformer stage because they have failed or constraints or because a write failure has occurred.

To define a constraint or specify an otherwise link, do one of the following:

- Select an output link and click the **constraints** button.
- Double-click the output link's constraint entry field.
- Choose **Constraints** from the background or header shortcut menus.

A dialog box appears which allows you either to define constraints for any of the Transformer output links or to define a link as an reject link.

Define a constraint by entering a expression in the **Constraint** field for that link. Once you have done this, any constraints will appear below the link's title bar in the Transformer Editor. This constraint expression will then be checked against the row data at runtime. If the data does not satisfy the constraint, the row will not be written to that link. It is also possible to define a link which can be used to catch these rows which have been rejected from a previous link.

A reject link can be defined by choosing **Yes** in the **Reject Row** field and setting the **Constraint** field as follows:

- To catch rows which are rejected from a specific output link, set the **Constraint** field to *linkname*.REJECTED. This will be set whenever a row is rejected on the *linkname* link, whether because the row fails to match a constraint on that output link, or because a write operation on the target fails for that row. Note that such an otherwise link should occur *after* the output link from which it is defined to catch rejects.
- To catch rows which caused a write failures on an output link, set the **Constraint** field to *linkname*.REJECTEDCODE. The value of *linkname*.REJECTEDCODE will be non-zero if the row was rejected due to a write failure or 0 (DSE.NOERROR) if the row was rejected due to the link constraint not being met. When editing the **Constraint** field, you can set return values for *linkname*.REJECTEDCODE by selecting from the Expression Editor **Link Variables > Constants...** menu options. These give a range of errors, but note that most write errors return DSE.WRITERROR.

  In order to set a reject constraint which differentiates between a write failure and a constraint not being met, a combination of the *linkname*.REJECTEDCODE and *linkname*.REJECTED flags can be used. For example:
  - To catch rows which have failed to be written to an output link, set the **Constraint** field to *linkname*.REJECTEDCODE
  - To catch rows which do not meet a constraint on an output link, set the **Constraint** field to *linkname*.REJECTEDCODE = DSE.NOERROR AND *linkname*.REJECTED
  - To catch rows which have been rejected due a a constraint or write error, set the **Constraint** field to *linkname*.REJECTED
- As a "catch all", the **Constraint** field can be left blank. This indicates that this otherwise link will catch all rows which have not been successfully written to *any* of the output links processed up to this point. Therefore, the otherwise link should be the last link in the defined processing order.
- Any other **Constraint** can be defined. This will result in the number of rows written to that link (that is, rows which satisfy the constraint) to be recorded in the job log as "rejected rows".

  Note: Due to the nature of the "catch all" case above, you should only use one reject link whose **Constraint** field is blank. To use multiple reject links, you should define them to use the *linkname*.REJECTED flag detailed in the first case above.

Specifying link order

You can specify links to be in a particular order.

About this task

You can specify the order in which output links process a row. The initial order of the links is the order in which they are added to the stage.

Procedure

1. Do one of the following:
   - Click the **output link execution order** button on the Transformer Editor toolbar.
   - Choose **output link reorder** from the background shortcut menu.
   - Click the **stage properties** button in the Transformer toolbar or choose **stage properties** from the background shortcut menu and click on the stage page Link Ordering tab.

     The Link Ordering tab appears:
2. Use the arrow buttons to rearrange the list of links in the execution order required.
3. When you are happy with the order, click **OK**.

   Note: Although the link ordering facilities mean that you can use a previous output column to derive a subsequent output column, this is not recommended, and you will receive a warning if you do so.

Defining local stage variables

You can declare a stage variable.

About this task

You can declare and use your own variables within a BASIC Transformer stage. Such variables are accessible only from the BASIC Transformer stage in which they are declared. They can be used as follows:

- They can be assigned values by expressions.
- They can be used in expressions which define an output column derivation.
- Expressions evaluating a variable can include other variables or the variable being evaluated itself.

Any stage variables you declare are shown in a table in the right pane of the links area. The table looks similar to an output link. You can display or hide the table by clicking the **Stage Variable** button in the Transformer toolbar or choosing **Stage Variable** from the background shortcut menu.

Note: Stage variables are not shown in the output link meta data area at the bottom of the right pane.

The table lists the stage variables together with the expressions used to derive their values. Link lines join the stage variables with input columns used in the expressions. Links from the right side of the table link the variables to the output columns that use them.

Procedure

1. Do one of the following:
   - Click the **stage properties** button in the Transformer toolbar.
   - Choose **stage properties** from the background shortcut menu.

     The Transformer Stage Properties dialog box appears.
2. Click the Variables tab on the General page. The Variables tab contains a grid showing currently declared variables, their initial values, and an optional description. Use the standard grid controls to add new variables. Variable names must begin with an alphabetic character (a-z, A-Z) and can only contain alphanumeric characters (a-z, A-Z, 0-9). Ensure that the variable does not use the name of any BASIC keywords.

Results

Variables entered in the Stage Properties dialog box appear in the Stage Variable table in the links pane.

You perform most of the same operations on a stage variable as you can on an output column (see Defining Output Column Derivations). A shortcut menu offers the same commands. You cannot, however,

paste a stage variable as a new column, or a column as a new stage variable.

The InfoSphere DataStage expression editor

The InfoSphere DataStage Expression Editor helps you to enter correct expressions when you edit BASIC Transformer stages. The Expression Editor can:

- Facilitate the entry of expression elements
- Complete the names of frequently used variables
- Validate variable names and the complete expression

The Expression Editor can be opened from:

- Output link **Derivation** cells
- Stage variable **Derivation** cells
- Constraint dialog box
- Transform dialog box in the Designer

Expression format

The format of an expression is as follows:

```
KEY:

something_like_this       is a token
something_in_italics      is a terminal, that is, does not break down any
   further
|     is a choice between tokens
[     is an optional part of the construction
"XXX"     is a literal token (that is, use XXX not
   including the quotes)
================================================
expression ::=    function_call |
      variable_name |
   other_name |
   constant |
   unary_expression |
   binary_expression |
   if_then_else_expression |
   substring_expression |
   "(" expression ")"

function_call ::= function_name "(" [argument_list] ")"
argument_list ::= expression | expression "," argument_list
function_name ::=    name of a built-in function |
   name of a user-defined_function
variable_name ::=       job_parameter name |
      stage_variable_name |
      link_variable name
other_name ::=    name of a built-in macro, system variable, and so on.
constant ::= numeric_constant | string_constant
numeric_constant ::= ["+" | "-"] digits ["." [digits]] ["E" | "e" ["+" | "-"] digits]
string_constant ::=    "'" [characters] "'" |
   """ [characters] """ |
   "\" [characters] "\"
unary_expression ::= unary_operator expression
unary_operator ::= "+" | "-"
binary_expression ::= expression binary_operator expression
binary_operator ::=    arithmetic_operator |
   concatenation_operator |
   matches_operator |
   relational_operator |
   logical_operator
arithmetic_operator ::= "+" | "-" | "*" | "/" | "^"
concatenation_operator ::= ":"
```

```
matches_operator ::= "MATCHES"
relational_operator ::=    "   =" |"EQ" |
      "<>" | "#" | "NE" |
      ">" | "GT" |
      ">=" | "=>" | "GE" |
      "<" | "LT" |
      "<=" | "=<" | "LE"
logical_operator ::= "AND" | "OR"
if_then_else_expression ::= "IF" expression "THEN" expression "ELSE" expression
substring_expression ::= expression "[" [expression ["," expression] "]"
field_expression ::=    expression "[" expression ","
   expression ","
   expression "]"
         /*    That is, always 3 args
```

Note: keywords like "AND" or "IF" or "EQ" might be in any case

Entering expressions
About this task

Whenever the insertion point is in an expression box, you can use the Expression Editor to suggest the next element in your expression. Do this by right-clicking the box, or by clicking the **Suggest** button to the right of the box. This opens the **Suggest Operand** or **Suggest Operator** menu. Which menu appears depends on context, that is, whether you should be entering an operand or an operator as the next expression element.

You will be offered a different selection on the **Suggest Operand** menu depending on whether you are defining key expressions, derivations and constraints, or a custom transform. The **Suggest Operator** menu is always the same.

Completing variable names
About this task

The Expression Editor stores variable names. When you enter a variable name you have used before, you can type the first few characters, then press **F5**. The Expression Editor completes the variable name for you.

If you enter the name of an input link followed by a period, for example, DailySales., the Expression Editor displays a list of the column names of that link. If you continue typing, the list selection changes to match what you type. You can also select a column name using the mouse. Enter a selected column name into the expression by pressing **Tab** or **Enter**. Press **Esc** to dismiss the list without selecting a column name.

Validating the expression
About this task

When you have entered an expression in the Transformer Editor, press **Enter** to validate it. The Expression Editor checks that the syntax is correct and that any variable names used are acceptable to the compiler. When using the Expression Editor to define a custom transform, click **OK** to validate the expression.

If there is an error, a message appears and the element causing the error is highlighted in the expression box. You can either correct the expression or close the Transformer Editor or Transform dialog box.

Within the Transformer Editor, the invalid expressions are shown in red. (In some cases this might simply mean that the expression does not meet the strict usage pattern rules of the server engine, but will actually function correctly.)

Exiting the expression editor

There are a few ways in which you can exit the expression editor.

About this task

You can exit the Expression Editor in the following ways:

- Press **Esc** (which discards changes).
- Press **Return** (which accepts changes).
- Click outside the Expression Editor box (which accepts changes).

Configuring the expression editor

About this task

You can resize the Expression Editor window by dragging. The next time you open the expression editor in the same context (for example, editing output columns) on the same client, it will have the same size.

The Expression Editor is configured by editing the Designer options. This allows you to specify how `helpful' the expression editor is. For more information, see *InfoSphere DataStage Designer Client Guide*.

BASIC Transformer stage properties

The Transformer stage has a Properties dialog box which allows you to specify details about how the stage operates.

The Transform Stage dialog box has three pages:

- Stage page. This is used to specify general information about the stage.
- Input page. This is where you specify details about the data input to the Transformer stage.
- Output page. This is where you specify details about the output links from the Transformer stage.

BASIC Transformer stage: Stage page

The Stage page has four tabs:

- General. Allows you to enter an optional description of the stage and specify a before-stage or after-stage subroutine.
- Variables. Allows you to set up stage variables for use in the stage.
- Link Ordering. Allows you to specify the order in which the output links will be processed.
- Advanced. Allows you to specify how the stage executes.

The General tab is described in "Before-Stage and After-Stage Routines" . The Variables tab is described in "Defining Local Stage Variables". The Link Ordering tab is described in "Specifying Link Order".

BASIC Transformer stage: Advanced tab:

The Advanced tab is the same as the Advanced tab of the generic stage editor as described in "Advanced Tab". This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In sequential mode the data is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is set to **Propagate** by default, this sets or clears the partitioning in accordance with what the previous stage has set. You can also select **Set** or **Clear**. If you select **Set**, the stage will request that the next stage preserves the partitioning as is.

- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

BASIC Transformer stage: Input page

The Input page allows you to specify details about data coming into the Transformer stage. The Transformer stage can have only one input link.

The General tab allows you to specify an optional description of the input link.

The Partitioning tab allows you to specify how incoming data is partitioned. This is the same as the Partitioning tab in the generic stage editor described in "Partitioning Tab".

BASIC Transformer stage: Partitioning tab:

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected when input to the BASIC Transformer stage. It also allows you to specify that the data should be sorted on input.

By default the BASIC Transformer stage will attempt to preserve partitioning of incoming data, or use its own partitioning method according to what the previous stage in the job dictates.

If the BASIC Transformer stage is operating in sequential mode, it will first collect the data before writing it to the file using the default collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the BASIC Transformer stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partitioning type** drop-down list. This will override any current partitioning.

If the BASIC Transformer stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Transformer stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.

- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default method for the Transformer stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The **Partitioning** tab also allows you to specify that data arriving on the input link should be sorted. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning method chosen.

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

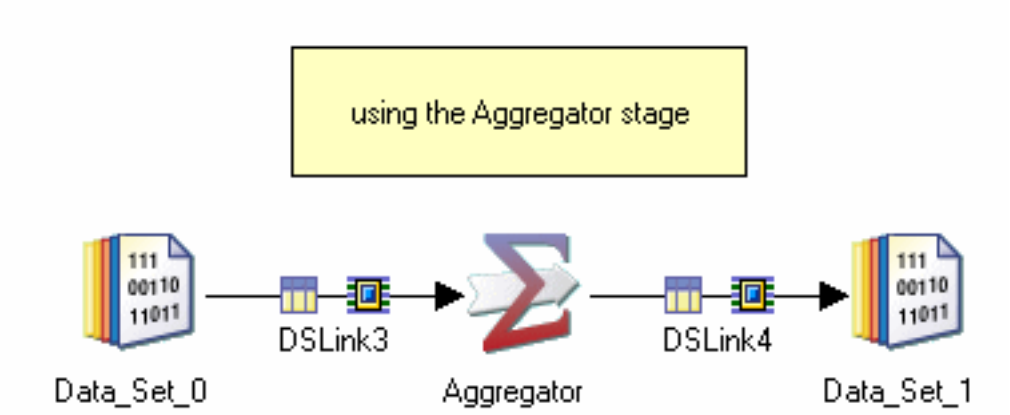
You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

BASIC Transformer stage: Output page

The Output Page has a General tab which allows you to enter an optional description for each of the output links on the BASIC Transformer stage. The Advanced tab allows you to change the default buffering settings for the output links.

Aggregator stage

The Aggregator stage is a processing stage. It classifies data rows from a single input link into groups and computes totals or other aggregate functions for each group. The summed totals for each group are output from the stage via an output link.



When you edit an Aggregator stage, the Aggregator stage editor appears. This is based on the generic stage editor described in Chapter 4, “Stage editors,” on page 49.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data being grouped or aggregated.
- **Output Page**. This is where you specify details about the groups being output from the stage.

The aggregator stage gives you access to grouping and summary operations. One of the easiest ways to expose patterns in a collection of records is to group records with similar characteristics, then compute statistics on all records in the group. You can then use these statistics to compare properties of the different groups. For example, records containing cash register transactions might be grouped by the day of the week to see which day had the largest number of transactions, the largest amount of revenue, and so on.

Records can be grouped by one or more characteristics, where record characteristics correspond to column values. In other words, a group is a set of records with the same value for one or more columns. For example, transaction records might be grouped by both day of the week and by month. These groupings might show that the busiest day of the week varies by season.

In addition to revealing patterns in your data, grouping can also reduce the volume of data by summarizing the records in each group, making it easier to manage. If you group a large volume of data on the basis of one or more characteristics of the data, the resulting data set is generally much smaller than the original and is therefore easier to analyze using standard workstation or PC-based tools.

At a practical level, you should be aware that, in a parallel environment, the way that you partition data before grouping and summarizing it can affect the results. For example, if you partitioned using the round robin method, records with identical values in the column you are grouping on would end up in different partitions. If you then performed a sum operation within these partitions you would not be operating on all the relevant columns. In such circumstances you might want to key partition the data on one or more of the grouping keys to ensure that your groups are entire.

It is important that you bear these facts in mind and take any steps you need to prepare your data set before presenting it to the Aggregator stage. In practice this could mean you use Sort stages or additional Aggregate stages in the job.

Aggregator stage example

The example data is from a freight carrier who charges customers based on distance, equipment, packing, and license requirements. They need a report of distance traveled and charges grouped by date and license type.

The following table shows a sample of the data:

Table 18. Sample of data

Ship Date	District	Distance	Equipment	Packing	License	Charge
...						
2000-06-02	1	1540	D	M	BUN	1300
2000-07-12	1	1320	D	C	SUM	4800
2000-08-02	1	1760	D	C	CUM	1300
2000-06-22	2	1540	D	C	CUN	13500
2000-07-30	2	1320	D	M	SUM	6000
...						

The stage will output the following columns:

Table 19. Output column definitions

Column name	SQL Type
DistanceSum	Decimal
DistanceMean	Decimal
ChargeSum	Decimal
ChargeMean	Decimal
license	Char
shipdate	Date

The stage first hash partitions the incoming data on the license column, then sorts it on license and date:

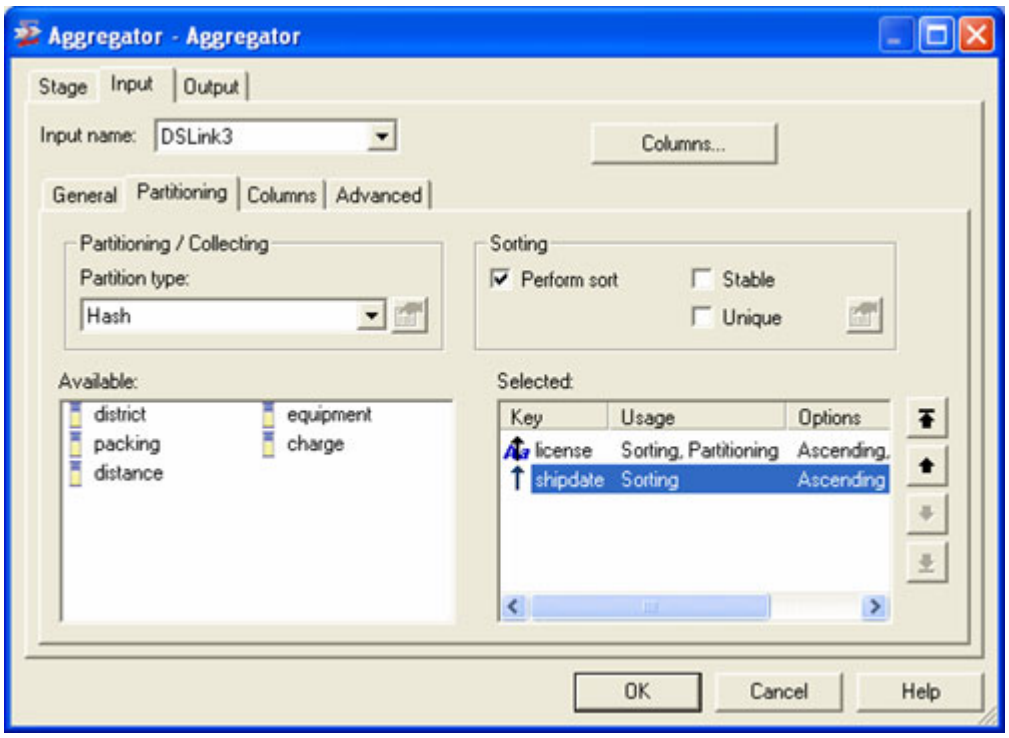


Figure 7. Partitioning tab

The properties are then used to specify the grouping and the aggregating of the data:

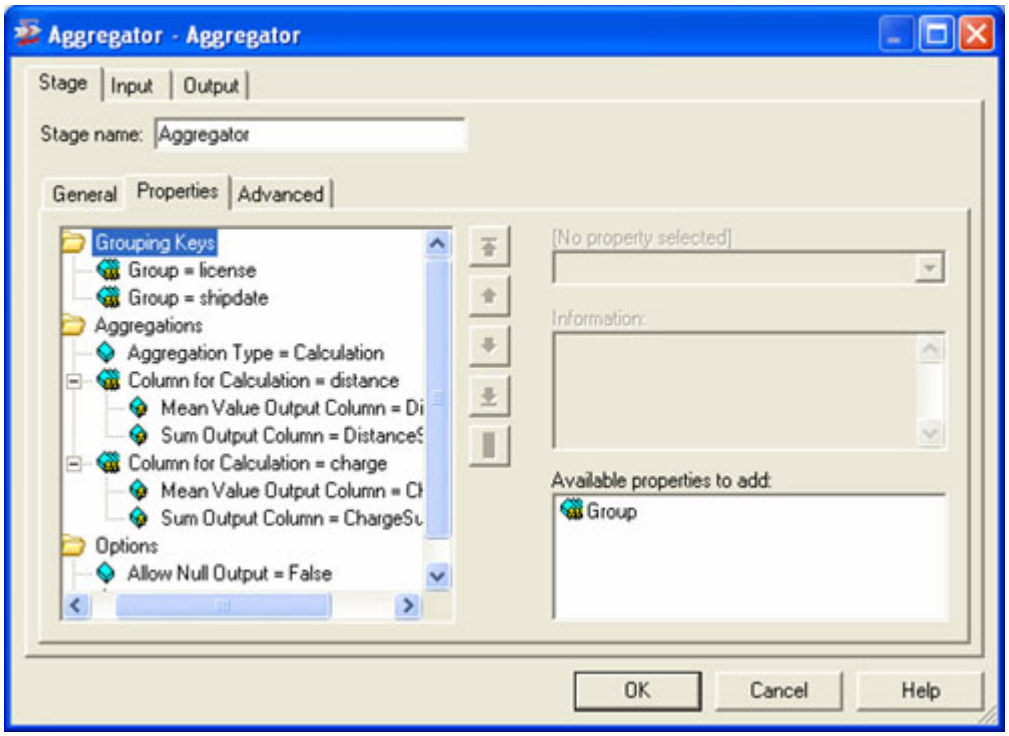


Figure 8. Properties tab

The following is a sample of the output data:

Table 20. Output data

Ship Date	License	Distance Sum	Distance Mean	Charge Sum	Charge Mean
...					
2000-06-02	BUN	1126053.00	1563.93	20427400.00	28371.39
2000-06-12	BUN	2031526.00	2074.08	22426324.00	29843.55
2000-06-22	BUN	1997321.00	1958.45	19556450.00	19813.26
2000-06-30	BUN	1815733.00	1735.77	17023668.00	18453.02
...					

If you wanted to go on and work out the sum of the distance and charge sums by license, you could insert another Aggregator stage with the following properties:

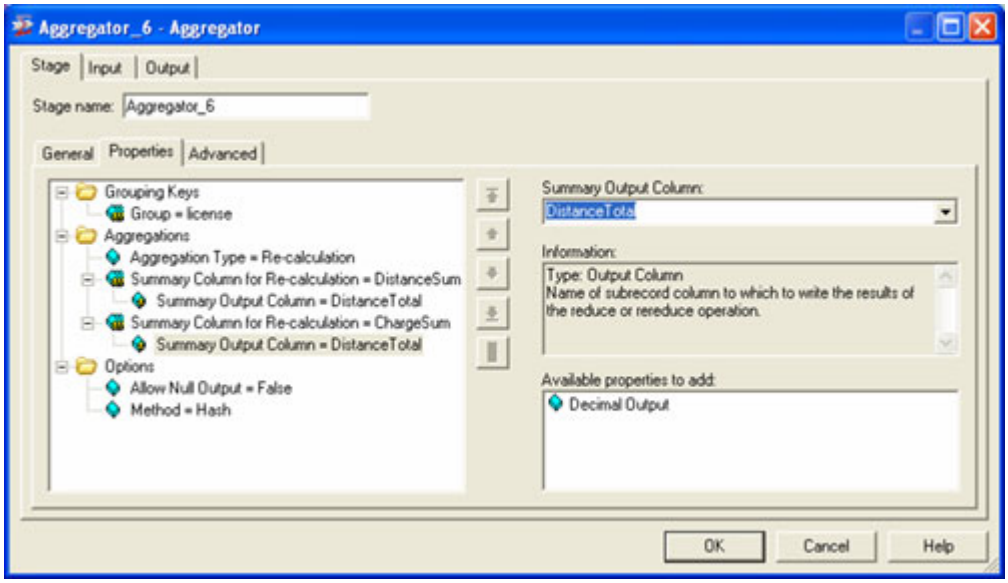


Figure 9. Second Aggregator stage

Aggregator stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Aggregator stages in a job. This section specifies the minimum steps to take to get an Aggregator stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use an aggregator stage:

- In the Stage page **Properties Tab**, under the Grouping Keys category:
  - Specify the key column that the data will be grouped on. You can repeat the key property to specify composite keys.

    Under the Aggregations category:
  - Choose an aggregation type. Calculation is the default, and allows you to summarize a column or columns. Count rows allows you to count the number of rows within each group. Re-calculation allows you to apply aggregate functions to a column that has already been summarized.

Other properties depend on the aggregate type chosen:

- If you have chosen the Calculation aggregation type, specify the column to be summarized in Column for Calculation. You can repeat this property to specify multiple columns. Choose one or more dependent properties to specify the type of aggregation to perform, and the name of the output column that will hold the result.
- If you have chosen the Count Rows aggregation type, specify the output column that will hold the count.
- If you have chosen the Re-calculation aggregation type, specify the column to be re-calculated. You can repeat this property to specify multiple columns. Choose one or more dependent properties to specify the type of aggregation to perform, and the name of the output column that will hold the result.

- In the Output page **Mapping Tab**, check that the mapping is as you expect (InfoSphere DataStage maps data onto the output columns according to what you specify in the **Properties** tab).

Aggregator stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Aggregator stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 21. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Grouping Keys/Group	Input column	N/A	Y	Y	N/A
Grouping Keys/Case Sensitive	True/ False	True	N	N	Group
Aggregations/ Aggregation Type	Calculation/ Recalculation/ Count rows	Calculation	Y	N	N/A
Aggregations/ Column for Calculation	Input column	N/A	Y (if Aggregation Type = Calculation)	Y	N/A
Aggregations/ Count Output Column	Output column	N/A	Y (if Aggregation Type = Count Rows)	Y	N/A
Aggregations/ Summary Column for Recalculation	Input column	N/A	Y (if Aggregation Type = Recalculation)	Y	N/A
Aggregations/ Default To Decimal Output	precision, scale	8,2	N	N	N/A

Table 21. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Aggregations/ Corrected Sum of Squares	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Maximum Value	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Mean Value	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Minimum Value	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Missing Value	Output column	N/A	N	Y	Column for Calculation
Aggregations/ Missing Values Count	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Non-missing Values Count	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Percent Coefficient of Variation	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Range	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Standard Deviation	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Standard Error	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation

Table 21. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Aggregations/ Sum of Weights	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Sum	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Summary	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Uncorrected Sum of Squares	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Variance	Output column	N/A	N	N	Column for Calculation & Summary Column for Recalculation
Aggregations/ Variance divisor	Default/ Nrecs	Default	N	N	Variance
Aggregations/ Calculation and Recalculation Dependent Properties	Input column	N/A	N	N	Column for Calculation or Count Output Column
Aggregations/ Decimal Output	precision, scale	8,2	N	N	Calculation or Recalculation method
Options/Group	hash/sort	hash	Y	Y	N/A
Options/Allow Null Outputs	True/ False	False	Y	N	N/A

Aggregator stage: Grouping keys category: Group

Specifies the input columns you are using as group keys. Repeat the property to select multiple columns as group keys. You can use the Column Selection dialog box to select several group keys at once if required). This property has a dependent property:

- **Case Sensitive**

  Use this to specify whether each group key is case sensitive or not, this is set to True by default, that is, the values "CASE" and "case" in would end up in different groups.

Aggregator stage: Aggregations category:

Aggregation type

This property allows you to specify the type of aggregation operation your stage is performing. Choose from Calculate (the default), Recalculate, and Count Rows.

Column for calculation

The Calculate aggregate type allows you to summarize the contents of a particular column or columns in your input data set by applying one or more aggregate functions to it. Select the column to be aggregated, then select dependent properties to specify the operation to perform on it, and the output column to carry the result. You can use the Column Selection dialog box to select several columns for calculation at once if required).

Count output column

The Count Rows aggregate type performs a count of the number of records within each group. Specify the column on which the count is output.

Summary column for recalculation

This aggregate type allows you to apply aggregate functions to a column that has already been summarized. This is like calculate but performs the specified aggregate operation on a set of data that has already been summarized. In practice this means you should have performed a calculate (or recalculate) operation in a previous Aggregator stage with the Summary property set to produce a subrecord containing the summary data that is then included with the data set. Select the column to be aggregated, then select dependent properties to specify the operation to perform on it, and the output column to carry the result. You can use the Column Selection dialog box to select several columns for recalculation at once if required).

Weighting column

Configures the stage to increment the count for the group by the contents of the weight column for each record in the group, instead of by 1. Not available for Summary Column for Recalculation. Setting this option affects only the following options:

- Percent Coefficient of Variation
- Mean Value
- Sum
- Sum of Weights
- Uncorrected Sum of Squares

Default to decimal output

The output type of a calculation or recalculation column is double. Setting this property causes it to default to decimal. You can also set a default precision and scale. (You can also specify that individual columns have decimal output while others retain the default type of double.)

Aggregator stage: Options category:
Method

The aggregate stage has two modes of operation: **hash** and **sort**. Your choice of mode depends primarily on the number of groupings in the input data set, taking into account the amount of memory available. You typically use hash mode for a relatively small number of groups; generally, fewer than about 1000 groups per megabyte of memory to be used.

When using hash mode, you should hash partition the input data set by one or more of the grouping key columns so that all the records in the same group are in the same partition (this happens automatically if auto is set in the Partitioning tab). However, hash partitioning is not mandatory, you can use any partitioning method you choose if keeping groups together in a single partition is not important. For example, if you're summing records in each partition and later you'll add the sums across all partitions, you don't need all records in a group to be in the same partition to do this. Note, though, that there will be multiple output records for each group.

If the number of groups is large, which can happen if you specify many grouping keys, or if some grouping keys can take on many values, you would normally use sort mode. However, sort mode requires the input data set to have been partition sorted with all of the grouping keys specified as hashing and sorting keys (this happens automatically if auto is set in the Partitioning tab). Sorting requires a pregrouping operation: after sorting, all records in a given group in the same partition are consecutive.

The method property is set to **hash** by default.

You might want to try both modes with your particular data and application to determine which gives the better performance. You might find that when calculating statistics on large numbers of groups, sort mode performs better than hash mode, assuming the input data set can be efficiently sorted before it is passed to group.

Allow null outputs

Set this to True to indicate that null is a valid output value when calculating minimum value, maximum value, mean value, standard deviation, standard error, sum, sum of weights, and variance. If False, the null value will have 0 substituted when all input values for the calculation column are null. It is False by default.

Aggregator stage: Calculation and recalculation dependent properties:

The following properties are dependents of both Column for Calculation and Summary Column for Recalculation. These specify the various aggregate functions and the output columns to carry the results.

- **Corrected Sum of Squares**

  Produces a corrected sum of squares for data in the aggregate column and outputs it to the specified output column.
- **Maximum Value**

  Gives the maximum value in the aggregate column and outputs it to the specified output column.
- **Mean Value**

  Gives the mean value in the aggregate column and outputs it to the specified output column.
- **Minimum Value**

  Gives the minimum value in the aggregate column and outputs it to the specified output column.
- **Missing Value**

  This specifies what constitutes a "missing" value, for example -1 or NULL. Enter the value as a floating point number. Not available for Summary Column to Recalculate.
- **Missing Values Count**

  Counts the number of aggregate columns with missing values in them and outputs the count to the specified output column. Not available for recalculate.
- **Non-missing Values Count**

  Counts the number of aggregate columns with values in them and outputs the count to the specified output column.
- **Percent Coefficient of Variation**

Calculates the percent coefficient of variation for the aggregate column and outputs it to the specified output column.

- **Range**

  Calculates the range of values in the aggregate column and outputs it to the specified output column.
- **Standard Deviation**

  Calculates the standard deviation of values in the aggregate column and outputs it to the specified output column.
- **Standard Error**

  Calculates the standard error of values in the aggregate column and outputs it to the specified output column.
- **Sum of Weights**

  Calculates the sum of values in the weight column specified by the Weight column property and outputs it to the specified output column.
- **Sum**

  Sums the values in the aggregate column and outputs the sum to the specified output column.
- **Summary**

  Specifies a subrecord to write the results of the calculate or recalculate operation to.
- **Uncorrected Sum of Squares**

  Produces an uncorrected sum of squares for data in the aggregate column and outputs it to the specified output column.
- **Variance**

  Calculates the variance for the aggregate column and outputs the sum to the specified output column. This has a dependent property:
  - **Variance divisor**

    Specifies the variance divisor. By default, uses a value of the number of records in the group minus the number of records with missing values minus 1 to calculate the variance. This corresponds to a vardiv setting of Default. If you specify NRecs, InfoSphere DataStage uses the number of records in the group minus the number of records with missing values instead.

Each of these properties has a dependent property as follows:

- **Decimal Output**. By default all calculation or recalculation columns have an output type of double. This property allows you to specify that the column has an output type of decimal. You can also specify a precision and scale for they type (by default 8,2).

Aggregator stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data set is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Set** by default. You can select **Set** or **Clear**. If you select **Set** the stage will request that the next stage in the job attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.

- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Aggregator stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Aggregator stage uses this when it is grouping by key to determine the order of the key fields. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Aggregator stage: Input page

The Input page allows you to specify details about the incoming data set.

The General tab allows you to specify an optional description of the input link. The **Partitioning** tab allows you to specify how incoming data is partitioned before being grouped or summarized. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Aggregator stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Aggregator stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is grouped or summarized. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Aggregator stage is operating in sequential mode, it will first collect the data before writing it to the file using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Aggregator stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Aggregator stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Aggregator stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Aggregator stage.

- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Aggregator stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being written to the file or files. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto modes).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Aggregator stage: Output page

The Output page allows you to specify details about data output from the Aggregator stage. The Aggregator stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The **Columns** tab specifies the column definitions of incoming data. The Mapping tab allows you to specify the relationship

between the processed data being produced by the Aggregator stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Aggregator stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Aggregator stage: Mapping tab

For the Aggregator stage the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them or how they are generated.

The left pane shows the input columns or the generated columns. These are read only and cannot be modified on this tab.

The right pane shows the output columns for each link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging columns over from the left pane, or by using the Auto-match facility.

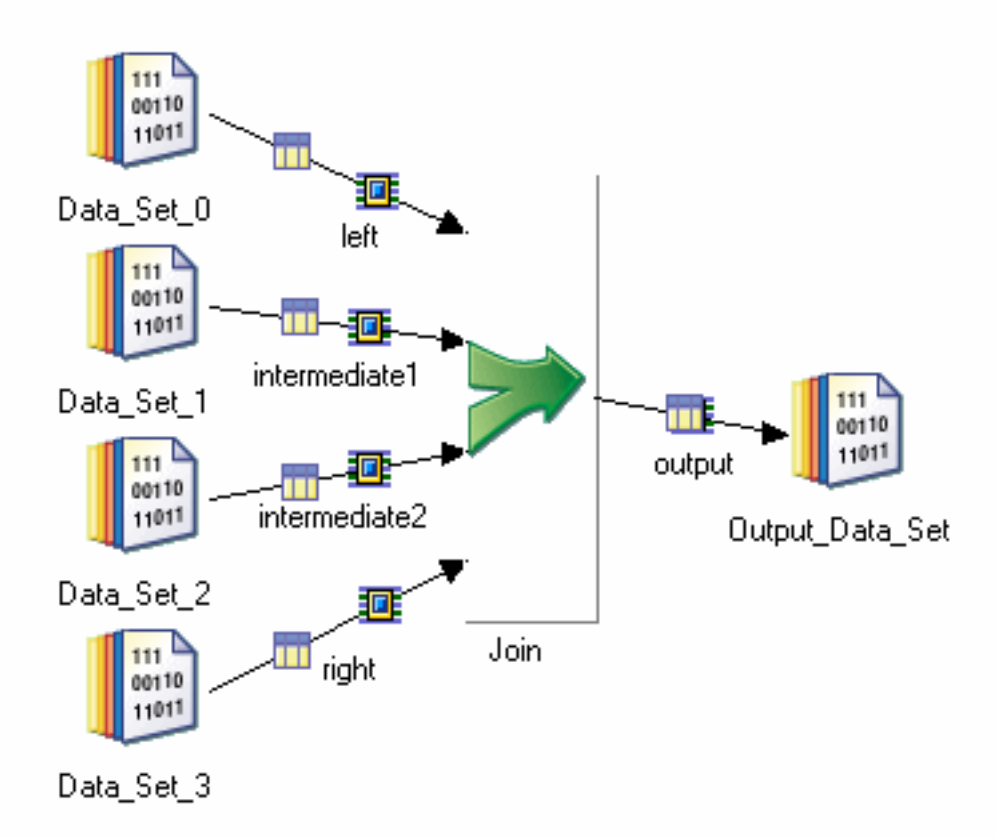
Join stage

The Join stage is a processing stage. It performs join operations on two or more data sets input to the stage and then outputs the resulting data set. The Join stage is one of three stages that join tables based on the values of key columns. The other two are:

- “Lookup Stage” on page 254
- “Merge Stage” on page 246

The three stages differ mainly in the memory they use, the treatment of rows with unmatched keys, and their requirements for data being input (for example, whether it is sorted). See "Join Versus Lookup" for help in deciding which stage to use.

In the Join stage, the input data sets are notionally identified as the "right" set and the "left" set, and "intermediate" sets. You can specify which is which. It has any number of input links and a single output link.



The stage can perform one of four join operations:

- **Inner** transfers records from input data sets whose key columns contain equal values to the output data set. Records whose key columns do not contain equal values are dropped.
- **Left outer** transfers all values from the left data set but transfers values from the right data set and intermediate data sets only where key columns match. The stage drops the key column from the right and intermediate data sets.
- **Right outer** transfers all values from the right data set and transfers values from the left data set and intermediate data sets only where key columns match. The stage drops the key column from the left and intermediate data sets.
- **Full outer** transfers records in which the contents of the key columns are equal from the left and right input data sets to the output data set. It also transfers records whose key columns contain unequal values from both input data sets to the output data set. (Full outer joins do not support more than two input links.)

The data sets input to the Join stage must be key partitioned and sorted in ascending order. This ensures that rows with the same key column values are located in the same partition and will be processed by the same node. It also minimizes memory requirements because fewer rows need to be in memory at any one time. Choosing the auto partitioning method will ensure that partitioning and sorting is done. If sorting and partitioning are carried out on separate stages before the Join stage, InfoSphere DataStage in auto mode will detect this and not repartition (alternatively you could explicitly specify the Same partitioning method).

The Join stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data sets being joined.
- **Output Page**. This is where you specify details about the joined data being output from the stage.

Join versus lookup

InfoSphere DataStage does not know how large your data is, so cannot make an informed choice whether to combine data using a join stage or a lookup stage. Here's how to decide which to use:

There are two data sets being combined. One is the primary or driving dataset, sometimes called the left of the join. The other data set(s) are the reference datasets, or the right of the join.

In all cases you are concerned with the size of the reference datasets. If these take up a large amount of memory relative to the physical RAM memory size of the computer you are running on, then a lookup stage might thrash because the reference datasets might not fit in RAM along with everything else that has to be in RAM. This results in very slow performance since each lookup operation can, and typically does, cause a page fault and an I/O operation.

So, if the reference datasets are big enough to cause trouble, use a join. A join does a high-speed sort on the driving and reference datasets. This can involve I/O if the data is big enough, but the I/O is all highly optimized and sequential. Once the sort is over the join processing is very fast and never involves paging or other I/O.

Example joins

The following examples show what happens to two data sets when each type of join operation is applied to them. Here are the two data sets:

Table 22. Data sets

Left Input Data Set		**Right Input Data Set**	
Status	Price	Price	ID
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

Price is the key column which is going to be joined on, and bold type indicates where the data sets share the same value for Price. The data sets are already sorted on that key.

Inner join

Here is the data set that is output if you perform an inner join on the Price key column:

Table 23. Output data set

Status	**Price**	**ID**
sold	125	NI6325
Offered	777	JA1081
Offered	908	DE1911
Offered	908	FR2081
Pending	908	DE1911

Table 23. Output data set (continued)

Status	Price	ID
Pending	908	FR2081

Left outer join

Here is the data set that is output if you perform a left outer join on the Price key column:

Output Data Set		
Status	**Price**	**ID**
sold	125	NI6325
sold	213	
offered	378	
Pending	575	
Pending	649	
Offered	777	JA1081
Offered	908	DE1911
Offered	908	FR2081
Pending	908	DE1911
Pending	908	FR2081

Right outer join

Here is the data set that is output if you perform a right outer join on the Price key column

Table 24. Output data set

Status	Price	ID
	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

Full outer join

Here is the data set that is output if you perform a full outer join on the Price key column:

Table 25. Output data set

Status	Price	Price	ID
		113	NI6325
sold	125	125	BR9658

Table 25. Output data set (continued)

Status	Price	Price	ID
sold	213		
		285	CZ2538
offered	378		
Pending	575		
		628	RU5713
Pending	649		
		668	SA5680
Status	Price	Price	ID
Offered	777	777	JA1081
Offered	908	908	DE1911
Offered	908	908	FR2081
Pending	908	908	DE1911
Pending	908	908	FR2081

Join stage: fast path

About this task

InfoSphere DataStage has many defaults which means that Joins can be simple to set up. This section specifies the minimum steps to take to get a Join stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

- In the Stage page **Properties Tab** specify the key column or columns that the join will be performed on.
- In the Stage page **Properties Tab** specify the join type or accept the default of Inner.
- In the Stage page **Link Ordering Tab**, check that your links are correctly identified as "left", "right", and "intermediate" and reorder if required.
- Ensure required column meta data has been specified (this might be done in another stage).
- In the **Output Page Mapping Tab**, specify how the columns from the input links map onto output columns.

Join stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The Link Ordering tab allows you to specify which of the input links is the right link and which is the left link and which are intermediate. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Join stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 26. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Join Keys/"Key"	Input Column	N/A	Y	Y	N/A
Join Keys/Case Sensitive	True/False	True	N	N	Key
Options/"Join stage: Options category"	Full Outer/ Inner/Left Outer/ Right Outer	Inner	Y	N	N/A

Join stage: Join keys category:
Key

Choose the input column you want to join on. You are offered a choice of input columns common to all links. For a join to work you must join on a column that appears in all input data sets, that is, have the same name and compatible data types. If, for example, you select a column called "name" from the left link, the stage will expect there to be an equivalent column called "name" on the right link.

You can join on multiple key columns. To do so, repeat the Key property. You can use the Column Selection dialog box to select several key columns at once if required).

Key has a dependent property:

Case Sensitive

- Case Sensitive

  Use this to specify whether each group key is case sensitive or not, this is set to True by default, that is, the values "CASE" and "case" in would not be judged equivalent.

Join stage: Options category:
Join type

Specify the type of join operation you want to perform. Choose one of:

- Full Outer
- Inner
- Left Outer
- Right Outer

The default is Inner.

Join stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.

- **Preserve partitioning**. This is **Propagate** by default. It adopts the setting which results from ORing the settings of the input stages, that is, if either of the input stages uses **Set** then this stage will use **Set**. You can explicitly select **Set** or **Clear**. Select **Set** to request that the next stage in the job attempts to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Join stage: Link ordering tab

This tab allows you to specify which input link is regarded as the left link and which link is regarded as the right link, and which links are regarded as intermediate. By default the first link you add is regarded as the left link, and the last one as the right link, with all other links labelled as Intermediate *N*. You can use this tab to override the default order.

In the example DSLink4 is the left link, click on it to select it then click on the down arrow to convert it into the right link.

Join stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Join stage uses this when it is determining the order of the key fields. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Join stage: Input page

The **Input page** allows you to specify details about the incoming data sets. Choose an input link from the **Input name** drop down list to specify which link you want to work on.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being joined. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Join stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Join stage: Partitioning on input links

The Partitioning tab allows you to specify details about how the data on each of the incoming links is partitioned or collected before it is joined. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. Auto mode ensures that data being input to the Join stage is key partitioned and sorted.

If the Join stage is operating in sequential mode, it will first collect the data before writing it to the file using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Join stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Join stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Join stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default collection method for the Join stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Join stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available. In the case of a Join stage, Auto will also ensure that the collected data is sorted.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to explicitly specify that data arriving on the input link should be sorted before being joined (you might use this if you have selected a partitioning method other than auto or same). The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Join stage: Output page

The **Output page** allows you to specify details about data output from the Join stage. The Join stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Join stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Join stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Join stage: Mapping tab

For Join stages the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them.

The left pane shows the input columns from the links whose tables have been joined. These are read only and cannot be modified on this tab.

The right pane shows the output columns for the output link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility.

Merge Stage

The Merge stage is a processing stage. It can have any number of input links, a single output link, and the same number of reject links as there are update input links.

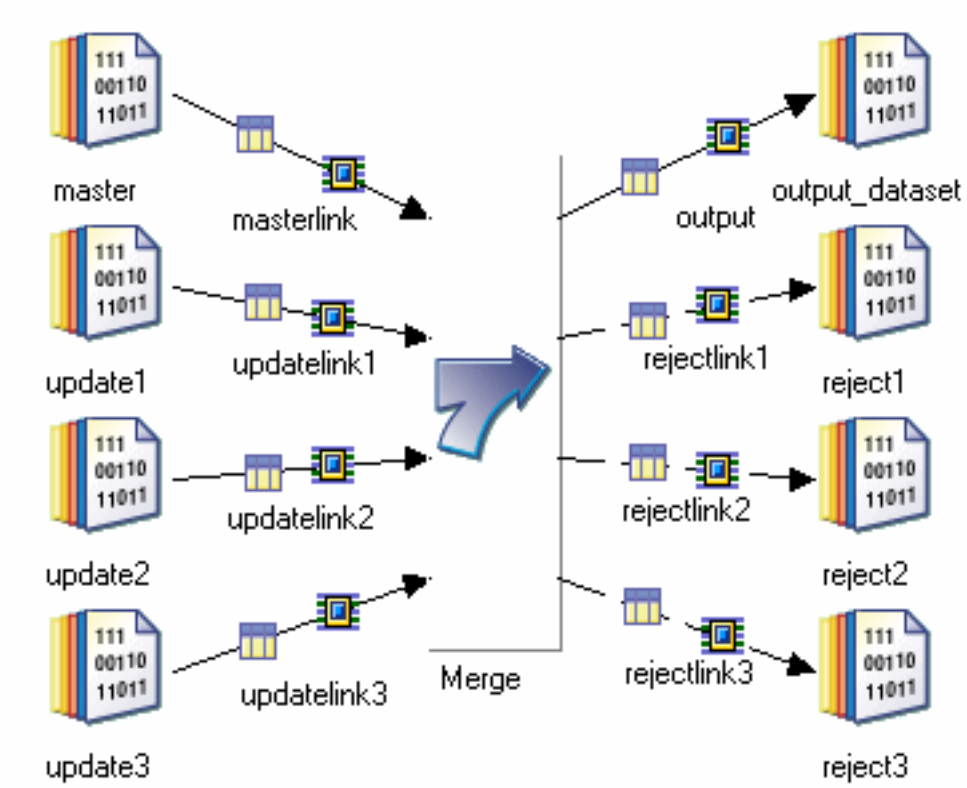
The Merge stage is one of three stages that join tables based on the values of key columns. The other two are:

- “Join stage” on page 238
- “Lookup Stage” on page 254

The three stages differ mainly in the memory they use, the treatment of rows with unmatched keys, and their requirements for data being input (for example, whether it is sorted).

The Merge stage combines a master data set with one or more update data sets. The columns from the records in the master and update data sets are merged so that the output record contains all the columns from the master record plus any additional columns from each update record that are required. A master

record and an update record are merged only if both of them have the same values for the merge key column(s) that you specify. Merge key columns are one or more columns that exist in both the master and update records.



The data sets input to the Merge stage must be key partitioned and sorted. This ensures that rows with the same key column values are located in the same partition and will be processed by the same node. It also minimizes memory requirements because fewer rows need to be in memory at any one time. Choosing the auto partitioning method will ensure that partitioning and sorting is done. If sorting and partitioning are carried out on separate stages before the Merge stage, InfoSphere DataStage in auto partition mode will detect this and not repartition (alternatively you could explicitly specify the Same partitioning method).

As part of preprocessing your data for the Merge stage, you should also 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. See “Remove Duplicates Stage” on page 291 for information about the Remove Duplicates stage.

Unlike Join stages and Lookup stages, the Merge stage allows you to specify several reject links. You can route update link rows that fail to match a master row down a reject link that is specific for that link. You must have the same number of reject links as you have update links. The **Link Ordering** tab on the Stage page lets you specify which update links send rejected rows to which reject links. You can also specify whether to drop unmatched master rows, or output them on the output data link.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data sets being merged.
- **Output Page**. This is where you specify details about the merged data being output from the stage and about the reject links.

Example merge

This example shows what happens to a master data set and two update data sets when they are merged. The key field is Horse, and all the data sets are sorted in descending order. Here is the master data set:

Table 27. Master data set

Horse	Freezemark	Mchip	Reg_Soc	Level
William	DAM7	N/A	FPS	Adv
Robin	DG36	N/A	FPS	Nov
Kayser	N/A	N/A	AHS	N/A
Heathcliff	A1B1	N/A	N/A	Adv
Fairfax	N/A	N/A	FPS	N/A
Chaz	N/A	a296100da	AHS	Inter

Here is the Update 1 data set:

Table 28. Update 1 data set

Horse	vacc.	last_worm
William	07.07.02	12.10.02
Robin	07.07.02	12.10.02
Kayser	11.12.02	12.10.02
Heathcliff	07.07.02	12.10.02
Fairfax	11.12.02	12.10.02
Chaz	10.02.02	12.10.02

Here is the Update 2 data set:

Table 29. Update 2 data set

Horse	last_trim	shoes
William	11.05.02	N/A
Robin	12.03.02	refit
Kayser	11.05.02	N/A
Heathcliff	12.03.02	new
Fairfax	12.03.02	N/A
Chaz	12.03.02	new

Here is the merged data set output by the stage:

Table 30. Merged data set

Horse	Freeze mark	Mchip	Reg. Soc	Level	vacc.	last worm	last trim	shoes
William	DAM7	N/A	FPS	Adv	07.07.02	12.10.02	11.05.02	N/A
Robin	DG36	N?A	FPS	Nov	07.07.02	12.10.02	12.03.02	refit
Kayser	N/A	N/A	AHS	Nov	11.12.02	12.10.02	11.05.02	N/A
Heathcliff	A1B1	N/A	N/A	Adv	07.07.02	12.10.02	12.03.02	new

Table 30. Merged data set (continued)

Horse	Freeze mark	Mchip	Reg. Soc	Level	vacc.	last worm	last trim	shoes
Fairfax	N/A	N/A	FPS	N/A	11.12.02	12.10.02	12.03.02	N/A
Chaz	N/A	a2961da	AHS	Inter	10.02.02	12.10.02	12.03.02	new

Merge stage: fast path

About this task

InfoSphere DataStage has many defaults which means that Merges can be simple to set up. This section specifies the minimum steps to take to get a Merge stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

- In the Stage page **Properties Tab** specify the key column or columns that the Merge will be performed on.
- In the Stage page **Properties Tab** set the Unmatched Masters Mode, Warn on Reject Updates, and Warn on Unmatched Masters options or accept the defaults.
- In the Stage page **Link Ordering Tab**, check that your input links are correctly identified as "master" and "update(s)", and your output links are correctly identified as "master" and "update reject". Reorder if required.
- Ensure required column meta data has been specified (this might be done in another stage, or might be omitted altogether if you are relying on Runtime Column Propagation).
- In the Output Page **Mapping Tab**, specify how the columns from the input links map onto output columns.

Merge stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Merge stage: Properties tab

The Properties tab allows you to specify properties that determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 31. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Merge Keys/Key	Input Column	N/A	Y	Y	N/A
Merge Keys/Sort Order	Ascending/ Descending	Ascending	Y	N	Key
Merge Keys/Nulls position	First/Last	First	N	N	Key
Merge Keys/Sort as EBCDIC	True/False	False	N	N	Key

Table 31. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Merge Keys/Case Sensitive	True/False	True	N	N	Key
Options/ Unmatched Masters Mode	Keep/Drop	Keep	Y	N	N/A
Options/Warn On Reject Masters	True/False	True	Y	N	N/A
Options/Warn On Reject Updates	True/False	True	Y	N	N/A

Merge stage: Merge keys category:
Key

This specifies the key column you are merging on. Repeat the property to specify multiple keys. You can use the Column Selection dialog box to select several keys at once if required. Key has the following dependent properties:

- **Sort Order**

  Choose Ascending or Descending. The default is Ascending.
- **Nulls position**

  By default columns containing null values appear first in the merged data set. To override this default so that columns containing null values appear last in the merged data set, select Last.
- **Sort as EBCDIC**

  To sort as in the EBCDIC character set, choose True.
- **Case Sensitive**

  Use this to specify whether each merge key is case sensitive or not, this is set to True by default, that is, the values "CASE" and "case" would not be judged equivalent.

Merge stage: Options category:
Unmatched Masters Mode

Set to Keep by default. It specifies that unmatched rows from the master link are output to the merged data set. Set to Drop to specify that rejected records are dropped instead.

Warn On Reject Masters

Set to True by default. This will warn you when bad records from the master link are rejected. Set it to False to receive no warnings.

Warn On Reject Updates

Set to True by default. This will warn you when bad records from any update links are rejected. Set it to False to receive no warnings.

Merge stage: Advanced Tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts the setting which results from ORing the settings of the input stages, that is, if any of the input stages uses **Set** then this stage will use **Set**. You can explicitly select **Set** or **Clear**. Select **Set** to request the next stage in the job attempts to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Merge stage: Link Ordering tab

This tab allows you to specify which of the input links is the master link and the order in which links input to the Merge stage are processed. You can also specify which of the output links is the master link, and which of the reject links corresponds to which of the incoming update links.

By default the links will be processed in the order they were added. To rearrange them, choose an input link and click the up arrow button or the down arrow button.

Merge stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Merge stage uses this when it is determining the order of the key fields. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Merge stage: Input page

The Input page allows you to specify details about the data coming in to be merged. Choose an input link from the **Input name** drop down list to specify which link you want to work on.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Merge stage partitioning are given in the following section. See , "Stage Editors," for a general description of the other tabs.

Merge stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the merge is performed.

By default the stage uses the auto partitioning method. If the Preserve Partitioning option has been set on the previous stage in the job, this stage will warn you if it cannot preserve the partitioning of the incoming data. Auto mode ensures that data being input to the Merge stage is key partitioned and sorted.

If the Merge stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Merge stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Merge stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Merge stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default collection method for the Merge stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Merge stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available. In the case of a Merge stage, Auto will also ensure that the collected data is sorted.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the merge is performed (you might use this if you have selected a partitioning method other than auto or same). The sort is always carried out within data partitions. If the stage is partitioning incoming

data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Merge stage: Output page

The Output page allows you to specify details about data output from the Merge stage. The Merge stage can have only one master output link carrying the merged data and a number of reject links, each carrying rejected records from one of the update links. Choose an output link from the **Output name** drop down list to specify which link you want to work on.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of incoming data. The Mapping tab allows you to specify the relationship between the columns being input to the Merge stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output links.

Details about Merge stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Merge stage: Reject links

You cannot change the properties of a Reject link. They have the meta data of the corresponding incoming update link and this cannot be altered.

Merge stage: Mapping tab

For Merge stages the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them.

The left pane shows the columns of the merged data. These are read only and cannot be modified on this tab. This shows the meta data from the master input link and any additional columns carried on the update links.

The right pane shows the output columns for the master output link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility.

In the above example the left pane represents the incoming data after the merge has been performed. The right pane represents the data being output by the stage after the merge operation. In this example the data has been mapped straight across.

Lookup Stage

The Lookup stage is a processing stage. It is used to perform lookup operations on a data set read into memory from any other Parallel job stage that can output data. It can also perform lookups directly in a DB2 or Oracle database (see Connectivity Guides for these two databases) or in a lookup table contained in a Lookup File Set stage (see “Lookup file set stage” on page 133)

The most common use for a lookup is to map short codes in the input data set onto expanded information from a lookup table which is then joined to the incoming data and output. For example, you could have an input data set carrying names and addresses of your U.S. customers. The data as presented identifies state as a two letter U. S. state postal code, but you want the data to carry the full name of the state. You could define a lookup table that carries a list of codes matched to states, defining the code as the key column. As the Lookup stage reads each line, it uses the key to look up the state in the lookup table. It adds the state to a new column defined for the output link, and so the full state name is added to each address. If any state codes have been incorrectly entered in the data set, the code will not be found in the lookup table, and so that record will be rejected.

Lookups can also be used for validation of a row. If there is no corresponding entry in a lookup table to the key's values, the row is rejected.

The Lookup stage is one of three stages that join tables based on the values of key columns. The other two are:

- Join stage - “Join stage” on page 238
- Merge stage - “Merge Stage” on page 246

The three stages differ mainly in the memory they use, the treatment of rows with unmatched keys, and their requirements for data being input (for example, whether it is sorted). See "Lookup Versus Join" for help in deciding which stage to use.

The Lookup stage can have a reference link, a single input link, a single output link, and a single rejects link. Depending upon the type and setting of the stage(s) providing the look up information, it can have multiple reference links (where it is directly looking up a DB2 table or Oracle table, it can only have a single reference link). A lot of the setting up of a lookup operation takes place on the stage providing the lookup table.

The input link carries the data from the source data set and is known as the *primary* link. The following pictures show some example jobs performing lookups.

For each record of the source data set from the primary link, the Lookup stage performs a table lookup on each of the lookup tables attached by reference links. The table lookup is based on the values of a set of *lookup key* columns, one set for each table. The keys are defined on the Lookup stage. For lookups of data accessed through the Lookup File Set stage, the keys are specified when you create the look up file set.

You can specify a condition on each of the reference links, such that the stage will only perform a lookup on that reference link if the condition is satisfied.

Lookup stages do not require data on the input link or reference links to be sorted. Be aware, though, that large in-memory lookup tables will degrade performance because of their paging requirements.

Each record of the *output* data set contains columns from a source record plus columns from all the corresponding lookup records where corresponding source and lookup records have the same value for the lookup key columns. The lookup key columns do not have to have the same names in the primary and the reference links.

The optional *reject* link carries source records that do not have a corresponding entry in the input lookup tables.

You can also perform a *range lookup*, which compares the value of a source column to a range of values between two lookup table columns. If the source column value falls within the required range, a row is passed to the output link. Alternatively, you can compare the value of a lookup column to a range of values between two source columns. Range lookups must be based on column values, not constant values. Multiple ranges are supported.

There are some special partitioning considerations for Lookup stages. You need to ensure that the data being looked up in the lookup table is in the same partition as the input data referencing it. One way of doing this is to partition the lookup tables using the Entire method. Another way is to partition it in the same way as the input data (although this implies sorting of the data).

Unlike most of the other stages in a Parallel job, the Lookup stage has its own user interface. It does not use the generic interface as described in Chapter 4, "Stage editors," on page 49.

When you edit a Lookup stage, the Lookup Editor appears. The left pane represents input data and lookup data, and the right pane represents output data.

Lookup Versus Join

InfoSphere DataStage does not know how large your data is, so cannot make an informed choice whether to combine data using a Join stage or a Lookup stage. Here's how to decide which to use:

There are two data sets being combined. One is the primary or driving data set, sometimes called the left of the join. The other data set(s) are the reference data sets, or the right of the join.

In all cases the size of the reference data sets is a concern. If these take up a large amount of memory relative to the physical RAM memory size of the computer you are running on, then a Lookup stage might thrash because the reference data sets might not fit in RAM along with everything else that has to be in RAM. This results in very slow performance since each lookup operation can, and typically does, cause a page fault and an I/O operation.

So, if the reference data sets are big enough to cause trouble, use a join. A join does a high-speed sort on the driving and reference data sets. This can involve I/O if the data is big enough, but the I/O is all highly optimized and sequential. After the sort is over, the join processing is very fast and never involves paging or other I/O.

Example Look Up

This example shows what happens when data is looked up in a lookup table. The stage in this case will look up the interest rate for each customer based on the account type. Here is the data that arrives on the primary link:

Table 32. Input data

Customer	accountNo	accountType	balance
Latimer	7125678	plat	7890.76
Ridley	7238892	flexi	234.88
Cranmer	7611236	gold	1288.00
Hooper	7176672	flexi	3456.99
Moore	7146789	gold	424.76

Here is the data in the lookup table:

accountType
InterestRate

bronze
1.25

silver 1.50

gold 1.75

plat 2.00

flexi 1.88

fixterm
3.00

Here is what the lookup stage will output:

Table 33. Stage output

Customer	accountNo	accountType	balance	InterestRate
Latimer	7125678	plat	7890.76	2.00
Ridley	7238892	flexi	234.88	1.88
Cranmer	7611236	gold	1288.00	1.75
Hooper	7176672	flexi	3456.99	1.88
Moore	7146789	gold	424.76	1.75

Here is a job that performs this simple lookup:

The accounts data set holds the details of customers and their account types, the interest rates are held in an Oracle table. The lookup stage is set as follows:

All the columns in the accounts data set are mapped over to the output link. The AccountType column in the accounts data set has been joined to the AccountType column of the interest_rates table. For each row, the AccountType is looked up in the interest_rates table and the corresponding interest rate is returned.

The reference link has a condition on it. This detects if the balance is null in any of the rows of the accounts data set. If the balance is null the row is sent to the rejects link (the rejects link does not appear in the lookup editor because there is nothing you can change).

Lookup stage: fast path

About this task

InfoSphere DataStage has many defaults which means that lookups can be simple to set up. This section specifies the minimum steps to take to get a Lookup stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

The exact steps you need to take when setting up a Lookup stage depend on what type of lookup table you are looking up.

Using In-Memory Lookup tables

About this task

If you are accessing a lookup table read into memory from some other stage, you need to do the following:

In the Data Input source stage:

- Specify details about the data source (for example, if using a File Set stage, give the name of the File Set).
- Ensure required column meta data has been specified.
- Fulfil any "must do's" for that particular stage editor.

In the stage providing the lookup table:

- Ensure required column meta data has been specified.
- Fulfil any "must do's" for that particular stage editor.

In the Lookup stage:

- Map the required columns from your data input link to the output link (you can drag them or copy and paste them).
- Map the required columns from your lookup table or tables to the output link (again you can drag them or copy and paste them).
- Specify the key column or columns which are used for the lookup. Do this by dragging - or copying and pasting - key columns from the data link to the **Key Expression** field in the lookup table link. Note that key expressions can only be specified on key fields (that is, columns that have the key field selected in the column definitions). If you drag a column that is not currently defined as a key, you are asked if you want to make it one. If you want the comparison performed on this column to ignore case, then select the **Caseless** check box.

If you want to impose conditions on your lookup, or want to use a reject link, you need to double-click on the Condition header of a reference link, choose **Conditions** from the link shortcut menu, or click the **Condition** toolbar button. The **Lookup Stage Conditions** dialog box appears. This allows you to:

- Specify that one of the reference links is allowed to return multiple rows when performing a lookup without causing an error (choose the relevant reference link from the **Multiple rows returned from link** list).
- Specify a condition for the required references. Double-click the Condition box (or press CTRL-E) to open the expression editor. This expression can access all the columns of the primary link, plus columns in reference links that are processed before this link.
- Specify what happens if the condition is not met on each link.
- Specify what happens if a lookup fails on each link.

If you want to specify a range lookup, select the **Range** check box on the stream link or select **Range** from the **Key Type** list on the reference link. Then double-click the **Key Expression** field to open the Range dialog box, where you can build the range expression.

Next you need to open the **Stage Properties** dialog box for the Lookup stage. Do this by choosing the Stage Properties icon from the stage editor toolbar, or by choosing **Stage Properties** or **Link Properties** from the stage editor shortcut menu (choosing **Link Properties** will open the dialog box with the link you are looking at selected, otherwise you might need to choose the correct link from the **Input name** or **Output name** list).

- In the Stage page **Link Ordering Tab**, check that your links are correctly identified as "primary" and "lookup(s)", and reorder if required (the links will be shown in the new order on the Lookup canvas).
- Unless you have particular partitioning requirements, leave the default auto setting on the **Input Page Partitioning Tab**.

Using Oracle or DB2 Databases Directly

About this task

If you are doing a direct lookup in an Oracle or DB2 database table (known as `sparse' mode), you need to do the following:

In the Data Input source stage:

- Specify details about the data source (for example, if using a File Set stage, give the name of the File Set).
- Ensure required column meta data has been specified (this can be done in another stage).
- Fulfil any "must do's" for that particular stage editor.

In the Oracle or DB2/UDB Enterprise Stage:

- Set the Lookup Type to sparse. If you don't do this the lookup will operate as an in-memory lookup.
- Specify required details for connecting to the database table.
- Ensure required column meta data has been specified (this can be omitted altogether if you are relying on Runtime Column Propagation).

See the Connectivity Guides for these two databases for more details.

In the Lookup stage:

- Map the required columns from your data input link to the output link (you can drag them or copy and paste them).
- Map the required columns from your lookup table or tables to the output link (again you can drag them or copy and paste them).

If you want to impose conditions on your lookup, or want to use a reject link, you need to double-click on the Condition header, choose **Conditions** from the link shortcut menu, or click the **Condition** toolbar icon. The **Lookup Stage Conditions** dialog box appears. This allows you to:

- Specify what happens if a lookup fails on this link.

If you want to specify a range lookup, select the **Range** check box on the stream link or select **Range** from the **Key Type** list on the reference link. Then double-click the **Key Expression** field to open the Range dialog box, where you can build the range expression.

Next you need to open the **Stage Properties** dialog box for the Lookup stage. Do this by choosing the Stage Properties icon from the stage editor toolbar, or by choosing **Stage Properties** or **Link Properties** from the stage editor shortcut menu (choosing **Link Properties** will open the dialog box with the link you are looking at selected, otherwise you might need to choose the correct link from the **Input name** or **Output name** list).

- In the Stage page **Link Ordering Tab**, check that your links are correctly identified as "primary" and "lookup(s)", and reorder if required.
- Unless you have particular partitioning requirements, leave the default auto setting on the **Input Page Partitioning Tab**.

Using Lookup File Set

About this task

If you are accessing a lookup table held in a lookup file set that you have previously created using InfoSphere DataStage, you need to do the following:

In the Data Input source stage:

- Specify details about the data source (for example, if using a File Set stage, give the name of the file set).
- Ensure required column meta data has been specified.
- Fulfil any "must do's" for that particular stage editor.

In the Lookup File Set stage:

- Specify the name of the file set holding the lookup table.
- If you are performing a range lookup, specify the upper and lower bound columns (or the bounded column if the lookup is reversed).
- Make sure that the key column or columns were specified when the file set holding the lookup table was created.
- Ensure required column meta data has been specified.

See “Lookup file set stage” on page 133 for details about the Lookup File Set stage.

In the Lookup stage:

- Map the required columns from your data input link to the output link (you can drag them or copy and paste them).
- Map the required columns from your lookup table or tables to the output link (again you can drag them or copy and paste them).

As you are using a lookup file set this is all the mapping you need to do, the key column or columns for the lookup is defined when you create the lookup file set.

If you are performing a range lookup, select the **Range** check box on the stream link or select **Range** from the **Key Type** list on the reference link. Then double-click the **Key Expression** field to open the Range dialog box, where you can build the range expression.

Next you need to open the **Stage Properties** dialog box for the Lookup stage. Do this by choosing the Stage Properties icon from the stage editor toolbar, or by choosing **Stage Properties** or **Link Properties** from the stage editor shortcut menu (choosing **Link Properties** will open the dialog with the link you are looking at selected, otherwise you might need to choose the correct link from the **Input name** or **Output name** list).

- In the Stage page **Link Ordering Tab**, check that your links are correctly identified as "primary" and "lookup(s)", and reorder if required.
- Unless you have particular partitioning requirements, leave the default auto setting on the **Input Page Partitioning Tab**.

Lookup Editor Components

The Lookup Editor has the following components.

Lookup stage: Toolbar

The Lookup toolbar contains the following buttons (from left to right):

- Stage properties
- Constraints
- Show all
- Show/hide stage variables
- Cut
- Copy
- Paste
- Find/replace
- Load column definition
- Save column definition
- Column auto-match
- Input link execution order
- Output link execution order

Lookup stage: link area

The top area displays links to and from the Lookup stage, showing their columns and the relationships between them.

The link area is divided into two panes; you can drag the splitter bar between them to resize the panes relative to one another. There is also a horizontal scroll bar, allowing you to scroll the view left or right.

The left pane shows the input link, the right pane shows output links. Output columns that have an invalid derivation defined are shown in red. Reference link input key columns with invalid key expressions are also shown in red.

Within the Lookup Editor, a single link can be selected at any one time. When selected, the link's title bar is highlighted, and arrowheads indicate any selected columns within that link.

Lookup stage: Meta Data Area

The bottom area shows the column meta data for input and output links. Again this area is divided into two panes: the left showing input link meta data and the right showing output link meta data.

The meta data for each link is shown in a grid contained within a tabbed page. Click the tab to bring the required link to the front. That link is also selected in the link area.

If you select a link in the link area, its meta data tab is brought to the front automatically.

You can edit the grids to change the column meta data on any of the links. You can also add and delete meta data.

As with column meta data grids on other stage editors, edit row in the context menu allows editing of the full meta data definitions (see "Columns Tab").

Lookup stage: Shortcut Menus

The Lookup Editor shortcut menus are displayed by right-clicking the links in the links area.

There are slightly different menus, depending on whether you right-click an input link, or an output link. The input link menu offers you operations on input columns, the output link menu offers you operations on output columns and their derivations.

The shortcut menu enables you to:

- Open the Stage Properties dialog box in order to specify stage or link properties.
- Open the Lookup Stage Conditions dialog box to specify a conditional lookup.
- Open the **Column Auto Match** dialog box.
- Display the **Find/Replace** dialog box.
- Display the Select dialog box.
- Validate, or clear a derivation.
- Append a new column to the selected link.
- Select all columns on a link.
- Insert or delete columns.
- Cut, copy, and paste a column or a key expression or a derivation.

If you display the menu from the links area background, you can:

- Open the Stage Properties dialog box in order to specify stage or link properties.

- Open the Lookup Stage Conditions dialog box to specify a conditional lookup.
- Open the Link Execution Order dialog box in order to specify the order in which links should be processed.
- Toggle between viewing link relations for all links, or for the selected link only.

Right-clicking in the meta data area of the Lookup Editor opens the standard grid editing shortcut menus.

Editing Lookup Stages

The Lookup Editor enables you to perform the following operations on a Lookup stage:

- Create new columns on a link
- Delete columns from within a link
- Move columns within a link
- Edit column meta data
- Specify key expressions
- Map input columns to output columns

Using Drag and Drop

Many of the Lookup stage edits can be made simpler by using the Lookup Editor's drag-and-drop functionality.

About this task

You can drag columns from any link to any other link. Common uses are:

- Copying input columns to output links
- Moving columns within a link
- Setting derivation or key expressions

Procedure

1. Click the source cell to select it.
2. Click the selected cell again and, without releasing the mouse button, drag the mouse pointer to the desired location within the target link. An insert point appears on the target link to indicate where the new cell will go. This can be to create a new column, or set a derivation. The exact action depends on where you drop.
3. Release the mouse button to drop the selected cell.

Results

You can drag multiple columns, key expressions, or derivations. Use the standard Explorer keys when selecting the source column cells, then proceed as for a single cell.

You can drag and drop the full column set by dragging the link title.

Find and Replace Facilities

About this task

If you are working on a complex job where several links, each containing several columns, go in and out of the Lookup stage, you can use the find/replace column facility to help locate a particular column or expression and change it.

The find/replace facility enables you to:

- Find and replace a column name

- Find and replace expression text
- Find the next empty expression
- Find the next expression that contains an error

To use the find/replace facilities, do one of the following:
- Click the **Find/Replace** button on the toolbar
- Choose **Find/Replace** from the link shortcut menu
- Press **Ctrl-F**

The Find and Replace dialog box appears. It has three tabs:
- **Expression Text**. Allows you to locate the occurrence of a particular string within an expression, and replace it if required. You can search up or down, and choose to match case, match whole words, or neither. You can also choose to replace all occurrences of the string within an expression.
- **Column Names**. Allows you to find a particular column and rename it if required. You can search up or down, and choose to match case, match the whole word, or neither.
- **Expression Types**. Allows you to find the next empty expression or the next expression that contains an error. You can also press **Ctrl-M** to find the next empty expression or **Ctrl-N** to find the next erroneous expression.

  Note: The find and replace results are shown in the color specified in **Tools** > **Options**.

Press **F3** to repeat the last search you made without opening the **Find and Replace** dialog box.

Select Facilities

If you are working on a complex job where several links, each containing several columns, go in and out of the Lookup stage, you can use the select column facility to select multiple columns.

About this task

The select facility enables you to:
- Select all columns whose expressions contains text that matches the text specified.
- Select all columns whose name contains the text specified (and, optionally, matches a specified type).
- Select all columns with a certain data type.
- Select all columns with missing or invalid expressions.

To use the select facilities, choose **Select** from the link shortcut menu. The Select dialog box appears. It has three tabs:
- **Expression Text**. The **Expression Text** tab allows you to select all columns/stage variables whose expressions contain text that matches the text specified. The text specified is a simple text match, taking into account the **Match case** setting.
- **Column Names**. The **Column Names** tab allows you to select all column/stage variables whose Name contains the text specified. There is an additional **Data Type** list, that will limit the columns selected to those with that data type. You can use the **Data Type** list on its own to select all columns of a certain data type. For example, all string columns can be selected by leaving the text field blank, and selecting String as the data type. The data types in the list are generic data types, where each of the column SQL data types belong to one of these generic types.
- **Expression Types**. The **Expression Types** tab allows you to select all columns with either empty expressions or invalid expressions.

Creating and Deleting Columns
About this task

You can create columns on links to the Lookup stage using any of the following methods:

- Select the link, then click the **Load Column Definition** button in the toolbar to open the standard load columns dialog box.
- Use drag-and-drop or copy and paste functionality to create a new column by copying from an existing column on another link.
- Use the shortcut menus to create a new column definition.
- Edit the grids in the link's meta data tab to insert a new column.

When copying columns, a new column is created with the same meta data as the column it was copied from.

To delete a column from within the Lookup Editor, select the column you want to delete and click the **cut** button or choose **Delete Column** from the shortcut menu.

Moving Columns Within a Link
About this task

You can move columns within a link using either drag-and-drop or cut and paste. Select the required column, then drag it to its new location, or cut it and paste it in its new location.

Editing Column Meta Data
About this task

You can edit column meta data from within the grid in the bottom of the Lookup Editor. Select the tab for the link meta data that you want to edit, then use the standard InfoSphere DataStage edit grid controls.

The meta data shown does not include column derivations since these are edited in the links area.

Defining Output Column Derivations

You can define the derivation of output columns from within the Lookup Editor in a number of ways.

About this task

Choose one of the following ways to define the derivation of output columns from within the Lookup Editor:

- To map an input column (from data input or reference input) onto an output column you can use drag-and-drop or copy and paste to copy an input column to an output link. The output columns will have the same names as the input columns from which they were derived.
- If the output column already exists, you can drag or copy an input column to the output column's **Derivation** field. This specifies that the column is directly derived from an input column, with no transformations performed.
- You can use the column auto-match facility to automatically set that output columns are derived from their matching input columns.

If a derivation is displayed in red (or the color defined in **Tools** > **Options**), it means that the Lookup Editor considers it incorrect. To see why it is invalid, choose **Validate Derivation** from the shortcut menu.

After an output link column has a derivation defined that contains any input link columns, then a relationship line is drawn between the input column and the output column. There can be one or more relationship lines between columns. You can choose whether to view the relationships for all links, or just

the relationships for the selected links, using the button in the toolbar.

Column Auto-Match Facility:

You can specify to use the column auto-match facility, a feature that automatically sets columns on an output link to be derived from matching columns on an input link.

About this task

This time-saving feature allows you to automatically set columns on an output link to be derived from matching columns on an input link. Using this feature you can fill in all the output link derivations to route data from corresponding input columns, then go back and edit individual output link columns where you want a different derivation.

Procedure

1. Do one of the following:
   - Click the **Auto-match** button in the Lookup Editor toolbar.
   - Choose **Auto-match** from the input link header or output link header shortcut menu.

   TheColumn Auto-Match dialog box appears.

   Choose the output link that you want to match columns with the input link from the list.
2. Click **Location match** or **Name match** from the **Match type** area.

   If you choose **Location match**, this will set output column derivations to the input link columns in the equivalent positions. It starts with the first input link column going to the first output link column, and works its way down until there are no more input columns left.
3. Click **OK** to proceed with the auto-matching.

   Note: Auto-matching does not take into account any data type incompatibility between matched columns; the derivations are set regardless.

Defining Input Column Key Expressions

About this task

You can define key expressions for key fields of reference inputs. This is similar to defining derivations for output columns.

The key expression is an equijoin from a primary input link column. You can specify it in two ways:

- Use drag-and-drop to drag a primary input link column to the appropriate key expression cell.
- Use copy and paste to copy a primary input link column and paste it on the appropriate key expression cell.

A relationship link is drawn between the primary input link column and the key expression.

You can also use drag-and-drop or copy and paste to copy an existing key expression to another input column, and you can drag or copy multiple selections.

If a key expression is displayed in red (or the color defined in **Tools** > **Options**), it means that the Transformer Editor considers it incorrect. To see why it is invalid, choose **Validate Derivation** from the shortcut menu.

Lookup Stage Properties

The Lookup stage has a Properties dialog box which allows you to specify details about how the stage operates.

The Lookup Stage Properties dialog box has three pages:

- **Stage Page**. This is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data input to the Lookup stage.
- **Output Page**. This is where you specify details about the output links from the Lookup stage.

Lookup stage: Stage Page

The **General** tab allows you to specify an optional description of the stage. The **Advanced** tab allows you to specify how the stage executes. The **Link Ordering** tab allows you to specify which order the input links are processed in. The **NLS Locale** tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules. The **Build** tab allows you to override the default compiler and linker flags for this particular stage.

Lookup stage: Advanced Tab:

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the **Advanced** tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts the setting of the previous stage on the stream link. You can explicitly select **Set** or **Clear**. Select **Set** to request the next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Lookup stage: Link Ordering Tab:

This tab allows you to specify which input link is the primary link and the order in which the reference links are processed.

By default the input links will be processed in the order they were added. To rearrange them, choose an input link and click the up arrow button or the down arrow button.

You can also access this tab by clicking the input link order button in the toolbar, or by choosing **Reorder input links** from the shortcut menu.

Lookup stage: NLS Locale Tab:

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Lookup stage uses this when it is determining the order of the key fields. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Lookup stage: Build Tab:

In some cases the Lookup stage might use C++ code to implement your lookup. In this case, you can use the **Build** tab to override the compiler and linker flags that have been set for the job or project. The flags you specify here will take effect for this stage and this stage alone. The flags available are platform and compiler-dependent.

Lookup stage: Input Page

The **Input page** allows you to specify details about the incoming data set and the reference links. Choose a link from the **Input name** list to specify which link you want to work on.

The **General** tab allows you to specify an optional description of the link. When you are performing an in-memory lookup, the **General** tab has two additional fields:

- **Save to lookup fileset**. Allows you to specify a lookup file set to save the look up data.
- **Diskpool**. Specify the name of the disk pool into which to write the file set. You can also specify a job parameter.

The **Partitioning** tab allows you to specify how incoming data on the source data set link is partitioned. The **Advanced** tab allows you to change the default buffering settings for the input link.

Details about Lookup stage partitioning are given in the following section. See Chapter 4, “Stage editors,” on page 49 for a general description of the other tabs.

Lookup stage: Partitioning Tab:

The **Partitioning** tab allows you to specify details about how the incoming data is partitioned or collected before the lookup is performed. It also allows you to specify that the data should be sorted before the lookup. Note that you cannot specify partitioning or sorting on the reference links, this is specified in their source stage.

By default the stage uses the auto partitioning method. If the Preserve Partitioning option has been set on the previous stage in the job the stage will warn you when the job runs if it cannot preserve the partitioning of the incoming data.

If the Lookup stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The **Partitioning** tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Lookup stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Lookup stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** list. This will override any current partitioning.

You might need to ensure that your lookup tables have been partitioned using the Entire method, so that the lookup tables will always contain the full set of data that might need to be looked up. For lookup files and lookup tables being looked up in databases, the partitioning is performed on those stages.

If the Lookup stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Lookup stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Lookup stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The **Partitioning** tab also allows you to specify that data arriving on the input link should be sorted before the lookup is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data, the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Lookup stage: Output Page

The **Output page** allows you to specify details about data output from the Lookup stage. The Lookup stage can have only one output link. It can also have a single reject link, where records can be sent if the lookup fails. The **Output Link** list allows you to choose whether you are looking at details of the main output link (the stream link) or the reject link.

The **General** tab allows you to specify an optional description of the output link. The **Advanced** tab allows you to change the default buffering settings for the output links.

Lookup stage: Reject Links:

You cannot set the mapping or edit the column definitions for a reject link. The link uses the column definitions for the primary input link.

Lookup Stage Conditions

The Lookup stage has a **Lookup Stage Conditions** dialog box that allows you to specify:

- Which reference link (if any) can return multiple rows from a lookup.
- A condition that should be fulfilled before a lookup is performed on a reference link.
- What action should be taken if a condition on a reference link is not met.
- What action should be taken if a lookup on a link fails.

You can open the **Lookup Stage Conditions** dialog box by:

- Double-clicking the Condition: bar on a reference link.
- Selecting **Conditions** from the background shortcut menu.
- Clicking the **Conditions** toolbar button.
- Selecting **Conditions** from the link shortcut menu.

To specify that a link can legitimately return multiple rows:

- Select the link name from the **Multiple rows returned from link** list (note that only one reference link in a Lookup stage is allowed to return multiple rows, and that this feature is only available for in-memory lookups).

To specify a condition for a reference link:

- Double-click the **Condition** field for the link you want to specify a condition for. The field expands to let you type in a condition, or click the browse button to open the expression editor to get help in specifying an expression. The condition should return a TRUE/FALSE result (for example DSLINK1.COL1 > 0).

To specify the action taken if the specified condition is not met:

- Choose an action from the **Condition Not Met** list. Possible actions are:
  - **Continue**. The fields from that link are set to NULL if the field is nullable, or to a default value if not. Continues processing any further lookups before sending the row to the output link.
  - **Drop**. Drops the row and continues with the next lookup.
  - **Fail**. Causes the job to issue a fatal error and stop.
  - **Reject**. Sends the row to the reject link.

To specify the action taken if a lookup on a link fails:

- Choose an action from the **Lookup Failure** list. Possible actions are:
  - **Continue**. The fields from that link are set to NULL if the field is nullable, or to a default value if not. Continues processing any further lookups before sending the row to the output link.
  - **Drop**. Drops the row and continues with the next lookup.

– **Fail**. Causes the job to issue a fatal error and stop.
– **Reject**. Sends the row to the reject link.

Range lookups

You can define a range lookup.

About this task

You can define a range lookup on the stream link or a reference link of a Lookup stage. On the stream link, the lookup compares the value of a source column to a range of values between two lookup columns. On the reference link, the lookup compares the value of a lookup column to a range of values between two source columns. Multiple ranges are supported.

Procedure

1. Select the column for the lookup:
   - To define the lookup on the stream link, select the **Range** check box next to the source column in the links area.
   - To define the lookup on a reference link, select the **Key** check box next to the reference column in the meta data area. In the links area, select **Range** from the **Key Type** list. The data on the reference link must be sorted.
2. Double-click the **Key Expression** field next to the selected column to open the Range dialog box.
3. Select a link from the **Lookup Link** list. (If you are defining the lookup on a reference link, the stream link appears by default.)
4. Define the range expression by selecting the upper bound and lower bound range columns and the required operators. For example:

   ```
   Account_Detail.Trans_Date >= Customer_Detail.Start_Date AND
   Account_Detail.Trans_Date <= Customer_Detail.End_Date
   ```

   As you build the expression, it appears in the **Expression** box.
5. Select **Caseless** if you want the lookup to ignore case.
6. Click **OK**.

The InfoSphere DataStage Expression Editor

The InfoSphere DataStage Expression Editor helps you to enter correct expressions when you edit Lookup stages. The Expression Editor can:

- Facilitate the entry of expression elements
- Complete the names of frequently used variables
- Validate the expression

The Expression Editor can be opened from:

- **Lookup Stage Conditions** dialog box

Expression Format

The format of an expression is as follows:

```
KEY:

something_like_this       is a token
something_in_italics      is a terminal, that is, does not break down any
   further
|     is a choice between tokens
[     is an optional part of the construction
"XXX"     is a literal token (that is, use XXX not
   including the quotes)
```

```
================================================
expression ::=    function_call |
      variable_name |
   other_name |
   constant |
   unary_expression |
   binary_expression |
   if_then_else_expression |
   substring_expression |
   "(" expression ")"

function_call ::= function_name "(" [argument_list] ")"
argument_list ::= expression | expression "," argument_list
function_name ::=    name of a built-in function |
   name of a user-defined_function
variable_name ::=       job_parameter name
other_name ::=    name of a built-in macro, system variable, and so on.
constant ::= numeric_constant | string_constant
numeric_constant ::= ["+" | "-"] digits ["." [digits]] ["E" | "e" ["+" | "-"] digits]
string_constant ::=    "'" [characters] "'" |
   """ [characters] """ |
   "\" [characters] "\"
unary_expression ::= unary_operator expression
unary_operator ::= "+" | "-"
binary_expression ::= expression binary_operator expression
binary_operator ::=    arithmetic_operator |
   concatenation_operator |
   matches_operator |
   relational_operator |
   logical_operator
arithmetic_operator ::= "+" | "-" | "*" | "/" | "^"
concatenation_operator ::= ":"
relational_operator ::=    "   =" |"EQ" |
      "<>" | "#" | "NE" |
      ">" | "GT" |
      ">=" | "=>" | "GE" |
      "<" | "LT" |
      "<=" | "=<" | "LE"
logical_operator ::= "AND" | "OR"
if_then_else_expression ::= "IF" expression "THEN" expression "ELSE" expression
substring_expression ::= expression "[" [expression ["," expression] "]"
field_expression ::=    expression "[" expression ","
   expression ","
   expression "]"
         /*    That is, always 3 args
Note: keywords like "AND" or "IF" or "EQ" might be in any case
```

Entering Expressions

About this task

Whenever the insertion point is in an expression box, you can use the Expression Editor to suggest the next element in your expression. Do this by right-clicking the box, or by clicking the **Suggest** button to the right of the box. This opens the **Suggest Operand** or **Suggest Operator** menu. Which menu appears depends on context, that is, whether you should be entering an operand or an operator as the next expression element. The Functions available from this menu are described in Appendix B, "Parallel Transform functions," on page 645. The DS macros are described in *InfoSphere DataStage Parallel Job Advanced Developer Guide*. You can also specify custom routines for use in the expression editor (see *InfoSphere DataStage Designer Client Guide*).

Completing Variable Names
About this task

The Expression Editor stores variable names. When you enter a variable name you have used before, you can type the first few characters, then press **F5**. The Expression Editor completes the variable name for you.

If you enter the name of the input link followed by a period, for example, **`DailySales.`**, the Expression Editor displays a list of the column names of the link. If you continue typing, the list selection changes to match what you type. You can also select a column name using the mouse. Enter a selected column name into the expression by pressing **Tab** or **Enter**. Press **Esc** to dismiss the list without selecting a column name.

Validating the Expression
About this task

When you have entered an expression in the Lookup Editor, press **Enter** to validate it. The Expression Editor checks that the syntax is correct and that any variable names used are acceptable to the compiler.

If there is an error, a message appears and the element causing the error is highlighted in the expression box. You can either correct the expression or close the Lookup Editor or Lookup dialog box.

For any expression, selecting Validate from its shortcut menu will also validate it and show any errors in a message box.

Exiting the Expression Editor
About this task

You can exit the Expression Editor in the following ways:

- Press **Esc** (which discards changes).
- Press **Return** (which accepts changes).
- Click outside the Expression Editor box (which accepts changes).

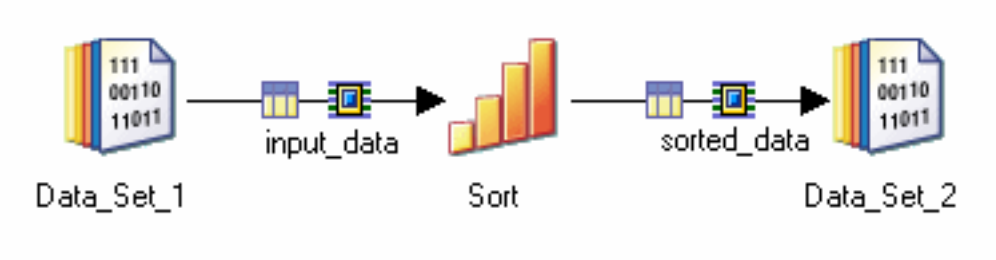
Configuring the Expression Editor
About this task

You can resize the Expression Editor window by dragging. The next time you open the expression editor in the same context (for example, editing output columns) on the same client, it will have the same size.

The Expression Editor is configured by editing the Designer options. This allows you to specify how `helpful' the expression editor is. For more information, see *InfoSphere DataStage Designer Client Guide*.

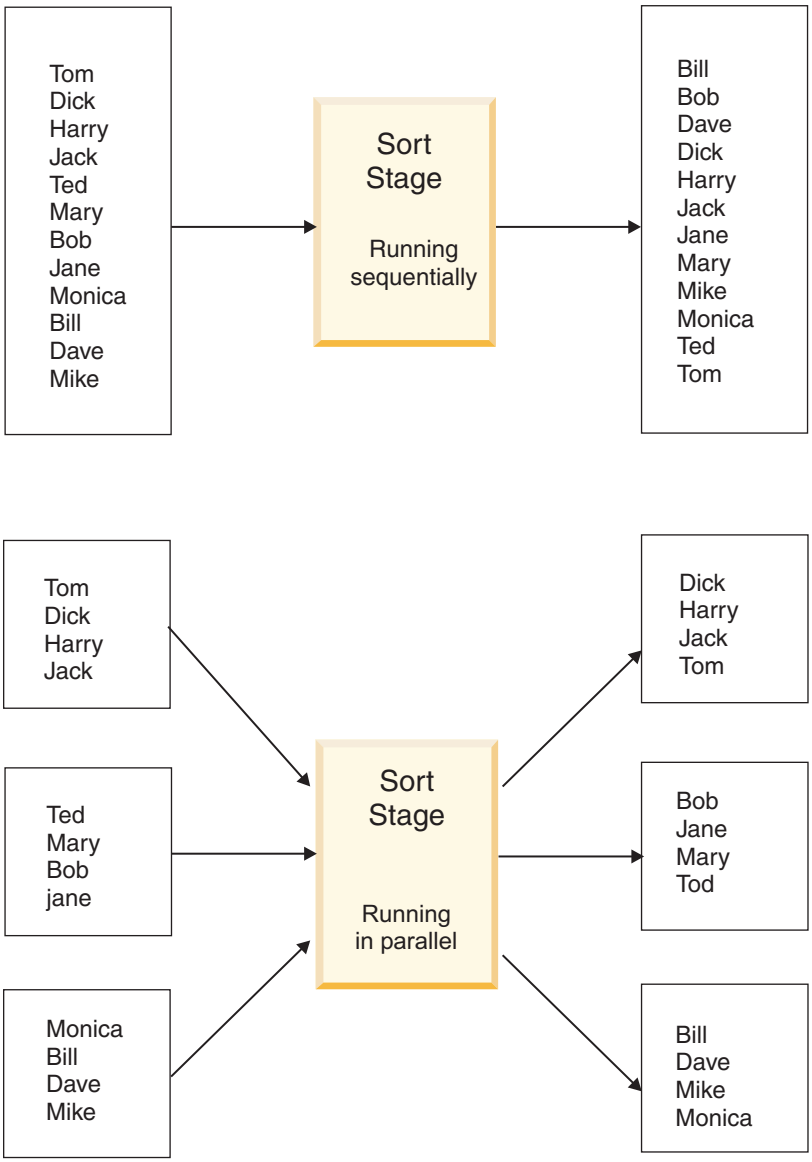
Sort stage

The Sort stage is a processing stage. It is used to perform more complex sort operations than can be provided for on the Input page Partitioning tab of parallel job stage editors. You can also use it to insert a more explicit simple sort operation where you want to make your job easier to understand. The Sort stage has a single input link which carries the data to be sorted, and a single output link carrying the sorted data.



You specify sorting keys as the criteria on which to perform the sort. A key is a column on which to sort the data, for example, if you had a name column you might specify that as the sort key to produce an alphabetical list of names. The first column you specify as a key to the stage is the primary key, but you can specify additional secondary keys. If multiple rows have the same value for the primary key column, then InfoSphere DataStage uses the secondary columns to sort these rows.

You can sort in sequential mode to sort an entire data set or in parallel mode to sort data within partitions, as shown below:



The stage uses temporary disk space when performing a sort. It 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 in the configuration file).
2. Scratch disks in the default disk pool (scratch disks are included here by default).
3. The directory specified by the TMPDIR environment variable.
4. The directory */tmp*.

You might perform a sort for several reasons. For example, you might want to sort a data set by a zip code column, 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 data set stage. The data set will therefore have four partitions. You then use that data set as input to a stage executing on a different number of nodes, possibly due to node constraints. InfoSphere DataStage automatically repartitions a data set to spread out the data set to all nodes in the system, unless you tell it not to, possibly destroying the sort

order of the data. You could avoid this by specifying the Same partitioning method. The stage does not perform any repartitioning as it reads the input data set; the original partitions are preserved.

You must also be careful when using a stage operating sequentially to process a sorted data set. A sequential stage executes on a single processing node to perform its action. Sequential stages will collect the data where the data set has more than one partition, which might also destroy the sorting order of its input data set. You can overcome this if you specify the collection method as follows:

- If the data was range partitioned before being sorted, you should use the ordered collection method 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 data was hash partitioned before being sorted, you should use the sort merge collection method specifying the same collection keys as the data was partitioned on.

  Note: If you write a sorted data set to an RDBMS there is no guarantee that it will be read back in the same order unless you specifically structure the SQL query to ensure this.

By default the stage will sort with the native InfoSphere DataStage sorter, but you can also specify that it uses the UNIX *sort* command.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data sets being sorted.
- **Output Page**. This is where you specify details about the sorted data being output from the stage.

Examples

Sequential Sort

This job sorts the contents of a sequential file, and writes it to a data set. The data is a list of GlobalCo customers sorted by customer number. We are going to sort it by customer name instead.

Here is a sample of the input data:

```
"JON SMITH","789 LEDBURY ROAD","GC13849","GlobalCoUS"
"MARY GARDENER","127 BORDER ST","GC13933","GlobalCoUS"
"CHRIS TRAIN","1400 NEW ST","GC14036","GlobalCoUS"
"HUW WILLIAMS","579 DIGBETH AVENUE","GC14127","GlobalCoUS"
"SARA PEARS","45 ALCESTER WAY","GC14263","GlobalCoUS"
"LUC TEACHER","3 BIRMINGHAM ROAD","GC14346","GlobalCoUS"
```

The meta data for the file is as follows:

Table 34. Example metadata for writing to a sequential file

Column name	Key	SQL Type	Length	Nullable
CUST_NAME		varchar	30	no
ADDR_1		varchar	30	no
CUSTOMER_NUMBER	yes	char	7	no
SOURCE		char	10	no

The Sequential File stage runs sequentially because it only has one source file to read. The Sort stage is set to run sequentially on the Stage page **Advanced** tab. The sort stage properties are used to specify the column CUST_NAME as the primary sort key:

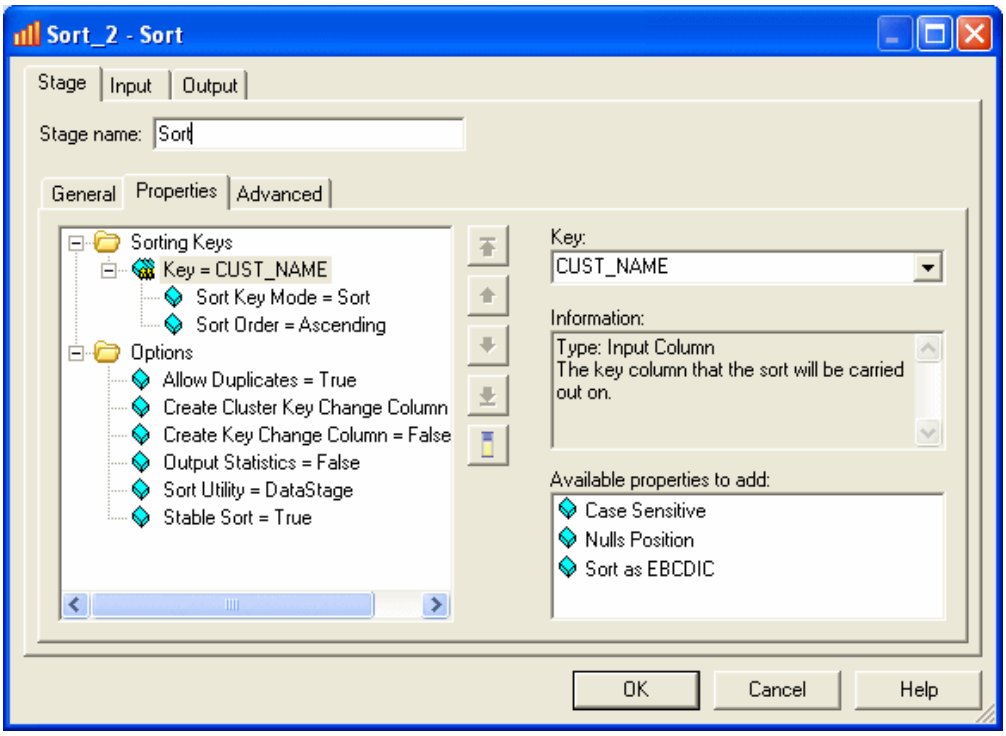


Figure 10. Properties tab

When the job is run the data is sorted into a single partition. The Data Set stage, GlobalCo_sorted, is set to run sequentially to write the data to a single partition. Here is a sample of the sorted data:

```
"CHRIS TRAIN","1400 NEW ST","GC14036","GlobalCoUS"
"HUW WILLIAMS","579 DIGBETH AVENUE","GC14127","GlobalCoUS"
"JON SMITH","789 LEDBURY ROAD","GC13849","GlobalCoUS"
"LUC TEACHER","3 BIRMINGHAM ROAD","GC14346","GlobalCoUS"
"MARY GARDENER","127 BORDER ST","GC13933","GlobalCoUS"
"SARA PEARS","45 ALCESTER WAY","GC14263","GlobalCoUS"
```

Parallel sort

This example uses the same job and the same data as the last previous example, but this time you are going to run the Sort stage in parallel and create a number of partitions.

In the Sort stage you specify parallel execution in the Stage page Advanced tab. In the Input page Partitioning tab you specify a partitioning type of Hash, and specify the column SOURCE as the hash key. Because the partitioning takes place on the input link, the data is partitioned before the sort stage actually tries to sort it. You hash partition to ensure that customers from the same country end up in the same partition. The data is then sorted within those partitions.

The job is run on a four-node system, so you end up with a data set comprising four partitions.

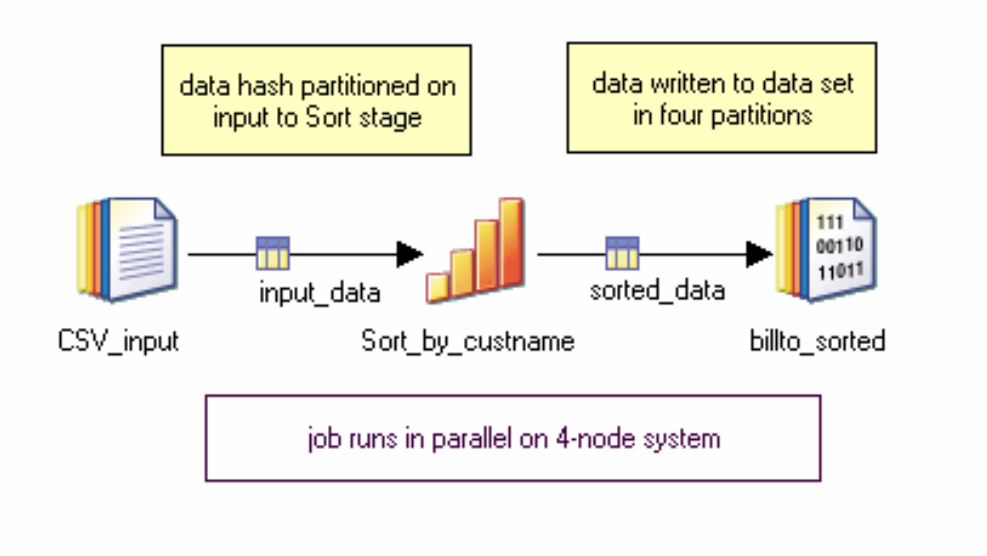


Figure 11. Sort stage

The following is a sample of the data in partition 2 after partitioning, but before sorting:

```
"JEAN DUPONT","576 RUE DE PARIS","GC20002","GlobalCoFR"
"MARIE TISON","14 AVENUE DE CALAIS","GC20012","GlobalCoFR"
"PIERRE FOURNIER","321 RUE VOLTAIRE","GC20021","GlobalCoFR"
"LOUIS LEROY","3 RUE DES TREUILS","GC20032","GlobalCoFR"
"NICOLE GIRARD","1234 QUAI DE LA TOURNELLE","GC20040","GlobalCoFR"
"DANIELLE BLANC","987 BOULEVARD AUXERRE","GC20049","GlobalCoFR"
```

And here is a sample of the data in partition 2 after it has been processed by the sort stage:

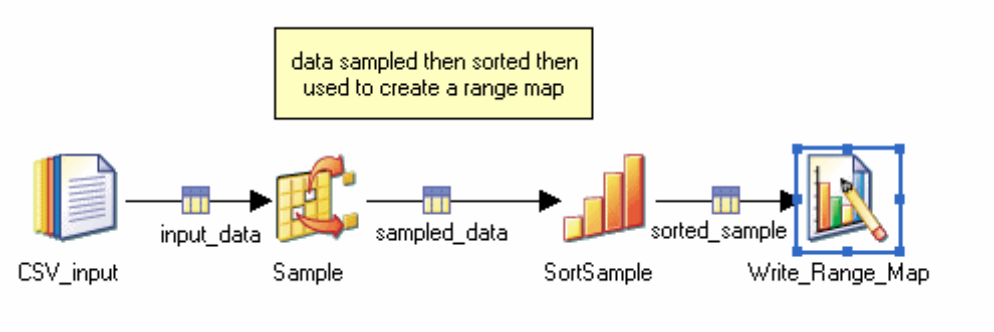
```
"DANIELLE BLANC","987 BOULEVARD AUXERRE","GC20049","GlobalCoFR"
"JEAN DUPONT","576 RUE DE PARIS","GC20002","GlobalCoFR"
"LOUIS LEROY","3 RUE DES TREUILS","GC20032","GlobalCoFR"
"MARIE TISON","14 AVENUE DE CALAIS","GC20012","GlobalCoFR"
"NICOLE GIRARD","1234 QUAI DE LA TOURNELLE","GC20040","GlobalCoFR"
"PIERRE FOURNIER","321 RUE VOLTAIRE","GC20021","GlobalCoFR"
```

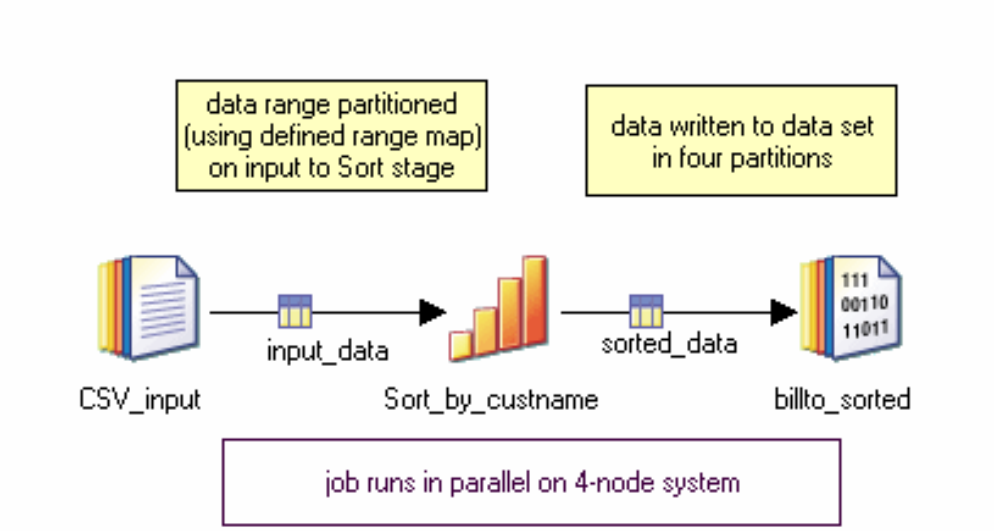
If you want to bring the data back together into a single partition, for example to write to another sequential file, you need to be mindful of how it is collected, or you will lose the benefit of the sort. If we use the sort/merge collection method, specifying the CUST_NAME column as the collection key, you will end up with a totally sorted data set.

Total sort

You can also perform a total sort on a parallel data set, such that the data is ordered within each partition and the partitions themselves are ordered. 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. The partitions also need to be approximately the same size so that no one node becomes a processing bottleneck.

In order to meet these two requirements, the input data is partitioned using the range partitioner. This guarantees that all records with the same key fields are in the same partition, and calculates the partition boundaries based on the key field to ensure fairly even distribution. In order to use the range partitioner you must first take a sample of your input data, sort it, then use it to build a range partition map as described in “Write Range Map stage” on page 540, You then specify this map when setting up the range partitioner in the Input page Partitioning tab of your Sort stage.





When you run the job it will produce a totally sorted data set across the four partitions.

Sort stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Sort stages in a job. This section specifies the minimum steps to take to get a Sort stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Sort stage:

- In the Stage page **Properties Tab**, under the Sorting Keys category:
  - specify the key that you are sorting on. Repeat the property to specify a composite key.
- In the Output page **Mapping Tab**, specify how the output columns are derived.

Sort stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Sort stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 35. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Sorting Keys/Key	Input Column	N/A	Y	Y	N/A
Sorting Keys/Sort Order	Ascending/ Descending	Ascending	Y	N	Key
Sorting Keys/Nulls position (only available for Sort Utility = DataStage)	First/Last	First	N	N	Key
Sorting Keys/Sort as EBCDIC	True/False	False	N	N	Key
Sorting Keys/Case Sensitive	True/False	True	N	N	Key
Sorting Keys/Sort Key Mode (only available for Sort Utility = DataStage)	Sort/Don't Sort (Previously Grouped)/Don't Sort (Previously Sorted)	Sort	Y	N	Key
Options/Sort Utility	DataStage/ UNIX	DataStage	Y	N	N/A
Options/Stable Sort	True/False	True for Sort Utility = DataStage, False otherwise	Y	N	N/A
Options/Allow Duplicates (not available for Sort Utility = UNIX)	True/False	True	Y	N	N/A
Options/Output Statistics	True/False	False	Y	N	N/A
Options/Create Cluster Key Change Column (only available for Sort Utility = DataStage)	True/False	False	N	N	N/A
Options/Create Key Change Column	True/False	False	N	N	N/A
Options/Restrict Memory Usage	number MB	20	N	N	N/A

Table 35. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Workspace	string	N/A	N	N	N/A

Sort stage: Sorting keys category: Key

Specifies the key column for sorting. This property can be repeated to specify multiple key columns. You can use the Column Selection dialog box to select several keys at once if required. Key has dependent properties depending on the Sort Utility chosen:

- **Sort Order**

  All sort types. Choose Ascending or Descending. The default is Ascending.
- **Nulls position**

  This property appears for sort type DataStage and is optional. By default columns containing null values appear first in the sorted data set. To override this default so that columns containing null values appear last in the sorted data set, select Last.
- **Sort as EBCDIC**

  To sort as in the EBCDIC character set, choose True.
- **Case Sensitive**

  All sort types. This property is optional. Use this to specify whether each group key is case sensitive or not, this is set to True by default, that is, the values "CASE" and "case" would not be judged equivalent.
- **Sort Key Mode**

  This property appears for sort type DataStage. It is set to Sort by default and this sorts on all the specified key columns.

  Set to Don't Sort (Previously Sorted) to specify that input records are already sorted by this column. The Sort stage will then sort on secondary key columns, 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 column(s) because you only need to sort your data on the secondary key column(s).

  Set to Don't Sort (Previously Grouped) to specify that input records are already grouped by this column, but not sorted. The operator will then sort on any secondary key columns. This option is useful when your records are already grouped by the primary key column(s), but not necessarily sorted, and you want to sort your data only on the secondary key column(s) within each group

Sort stage: Options category: Sort utility

The type of sort the stage will carry out. Choose from:

- **DataStage**. The default. This uses the built-in InfoSphere DataStage sorter, you do not require any additional software to use this option.
- **UNIX**. This specifies that the UNIX *sort* command is used to perform the sort.

Stable sort

Applies to a Sort Utility type of DataStage, the default is True. It is set to True to guarantee that this sort operation will not rearrange records that are already in a properly sorted data set. If set to False no prior ordering of records is guaranteed to be preserved by the sorting operation.

Allow duplicates

Set to True by default. If False, specifies that, if multiple records have identical sorting key values, only one record is retained. If Stable Sort is True, then the first record is retained. This property is not available for the UNIX sort type.

Output statistics

Set False by default. If True it causes the sort operation to output statistics. This property is not available for the UNIX sort type.

Create cluster key change column

This property appears for sort type DataStage and is optional. It is set False by default. If set True it tells the Sort stage to create the column **clusterKeyChange** in each output record. The **clusterKeyChange** column is set to 1 for the first record in each group where groups are defined by using a Sort Key Mode of Don't Sort (Previously Sorted) or Don't Sort (Previously Grouped). Subsequent records in the group have the **clusterKeyChange** column set to 0.

Create key change column

This property appears for sort type DataStage and is optional. It is set False by default. If set True it tells the Sort stage to create the column **KeyChange** in each output record. The **KeyChange** column is set to 1 for the first record in each group where the value of the sort key changes. Subsequent records in the group have the **KeyChange** column set to 0.

Restrict memory usage

This is set to 20 by default. It causes the Sort stage to restrict itself to the specified number of megabytes of virtual memory on a processing node.

The number of megabytes specified should be smaller than the amount of physical memory on a processing node. For Windows systems, the value for **Restrict Memory Usage** should not exceed 500.

Workspace

This property appears for sort type UNIX only. Optionally specifies the workspace used by the stage.

Sort stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the **Advanced** tab. In Sequential mode the entire data set is processed by the conductor node.
- **Preserve partitioning**. This is **Set** by default. You can explicitly select **Set** or **Clear**. Select **Set** to request the next stage in the job should attempt to maintain the partitioning.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse

button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Sort stage: NLS Locale tab

This appears if you have NLS enabled on your system. If you are using the DataStage sort type, it lets you view the current default collate convention, and select a different one for this stage if required (for UNIX sorts, it is blank). You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Sort stage uses this when it is determining the order of the sorted fields. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Sort stage: Input page

The input page allows you to specify details about the data coming in to be sorted. The Sort stage can have only one input link.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Sort stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Sort stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the sort is performed.

By default the stage uses the auto partitioning method. If the Preserve Partitioning option has been set on the previous stage in the job the stage will warn you when the job runs if it cannot preserve the partitioning of the incoming data.

If the Sort Set stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Sort stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Sort stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Sort stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Sort stage.
- **Entire**. Each file written to receives the entire data set.

- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Sort stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the Sort is performed. This is a standard feature of the stage editors, if you make use of it you will be running a simple sort before the main Sort operation that the stage provides. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Sort stage: Output page

The Output page allows you to specify details about data output from the Sort stage. The Sort stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Sort stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Sort stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Sort stage: Mapping tab

For Sort stages the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them.

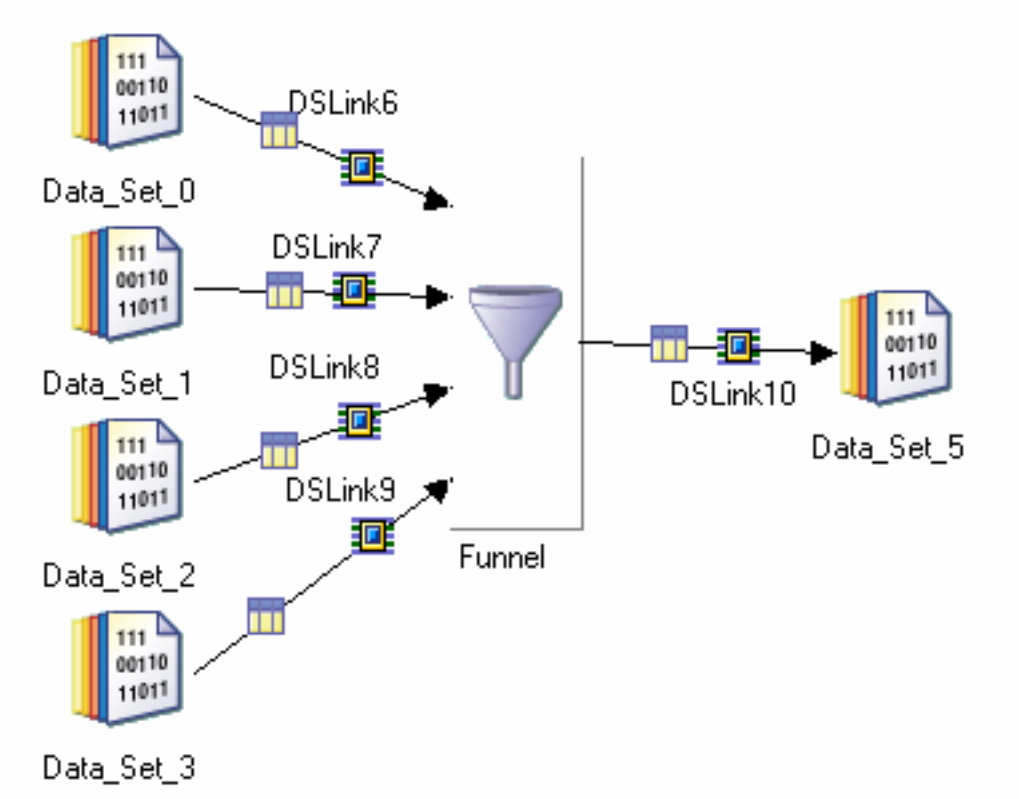
The left pane shows the columns of the sorted data. These are read only and cannot be modified on this tab. This shows the meta data from the input link.

The right pane shows the output columns for the output link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility.

In the above example the left pane represents the incoming data after the sort has been performed. The right pane represents the data being output by the stage after the sort operation. In this example the data has been mapped straight across.

Funnel Stage

The Funnel stage is a processing stage. It copies 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. The stage can have any number of input links and a single output link.



The Funnel stage can operate in one of three modes:

- **Continuous Funnel** combines the records of the input data in no guaranteed order. It takes one record from each input link in turn. If data is not available on an input link, the stage skips to the next link rather than waiting.
- **Sort Funnel** combines the input records in the order defined by the value(s) of one or more key columns and the order of the output records is determined by these sorting keys.
- **Sequence** 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.

For all methods the meta data of all input data sets must be identical.

The sort funnel method has some particular requirements about its input data. All input data sets must be sorted by the same key columns as to be used by the Funnel operation.

Typically all input data sets for a sort funnel operation are hash-partitioned before they're sorted (choosing the auto partitioning method will ensure that this is done). Hash partitioning guarantees that all records with the same key column values are located in the same partition and so are processed on the same node. If sorting and partitioning are carried out on separate stages before the Funnel stage, this partitioning must be preserved.

The **sortfunnel** operation allows you to set one *primary* key and multiple *secondary* keys. The Funnel stage first examines the primary key in each input record. For multiple records with the same primary key value, it then examines secondary keys to determine the order of records it will output.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data sets being joined.
- **Output Page**. This is where you specify details about the joined data being output from the stage.

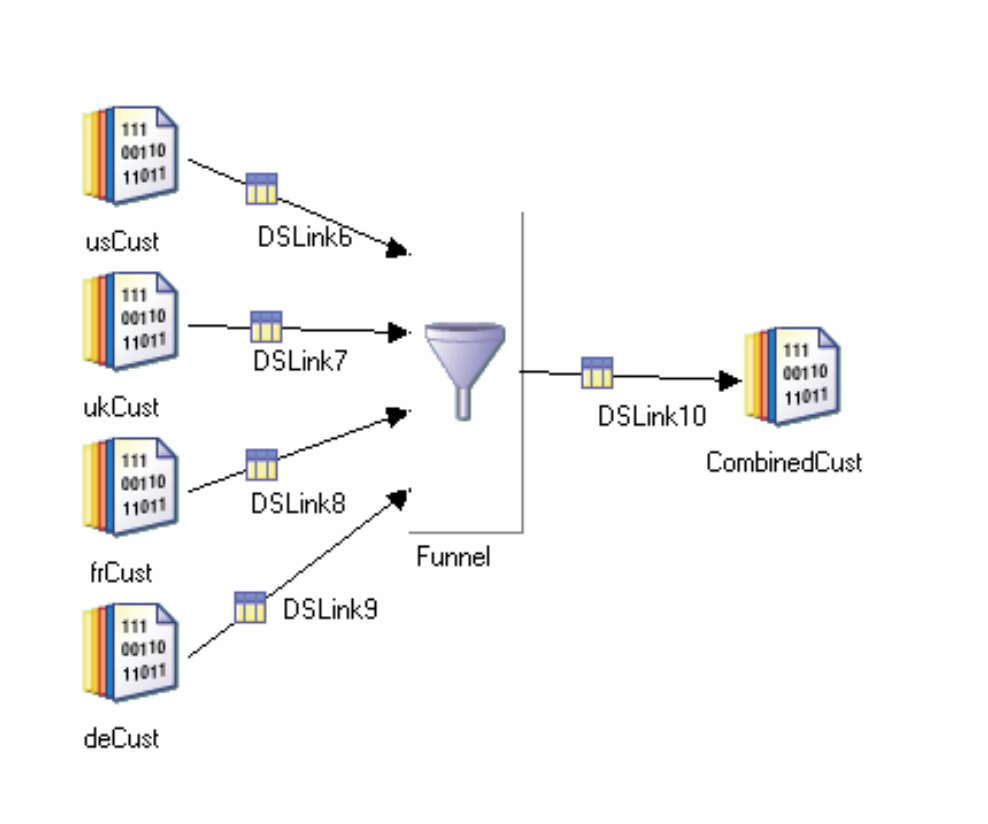
Examples

Continuous funnel example

the example data comprises four separate data sets. Each data set contains a list of GlobalCo customers from a particular country. Here is a sample of the data for the US customers:

```
"JON SMITH","789 LEDBURY ROAD","GC13849","GlobalCoUS"
"MARY GARDENER","127 BORDER ST","GC13933","GlobalCoUS"
"CHRIS TRAIN","1400 NEW ST","GC14036","GlobalCoUS"
"HUW WILLIAMS","579 DIGBETH AVENUE","GC14127","GlobalCoUS"
"SARA PEARS","45 ALCESTER WAY","GC14263","GlobalCoUS"
"LUC TEACHER","3 BIRMINGHAM ROAD","GC14346","GlobalCoUS"
```

The Funnel stage, when set to continuous funnel, will combine these into a single data set. The job to perform the funnel is as follows:



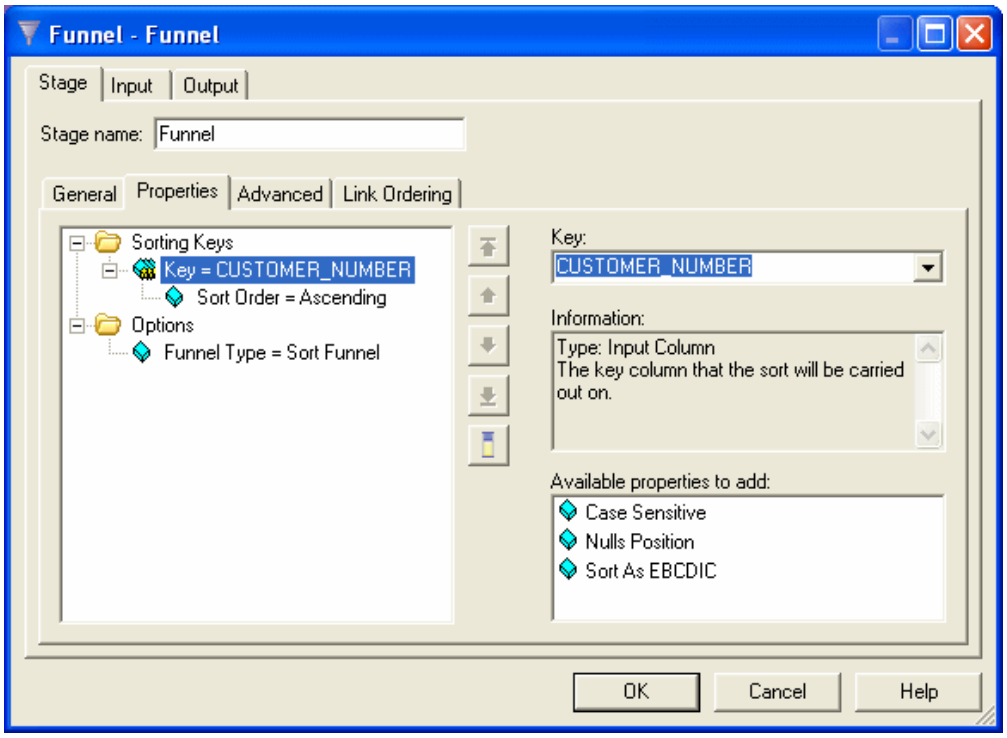
The continuous funnel method is selected on the Stage page **Properties** tab of the Funnel stage:

The continuous funnel method does not attempt to impose any order on the data it is processing. It simply writes rows as they become available on the input links. In the example the stage has written a row from each input link in turn. A sample of the final, funneled, data is as follows:

```
"JON SMITH","789 LEDBURY ROAD","GC13849","GlobalCoUS"
"ELIZABETH PARKER","35 YORK ROAD","GC21745","GlobalCoUK"
"JEAN DUPONT","576 RUE DE PARIS","GC20002","GlobalCoFR"
"ADELE BECKER","AM HUNGERSBERG 123","GC22145","GlobalCoDE"
"MARY GARDENER","127 BORDER ST","GC13933","GlobalCoUS"
"WINSTON HILL","87 MULBERRY CLOSE","GC21874","GlobalCoUK"
"MARIE TISON","14 AVENUE DE CALAIS","GC20012","GlobalCoFR"
"KLAUS SCHMIDT","WOLBURGSWEG 7645","GC22478","GlobalCoDE"
```

Sort Funnel Example

In this example we are going to use the funnel stage to sort the GlobalCo data by customer number as it combines the data into a single data set. The data and the basic job are the same as for the Continuous Funnel example, but now we set the Funnel stage properties as follows:



The following is a sample of the output data set:

```
"JON SMITH","789 LEDBURY ROAD","GC13849","GlobalCoUS"
"MARY GARDENER","127 BORDER ST","GC13933","GlobalCoUS"
"CHRIS TRAIN","1400 NEW ST","GC14036","GlobalCoUS"
"HUW WILLIAMS","579 DIGBETH AVENUE","GC14127","GlobalCoUS"
"SARA PEARS","45 ALCESTER WAY","GC14263","GlobalCoUS"
"LUC TEACHER","3 BIRMINGHAM ROAD","GC14346","GlobalCoUS"
"JEAN DUPONT","576 RUE DE PARIS","GC20002","GlobalCoFR"
"MARIE TISON","14 AVENUE DE CALAIS","GC20012","GlobalCoFR"
"PIERRE FOURNIER","321 RUE VOLTAIRE","GC20021","GlobalCoFR"
"LOUIS LEROY","3 RUE DES TREUILS","GC20032","GlobalCoFR"
"NICOLE GIRARD","1234 QUAI DE LA TOURNELLE","GC20040","GlobalCoFR"
"DANIELLE BLANC","987 BOULEVARD AUXERRE","GC20049","GlobalCoFR"
```

Note: If you are running your sort funnel stage in parallel, you should be aware of the various considerations about sorting data and partitions. These are described in "Sort Stage."

Sequence funnel example

In this example you funnel the GlobalCo data on input one data set at a time. You get a data set that contains all the US customers, then all the UK ones, then all the French ones and so on. Again the basic job and the source data are the same as for the continuous funnel example. The Funnel type property is set to Sequence on the Properties tab.

When using the sequence method, you need to specify the order in which the Funnel stage processes its input links, as this affects the order of the sequencing. This is done on the Stage page Link Ordering tab.

If you run the sequence funnel stage in parallel, you need to be mindful of the effects of data partitioning. If, for example, you ran the example job on a four-node system, you would get four partitions each containing a section of US data, a section of UKdata, a section of FR data and so on.

Funnel stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Funnel stages in a job. This section specifies the minimum steps to take to get a Funnel stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Funnel stage:

- In the Stage page **Properties Tab**, specify the Funnel Type. Continuous Funnel is the default, but you can also choose Sequence or Sort Funnel.

  If you choose to use the Sort Funnel method, you also need to specify the key on which data will be sorted. You can repeat the key property to specify a composite key.
- If you are using the Sequence method, in the Stage page **Link Ordering Tab** specify the order in which your data sets will be combined.
- In the Output page **Mapping Tab**, specify how the output columns are derived.

Funnel stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The Link Ordering tab allows you to specify which order the input links are processed in. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Funnel stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 36. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Funnel Type	Continuous Funnel/ Sequence/ Sort funnel	Continuous Funnel	Y	N	N/A
Sorting Keys/Key	Input Column	N/A	Y (if Funnel Type = Sort Funnel)	Y	N/A
Sorting Keys/Sort Order	Ascending/ Descending	Ascending	Y (if Funnel Type = Sort Funnel)	N	Key
Sorting Keys/Nulls position	First/Last	First	Y (if Funnel Type = Sort Funnel)	N	Key
Sorting Keys/Case Sensitive	True/False	True	N	N	Key
Sorting Keys/Sort as EBCDIC	True/False	False	N	N	Key

Funnel stage: Options category:
Funnel type

Specifies the type of Funnel operation. Choose from:

- Continuous Funnel
- Sequence
- Sort Funnel

The default is Continuous Funnel.

Funnel stage: Sorting keys category:
Key

This property is only required for Sort Funnel operations. Specify the key column that the sort will be carried out on. The first column you specify is the primary key, you can add multiple secondary keys by repeating the key property. You can use the Column Selection dialog box to select several keys at once if required.

Key has the following dependent properties:

- **Sort Order**

  Choose Ascending or Descending. The default is Ascending.
- **Nulls position**

  By default columns containing null values appear first in the funneled data set. To override this default so that columns containing null values appear last in the funneled data set, select Last.
- **Sort as EBCDIC**

  To sort as in the EBCDIC character set, choose True.
- **Case Sensitive**

  Use this to specify whether each key is case sensitive or not, this is set to True by default, that is, the values "CASE" and "case" would not be judged equivalent.

Funnel stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the **Advanced** tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts the setting which results from ORing the settings of the input stages, that is, if any of the input stages uses **Set** then this stage will use **Set**. You can explicitly select **Set** or **Clear**. Select **Set** to request that the next stage in the job attempts to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse

button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Funnel stage: Link Ordering tab

This tab allows you to specify the order in which links input to the Funnel stage are processed. This is only relevant if you have chosen the Sequence Funnel Type.

By default the input links will be processed in the order they were added. To rearrange them, choose an input link and click the up arrow button or the down arrow button.

Funnel stage: NLS Locale tab

This appears if you have NLS enabled on your system. If you are using the Sort Funnel funnel type, it lets you view the current default collate convention, and select a different one for this stage if required (for other funnel types, it is blank). You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Funnel stage uses this when it is determining the sort order for sort funnel. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Funnel stage: Input page

The Input page allows you to specify details about the incoming data sets. Choose an input link from the **Input name** drop down list to specify which link you want to work on.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being funneled. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Funnel stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Funnel stage: Partitioning on input links

The Partitioning tab allows you to specify details about how the data on each of the incoming links is partitioned or collected before it is funneled. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Funnel stage is operating in sequential mode, it will first collect the data before writing it to the file using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Funnel stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Funnel stage is set to execute in parallel, then you can set a partitioning method by selecting from the Partition type drop-down list. This will override any current partitioning.

If you are using the Sort Funnel method, and haven't partitioned the data in a previous stage, you should key partition it by choosing the Hash or modulus partition method on this tab.

If the Funnel stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Funnel stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Funnel stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being funneled. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection.

If you are using the Sort Funnel method, and haven't sorted the data in a previous stage, you should sort it here using the same keys that the data is hash partitioned on and funneled on. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Funnel stage: Output page

The Output page allows you to specify details about data output from the Funnel stage. The Funnel stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Funnel stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Funnel stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Funnel stage: Mapping tab

For Funnel stages the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them or how they are generated.

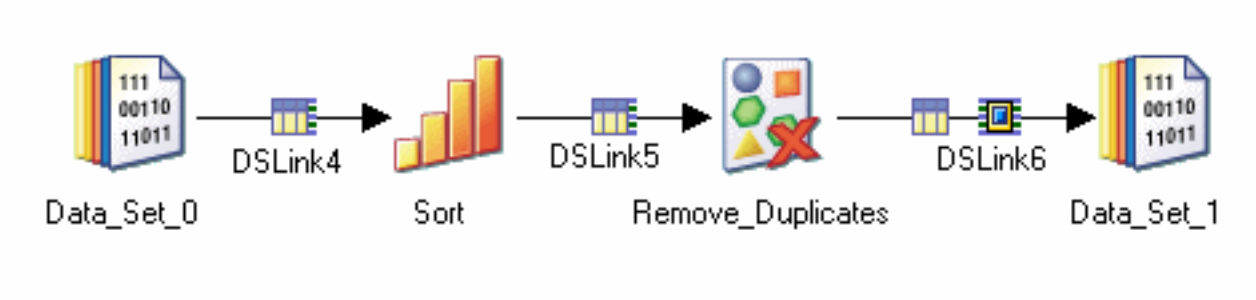
The left pane shows the input columns. These are read only and cannot be modified on this tab. It is a requirement of the Funnel stage that all input links have identical meta data, so only one set of column definitions is shown.

The right pane shows the output columns for each link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility.

Remove Duplicates Stage

The Remove Duplicates stage is a processing stage. It can have a single input link and a single output link.

The Remove Duplicates stage takes a single sorted data set as input, removes all duplicate rows, 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 rows are considered duplicates if they are adjacent in the input data set and have identical values for the key column(s). A key column is any column you designate to be used in determining whether two rows are identical.

The data set input to the Remove Duplicates stage must be sorted so that all records with identical key values are adjacent. You can either achieve this using the in-stage sort facilities available on the Input page Partitioning tab, or have an explicit Sort stage feeding the Remove Duplicates stage.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data set that is having its duplicates removed.
- **Output Page**. This is where you specify details about the processed data that is being output from the stage.

Example of the Remove Duplicates stage

In the example the data is a list of GlobalCo's customers. The data contains some duplicate entries, and you want to remove these.

The first step is to sort the data so that the duplicates are actually next to each other. As with all sorting operations, there are implications around data partitions if you run the job in parallel (see "Copy Stage," for a discussion of these). You should hash partition the data using the sort keys as hash keys in order to guarantee that duplicate rows are in the same partition. In the example you sort on the CUSTOMER_NUMBER columns and the sample of the sorted data shows up some duplicates:

```
"GC13849","JON SMITH","789 LEDBURY ROAD","2/17/2007"
"GC13933","MARY GARDENER","127 BORDER ST","8/28/2009"
"GC13933","MARY GARDENER","127 BORDER ST","8/28/2009"
"GC14036","CHRIS TRAIN","1400 NEW ST","9/7/1998"
"GC14127","HUW WILLIAMS","579 DIGBETH AVENUE","6/29/2011"
"GC14263","SARA PEARS","45 ALCESTER WAY","4/12/2008"
"GC14263","SARA PEARS","45 ALCESTER WAY","4/12/2008"
"GC14346","LUC TEACHER","3 BIRMINGHAM ROAD","11/7/2010"
```

Next, you set up the Remove Duplicates stage to remove rows that share the same values in the CUSTOMER_NUMBER column. The stage will retain the first of the duplicate records:

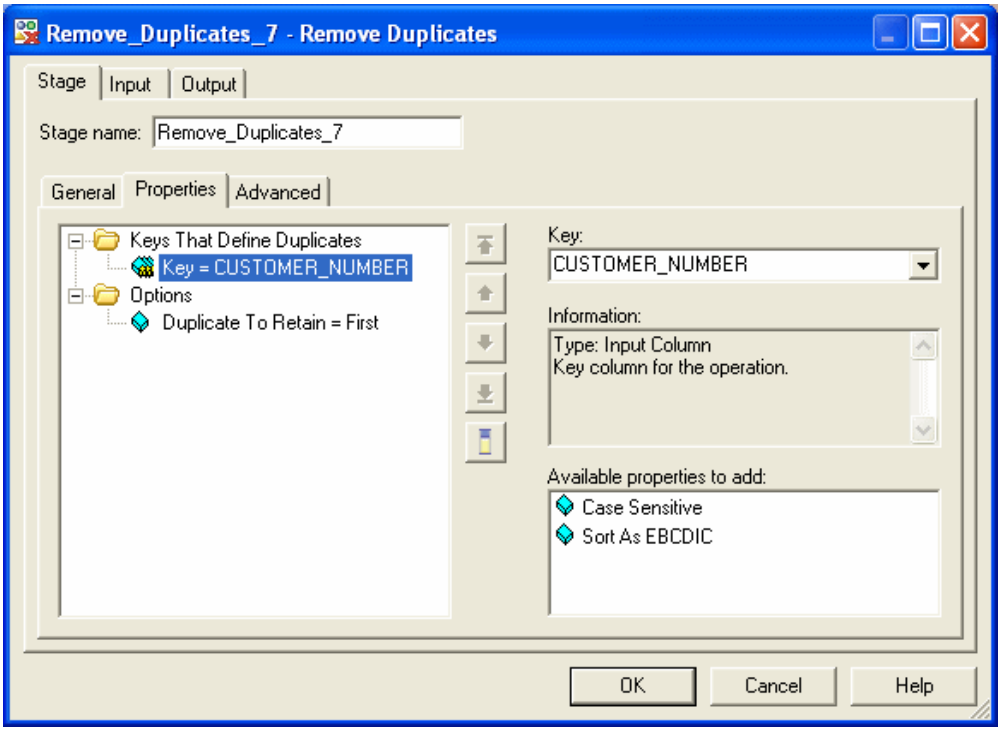


Figure 12. Property settings

Here is a sample of the data after the job has been run and the duplicates removed:

```
"GC13849","JON SMITH","789 LEDBURY ROAD","2/17/2007"
"GC13933","MARY GARDENER","127 BORDER ST","8/28/2009"
"GC14036","CHRIS TRAIN","1400 NEW ST","9/7/1998"
"GC14127","HUW WILLIAMS","579 DIGBETH AVENUE","6/29/2011"
"GC14263","SARA PEARS","45 ALCESTER WAY","4/12/2008"
"GC14346","LUC TEACHER","3 BIRMINGHAM ROAD","11/7/2010"
```

Remove Duplicates stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Remove Duplicates stages in a job. This section specifies the minimum steps to take to get a Remove Duplicates stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Remove Duplicates stage:

- In the Stage page **Properties Tab** select the key column. Identical values in this column will be taken to denote duplicate rows, which the stage will remove. Repeat the property to specify a composite key.
- In the Output Page **Mapping Tab**, specify how output columns are derived.

Remove Duplicates stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Remove Duplicates stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 37. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Keys that Define Duplicates/Key	Input Column	N/A	Y	Y	N/A
Keys that Define Duplicates/Sort as EBCDIC	True/False	False	N	N	Key
Keys that Define Duplicates/Case Sensitive	True/False	True	N	N	Key
Options/ Duplicate to retain	First/Last	First	Y	N	N/A

Remove Duplicates stage: Keys that define duplicates category: Key

Specifies the key column for the operation. This property can be repeated to specify multiple key columns. You can use the Column Selection dialog box to select several keys at once if required. Key has dependent properties as follows:

- **Sort as EBCDIC**

  To sort as in the EBCDIC character set, choose True.
- **Case Sensitive**

  Use this to specify whether each key is case sensitive or not, this is set to True by default, that is, the values "CASE" and "case" would not be judged equivalent.

Remove Duplicates stage: Options category: Duplicate to retain

Specifies which of the duplicate columns encountered to retain. Choose between First and Last. It is set to First by default.

Remove Duplicates stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the **Advanced** tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.

- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Remove Duplicates stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Remove Duplicates stage uses this when it is determining the sort order for the key column(s). Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Remove Duplicates stage: Input page

The Input page allows you to specify details about the data coming in to be sorted. Choose an input link from the **Input name** drop down list to specify which link you want to work on.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Remove Duplicates stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Remove Duplicates stage: Partitioning on input links

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the operation is performed.

By default the stage uses the auto partitioning method.

If the Remove Duplicates stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Remove Duplicates stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Remove Duplicates stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Remove Duplicates stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Remove Duplicates stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Remove Duplicates stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the remove duplicates operation is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Remove Duplicates stage: Output page

The Output page allows you to specify details about data output from the Remove Duplicates stage. The stage only has one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Remove Duplicates stage and the output columns. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Remove Duplicates stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Remove Duplicates stage: Mapping tab

For Remove Duplicates stages the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them.

The left pane shows the columns of the input data. These are read only and cannot be modified on this tab. This shows the meta data from the incoming link.

The right pane shows the output columns for the master output link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility.

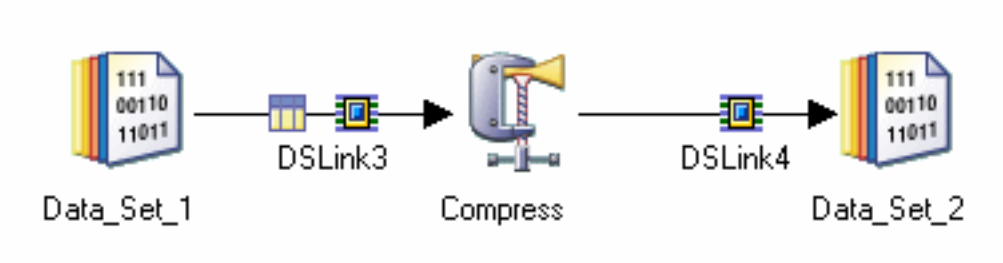
Compress stage

The Compress stage is a processing stage. It can have a single input link and a single output link.

The Compress stage uses the UNIX *compress* or *GZIP* utility to compress a data set. It converts a data set from a sequence of records into a stream of raw binary data. The complement to the Compress stage is the Expand stage, which is described in “Expand Stage” on page 301.

A compressed data set is similar to an ordinary data set and can be stored in a persistent form by a Data Set stage. However, a compressed data set cannot be processed by many stages until it is expanded, that is, until its rows are returned to their normal format. Stages that do not perform column-based processing or reorder the rows can operate on compressed data sets. For example, you can use the Copy stage to create a copy of the compressed data set.

Because compressing a data set removes its normal record boundaries, the compressed data set must not be repartitioned before it is expanded.



DataStage puts the existing data set schema as a subrecord to a generic compressed schema. For example, given a data set with a schema of:

```
a:int32;
b:string[50];
```

The schema for the compressed data set would be:

```
record
  ( t: tagged {preservePartitioning=no}
    ( encoded: subrec
        ( bufferNumber: dfloat;
```

```
    bufferLength: int32;
    bufferData: raw[32000];
   );
schema: subrec
  ( a: int32;
    b: string[50];
   );
```

Therefore, when you are looking to reuse a file that has been compressed, ensure that you use the 'compressed schema' to read the file rather than the schema that had gone into the compression.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data set being compressed.
- **Output Page**. This is where you specify details about the compressed data being output from the stage.

Compress stage: fast path

This section specifies the minimum steps to take to get a Compress stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Compress stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

1. In the Stage page **Properties Tab** choose the compress command to use. Compress is the default but you can also choose gzip.
2. Ensure column meta data is defined for both the input and output link.

Compress stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Compress stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. The stage only has a single property which determines whether the stage uses *compress* or *GZIP*.

Table 38. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Command	compress/gzip	compress	Y	N	N/A

Compress stage: Options category: Command

Specifies whether the stage will use *compress* (the default) or *GZIP*.

Compress stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Set** by default. You can explicitly select **Set** or **Clear**. Select **Set** to request the next stage should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Compress stage: Input page

The Input page allows you to specify details about the data set being compressed. There is only one input link.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Compress stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Compress stage: Partitioning on input links

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the compress is performed.

By default the stage uses the auto partitioning method.

If the Compress stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Compress stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Compress stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Compress stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Compress stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Compress stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the compression is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Compress stage: Output page

The Output page allows you to specify details about data output from the Compress stage. The stage only has one output link.

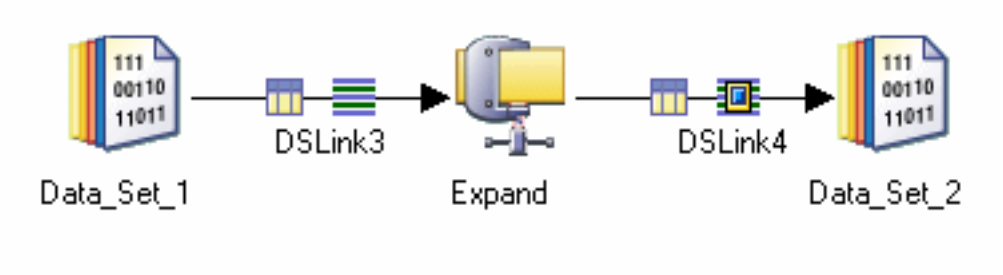
The General tab allows you to specify an optional description of the output link. The **Columns** tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

See "Stage Editors," for a general description of the tabs.

Expand Stage

The Expand stage is a processing stage. It can have a single input link and a single output link.

The Expand stage uses the UNIX *uncompress* or *GZIP* utility to expand a data set. It converts a previously compressed data set back into a sequence of records from a stream of raw binary data. The complement to the Expand stage is the Compress stage which is described in “Compress stage” on page 297.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data set being expanded.
- **Output Page**. This is where you specify details about the expanded data being output from the stage.

Expand stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Expand stages in a job. This section specifies the minimum steps to take to get an Expand stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use an Expand stage:

- In the Stage page **Properties Tab** choose the uncompress command to use. This is *uncompress* by default but you can also choose *gzip*.
- Ensure column meta data is defined for both the input and output link.

Expand stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Expand stage: Properties tab

The **Properties** tab allows you to specify properties which determine what the stage actually does. The stage only has a single property which determines whether the stage uses *uncompress* or *GZIP*.

Table 39. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Command	uncompress/gzip	uncompress	Y	N	N/A

Expand stage: Options category: Command

Specifies whether the stage will use *uncompress* (the default) or *GZIP*.

Expand stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. The stage has a mandatory partitioning method of **Same**, this overrides the preserve partitioning flag and so the partitioning of the incoming data is always preserved.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Expand stage: Input page

The Input page allows you to specify details about the data set being expanded. There is only one input link.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data. The **Advanced** tab allows you to change the default buffering settings for the input link.

Details about Expand stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Expand stage: Partitioning on input Links

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the expansion is performed.

By default the stage uses the **Same** partitioning method and this cannot be altered. This preserves the partitioning already in place.

If the Expand stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Expand stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab normally also allows you to specify that data arriving on the input link should be sorted before the expansion is performed. This facility is not available on the expand stage.

Expand stage: Output page

The Output page allows you to specify details about data output from the Expand stage. The stage only has one output link.

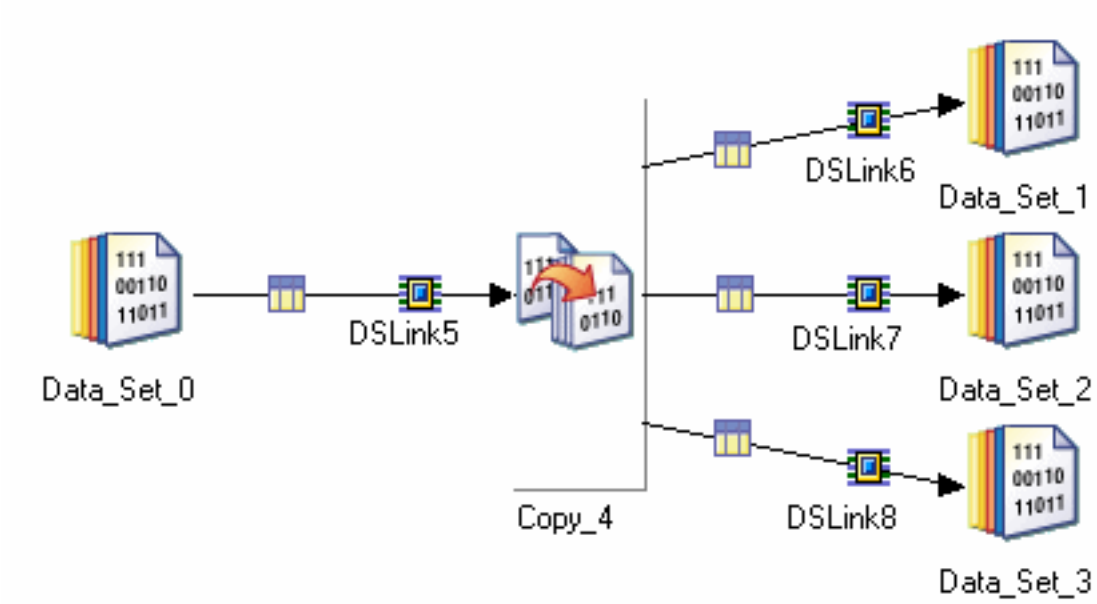
The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

See "Stage Editors," for a general description of the tabs.

Copy stage

The Copy stage is a processing stage. It can have a single input link and any number of output links.

The Copy stage copies a single input data set to a number of output data sets. Each record of the input data set is copied to every output data set. Records can be copied without modification or you can drop or change the order of columns (to copy with more modification - for example changing column data types - use the Modify stage as described in “Modify stage” on page 313). Copy lets you make a backup copy of a data set on disk while performing an operation on another copy, for example.



Where you are using a Copy stage with a single input and a single output, you should ensure that you set the Force property in the stage editor TRUE. This prevents InfoSphere DataStage from deciding that the Copy operation is superfluous and optimizing it out of the job.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the input link carrying the data to be copied.
- **Output Page**. This is where you specify details about the copied data being output from the stage.

Copy stage example

In this example you are going to copy data from a table containing billing information for GlobalCo's customers. You are going to copy it to three separate data sets, and in each case you are only copying a subset of the columns. The Copy stage will drop the unwanted columns as it copies the data set.

The column names for the input data set are as follows:

- BILL_TO_NUM
- CUST_NAME
- ADDR_1
- ADDR_2
- CITY
- REGION_CODE
- ZIP
- ATTENT
- COUNTRY_CODE
- TEL_NUM
- FIRST_SALES_DATE
- LAST_SALES_DATE
- REVIEW_MONTH

- SETUP_DATE
- STATUS_CODE
- REMIT_TO_CODE
- CUST_TYPE_CODE
- CUST_VEND
- MOD_DATE
- MOD_USRNM
- CURRENCY_CODE
- CURRENCY_MOD_DATE
- MAIL_INVC_FLAG
- PYMNT_CODE
- YTD_SALES_AMT
- CNTRY_NAME
- CAR_RTE
- TPF_INVC_FLAG,
- INVC_CPY_CNT
- INVC_PRT_FLAG
- FAX_PHONE,
- FAX_FLAG
- ANALYST_CODE
- ERS_FLAG

Here is the job that will perform the copying:

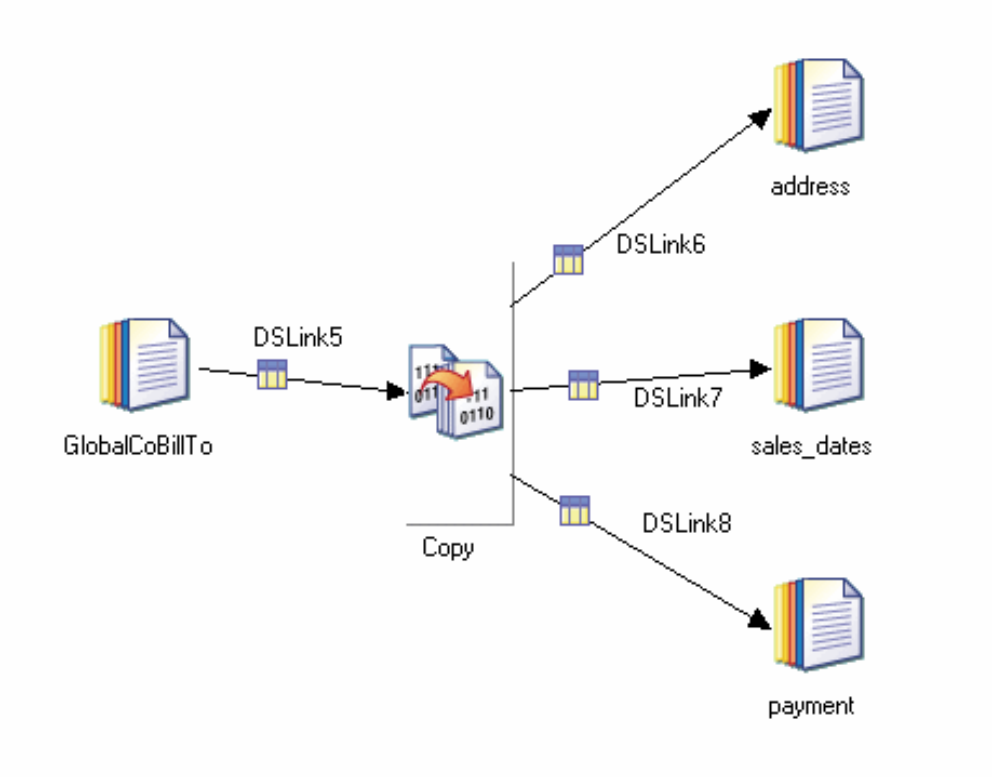


Figure 13. Example job

The Copy stage properties are fairly simple. The only property is Force, and you do not need to set it in this instance as you are copying to multiple data sets (and InfoSphere DataStage will not attempt to optimize it out of the job). You need to concentrate on telling InfoSphere DataStage which columns to drop on each output link. The easiest way to do this is using the Output page Mapping tab. When you open this for a link the left pane shows the input columns, simply drag the columns you want to preserve across to the right pane. You repeat this for each link as follows:

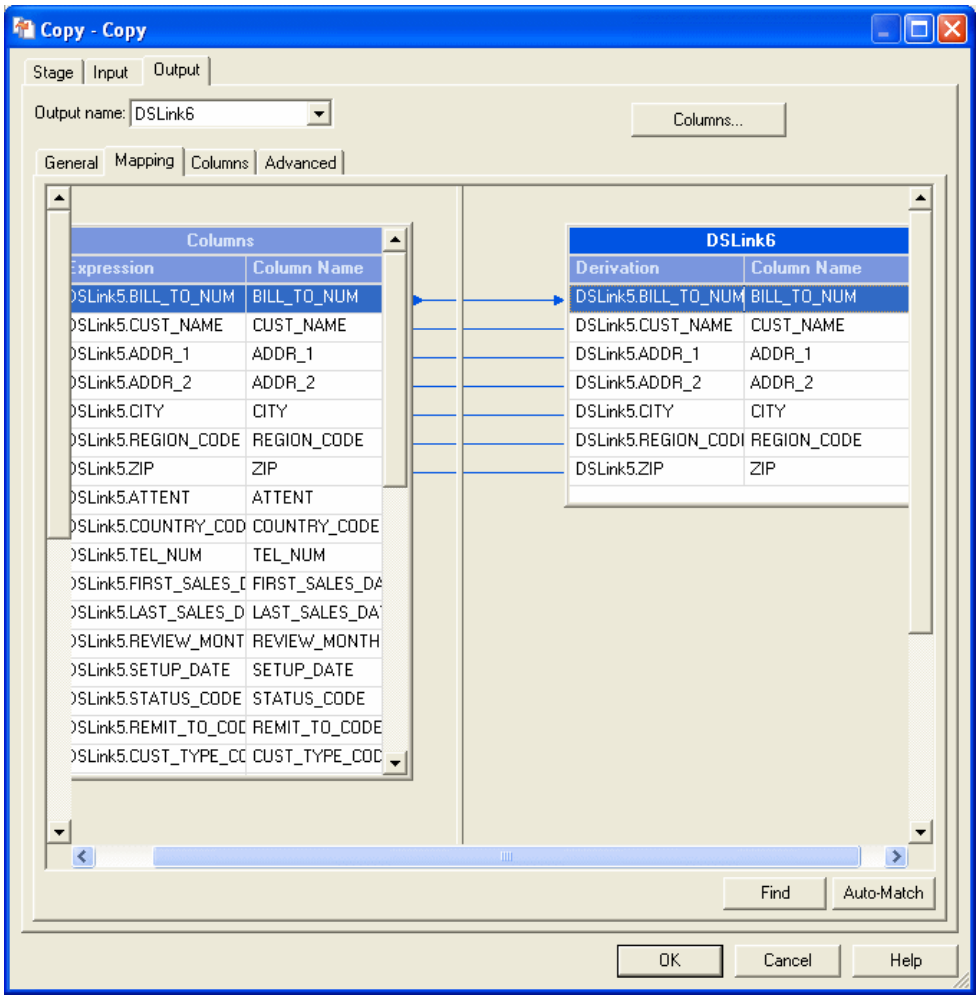


Figure 14. Mapping tab: first output link

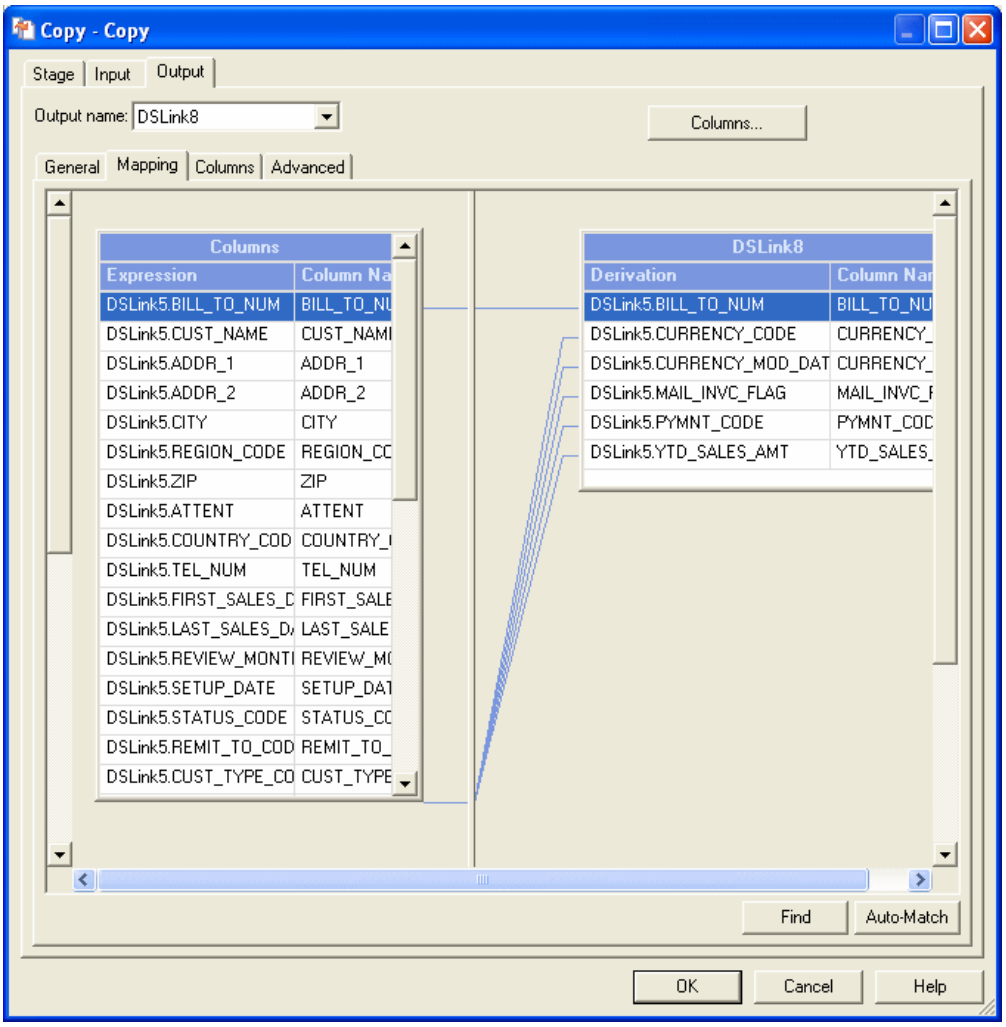


Figure 15. Mapping tab: second output link

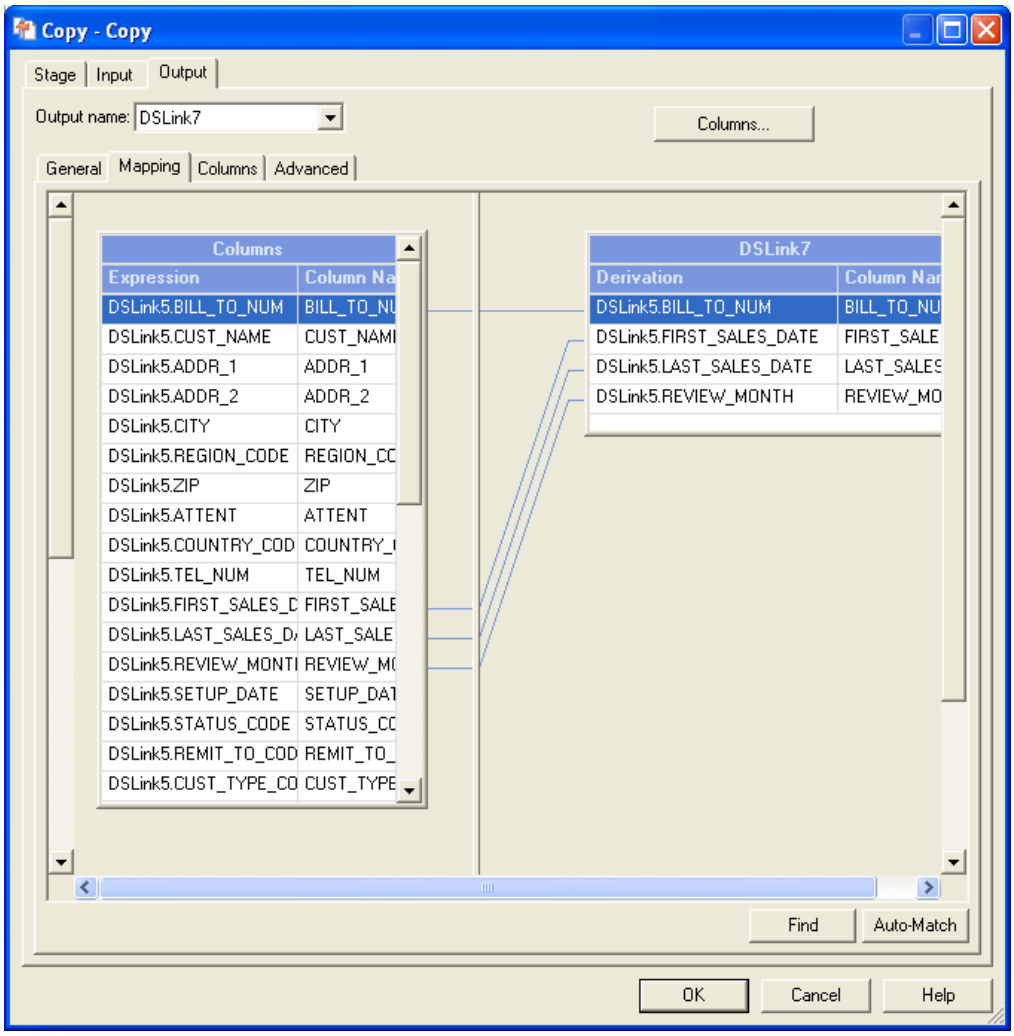


Figure 16. Mapping tab: third output link

When the job is run, three copies of the original data set are produced, each containing a subset of the original columns, but all of the rows. Here is some sample data from each of the data set on DSLink6, which gives name and address information:

```
"GC13849","JON SMITH","789 LEDBURY ROAD"," ","TAMPA","FL","12345"
"GC13933","MARY GARDENER","127 BORDER ST"," ","NORTHPORT","AL","23456"
"GC14036","CHRIS TRAIN","1400 NEW ST"," ","BRENHAM","TX","34567"
"GC14127","HUW WILLIAMS","579 DIGBETH AVENUE"," ","AURORA","CO","45678"
"GC14263","SARA PEARS","45 ALCESTER WAY"," ","SHERWOOD","AR","56789"
"GC14346","LUC TEACHER","3 BIRMINGHAM ROAD"," ","CHICAGO","IL","67890"
```

Copy stage: fast path

This section specifies the minimum steps to take to get a Copy stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Copy stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

1. Ensure that meta data has been defined for input link and output links.
2. In the **Output Page Mapping Tab**, specify how the input columns of the data set being copied map onto the columns of the various output links.

Copy stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Copy stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. The Copy stage only has one property.

Table 40. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Force	True/False	False	N	N	N/A

Copy stage: Options category: Force

Set True to specify that InfoSphere DataStage should not try to optimize the job by removing a Copy operation where there is one input and one output. Set False by default.

Copy stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts the setting of the previous stage.You can explicitly select **Set** or **Clear**. Select **Set** to request the stage should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Copy stage: Input page

The Input page allows you to specify details about the data set being copied. There is only one input link.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Copy stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Copy stage: Partitioning on input links

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the copy is performed.

By default the stage uses the auto partitioning method.

If the Copy stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Copy stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Copy stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Copy stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Copy stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Copy stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.

- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the remove duplicates operation is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Copy stage: Output page

The Output page allows you to specify details about data output from the Copy stage. The stage can have any number of output links, choose the one you want to work on from the **Output name** drop down list.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Copy stage and the output columns. The Advanced tab allows you to change the default buffering settings for the output links.

Details about Copy stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Copy stage: Mapping tab

For Copy stages the Mapping tab allows you to specify how the output columns are derived, that is, what copied columns map onto them.

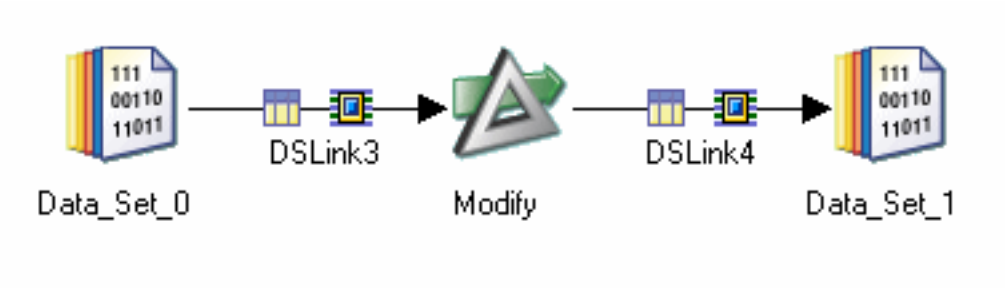
The left pane shows the copied columns. These are read only and cannot be modified on this tab.

The right pane shows the output columns for the output link. This has a **Derivations** field where you can specify how the column is derived.You can fill it in by dragging copied columns over, or by using the Auto-match facility.

Modify stage

The Modify stage is a processing stage. It can have a single input link and a single output link.

The Modify stage alters the record schema of its input data set. The modified data set is then output. You can drop or keep columns from the schema, or change the type of a column.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the input link.
- **Output Page**. This is where you specify details about the modified data being output from the stage.

Although the Modify stage is classified as a processing stage, it does not behave in the same way as other processing stages. In the Monitor window of the Director client, the stage does not show processing rows, and the stage does not show an outgoing row count in the Designer client.

Examples

Dropping and keeping columns

The following example takes a data set comprising the following columns:

Column name	SQL Type	Length	Scale
CUSTID	Decimal	6	
NAME	Char	45	
ADDRESS	Char	40	
CITY	Char	30	
STATE	Char	2	
ZIP	Char	9	
AREA	Decimal	3	
PHONE	Char	9	
REPID	Decimal	4	
CREDITLIMIT	Decimal	9	2
COMMENTS	VarChar	254	

The modify stage is used to drop the REPID, CREDITLIMIT, and COMMENTS columns. To do this, the stage properties are set as follows:

```
Specification = DROP REPID, CREDITLIMIT, COMMENTS
```

The easiest way to specify the outgoing meta data in this example would be to use runtime column propagation. You could, however, choose to specify the meta data manually, in which case it would look like:

Column name	SQL Type	Length	Scale
CUSTID	Decimal	6	
NAME	Char	45	
ADDRESS	Char	40	
CITY	Char	30	
STATE	Char	2	
ZIP	Char	9	
AREA	Decimal	3	
PHONE	Char	9	

You could achieve the same effect by specifying which columns to keep, rather than which ones to drop. In the case of this example the required specification to use in the stage properties would be:

```
KEEP CUSTID, NAME, ADDRESS, CITY, STATE, ZIP, AREA, PHONE
```

Changing data type

You could also change the data types of one or more of the columns from the above example. Say you wanted to convert the CUSTID from decimal to string, you would specify a new column to take the converted data, and specify the conversion in the stage properties:

Column name	SQL Type	Length	Scale
conv_CUSTID	Char	20	
NAME	Char	45	
ADDRESS	Char	40	
CITY	Char	30	
STATE	Char	2	
ZIP	Char	9	
AREA	Decimal	3	
PHONE	Char	9	
REPID	Decimal	4	
CREDITLIMIT	Decimal	9	2
COMMENTS	VarChar	254	

```
Specification = conv_CUSTID = CUSTID
```

Some data type conversions require you to use a transform command, a list of these, and the available transforms, is given in "Specification" . The decimal to string conversion is one that can be performed using an explicit transform. In this case, the specification on the Properties page is as follows:

```
conv_CUSTID:string = string_from_decimal(CUSTID)
```

Null handling

You can also use the Modify stage to handle columns that might contain null values. Any of the columns in the example, other than CUSTID, could legally contain a null value. You could use the modify stage to detect when the PHONE column contains a null value, and handle it so no errors occur. In this case, the specification on the Properties page would be:

```
PHONE = handle_null (PHONE,-128)
```

Other null handling transforms are described in "Specification" .

Modify stage: fast path
About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Modify stages in a job. This section specifies the minimum steps to take to get a Modify stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Modify stage:

- In the Stage page **Properties Tab**, supply the Modify specification.
- Ensure you have specified the meta data for the input and output columns

Modify stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Modify stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. The modify stage only has one property, although you can repeat this as required.

Table 41. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Specification	string	N/A	Y	Y	N/A

Modify stage: Options category: Specification

This is a statement with one of the following the forms:

- DROP *columnname* [, *columnname*]
- KEEP *columnname* [, *columnname*]
- *new_columnname* [:*new_type*] = [*explicit_conversion_function*] *old_columnname*

If you choose to drop a column or columns, all columns are retained except those you explicitly drop. If you chose to keep a column or columns, all columns are excluded except those you explicitly keep.

If you specify multiple specifications each will be carried out sequentially.

Some type conversions InfoSphere DataStage can carry out automatically, others need you to specify an explicit conversion function. Some conversions are not available.

The following table summarizes the availability, with the source fields shown vertically and the target fields shown horizontally. A value of "d" indicates automatic (default) conversion, "m" indicates that manual conversion is required, a blank square indicates that conversion is not possible:

	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
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

	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	

For a default type conversion, your specification would take the following form:

```
new_columnname [:new_type] = [explicit_conversion_function] old_columnname
```

For example, to produce an int8 column type:

```
int8col:int8 = uint64col
```

Where a manual conversion is required, your specification takes the form:

```
new_columnname:new_type = conversion_function (old_columnname)
```

For example:

```
day_column:int8 = month_day_from_date (date_column)
```

The *new_type* can be any of the destination types that are supported for conversions from the source (that is, any of the columns marked "m" in the above table). 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. InfoSphere DataStage warns you when it is performing an implicit data type conversion, for example **hours_from_time** expects to convert a time to an int8, and will warn you if converting to a int16, int32, or dfloat.

The following table lists the available conversion functions. The source and destinations are always specified in terms of column names. Preliminary arguments are enclosed in square brackets, the source column name is enclosed in round brackets.

Conversion	Arguments	Output type	Description	Example
date_from_days_ since	[*base_date* (date)] (*number_col* (int32))	date	Converts an integer field into a date by adding the integer to the specified base date. The base_date must be in the format *yyyy-mm-dd* and must be either double quoted or a variable.	date_col:date = date_from_days_since ["1958-08-18"] (int_col)
date_from_julian_day	(*juliandate_col* (uint32))	date	Date from Julian day.	date_col:date = date_from_julian_day (julian_col)
date_from_string	[*date_format*] (*string_col* (string))	date	Converts the string to a date representation using the specified *date_format*. By default the string format is *yyyy-mm-dd*.	date_col:date = date_from_string ["%yyyy-%mm-%dd"] (string_col)
date_from_timestamp	(*timestamp_col* (timestamp))	date	Converts the timestamp to a date representation.	date_col:date = date_from_timestamp (ts_col)

Conversion	Arguments	Output type	Description	Example
date_from_ustring	[*date_format*] (*string_col* (ustring))	date	Converts the string to a date representation using the specified *date_format*. By default the string format is *yyyy-mm-dd*.	date_col:date = date_from_ustring (ustring_col, "%yyyy-%mm-%dd")
days_since_from_date	[*source_date* (date)] (*date_col* (string))	int32	Returns a value corresponding to the number of days from *source_date* to the specified date. *source_date* must be in the form *yyyy-mm-dd* and can be quoted or unquoted.	dayssince_col:int32 = days_since_from_date ["1958-08-18"] (sourcedate_col,)
decimal_from_decimal	[*r_type*] (*source_decimal_col* (decimal))	decimal	Decimal from decimal.	decimal_col:decimal = decimal_from_decimal [ceil] (source_col)
decimal_from_dfloat	[*r_type*] (*source_dfloat_col* (dfloat))	decimal	Decimal from dfloat.	decimal_col:decimal = decimal_from_dfloat [ceil] (source_col)
decimal_from_string	[*r_type*] (*source_string_col* (string))	decimal	Decimal from string.	decimal_col:decimal = decimal_from_string [ceil] (source_col)
decimal_from_ustring	[*r_type*] (*source_ustring_col* (ustring))	decimal	Decimal from ustring.	decimal_col:decimal = decimal_from_ustring [ceil] (source_col)
dfloat_from_decimal	[fix_zero] (*source_dec_col* (decimal))	dfloat	Dfloat from decimal.	dfloat_col:dfloat = dfloat_from_decimal [fix_zero] (source_col)
hours_from_time	(*source_time_col* (time))	int8	Hours from time.	hours_col:int8 = hours_from_time (time_col)
int32_from_decimal	[*r_type*, fix_zero] (*source_decimal_col* (decimal))	int32	Int32 from decimal.	int32_col:int32 = int32_from_decimal [ceil] [fix_zero] (dec_col)
int64_from_decimal	[*r_type*, fix_zero] (*source_decimal_col* (decimal))	int64	Int64 from decimal.	int64_col:int64 = int64_from_decimal [ceil] (dec_col)
julian_day_from_date	(*date_col* (date))	uint32	Julian day from date.	julianday_col:uint32 = julian_day_from_from_date (date_col)
lookup_string_from _int16	[*table_definition*], (*number_col* (int16))	string	Converts numeric values to strings by means of a lookup table.	gendercol:string = lookup_string_from_int16 [{default_value = 2} ('f' = 1; 'm' = 2)] (gendercode)

Conversion	Arguments	Output type	Description	Example
lookup_ustring_from _int16	[*table_definition*] (*number_col* (int16))	ustring	Converts numeric values to ustrings by means of a lookup table.	gendercol:ustring = lookup_ustring_from_int16 [{default_value = 2} ('f' = 1; 'm' = 2)] (gendercode)
lookup_ustring_from _int32	[*table_definition*] (*number_col* (int32))	ustring	Converts numeric values to ustrings by means of a lookup table..	gendercol:ustring = lookup_string_from_int32 [{default_value = 2} ('f' = 1; 'm' = 2)] (gendercode)
lookup_string_from _uint32	[*table_definition*] (*number_col* (uint32))	string	Converts numeric values to strings by means of a lookup table.	gendercol:string = lookup_string_from_uint16 [{default_value = 2} ('f' = 1; 'm' = 2)] (gendercode)
lookup_int16_from _string	[*table_definition*] (*string_col* (string))	int16	Converts strings to numeric values by means of a lookup table.	int_col:int16 = lookup_int16_from_string [{default_value = 2} ('f' = 1; 'm' = 2)] (gendercode)
lookup_int16_from _ustring	[*table_definition*] (*ustring_col* (ustring))	int16	Converts strings to numeric values by means of a lookup table.	int_col:int16 = lookup_int16_from_ustring [{default_value = 2} ('f' = 1; 'm' = 2)] (gendercode)
lookup_uint32_from _string	[*table_definition*] (*string_col* (string))	uint32	Converts strings to numeric values by means of a lookup table.	int_col:uint32 = lookup_uint32_from_string [{default_value = 2} ('f' = 1; 'm' = 2)] (gendercode)
lookup_uint32_from _ustring	[*table_definition*] (*ustring_col* (ustring))	uint32	Converts ustrings to numeric values by means of a lookup table.	int_col:uint32 = lookup_uint32_from_ustring [{default_value = 2} ('f' = 1; 'm' = 2)] (gendercode)
lowercase_string	(*instring_col* (string))	string	Convert strings to all lower case. Non-alphabetic characters are ignored in the conversion.	ostring_col:string = lowercase_string (istring_col)
lowercase_ustring	(*instring_col* (ustring))	string	Convert ustrings to all lower case. Non-alphabetic characters are ignored in the conversion.	ostring_col:ustring = lowercase_string (istring_col)
mantissa_from _decimal	(*decimal_col* (decimal))	dfloat	Returns the mantissa from the given decimal	matissa_col:dfloat = mantissa_from_decimal (dec_col)
mantissa_from_dfloat	(*dfloat_col* (dfloat))	dfloat	Returns the mantissa from the given dfloat	matissa_col:dfloat = mantissa_from_dfloat (dfloat_col)
microseconds _from_time	(*time_col* (time))	int32	Returns the microseconds from a time field.	msec_col:int32 = microseconds_from_time (time_col)

Conversion	Arguments	Output type	Description	Example
midnight_seconds _from_time	(*time_col* (time))	dfloat	Returns the seconds-from-midnight from the supplied time.	midsec_col:dfloat = midnight_seconds_from_time (time_col)
minutes_from_time	(*time_col* (time))	int8	Returns the minutes from a time field.	minsec_col:int8 = minutes_from_time (time_col)
month_day _from_date	(*date_col* (date))	int8	Returns the day of month from a date field.	monthday_col:int8 = month_day_from_date (date_col)
month_from_date	(*date_col* (date))	int8	Returns the numeric month from a date field.	month_col:int8 = month_from_date (date_col)
next_weekday_from _date	[*day*] (*date_col* (date))	date	Returns the date of the specified day of the week soonest after the source date (including the source date). *day* is a string specifying a day of the week. You can specify *day* by either the first three characters of the day name or the full day name. The *day* can be quoted in either single or double quotes or quotes can be omitted.	nextday_col:date = next_weekday_from_date [wed](date_col)
notnull	(*any*)	int8	Returns true (1) when an expression does not evaluate to the null value.	isnotnull_col:int8 = notnull (test_col)
null	(*any*)	int8	Returns true (1) when an expression does evaluate to the null value	isnull_col:int8 = null (test_col)

Conversion	Arguments	Output type	Description	Example
previous_weekday _from_date	[*day*] (*date_col* (date))	date	The destination contains the closest date for the specified day of the week earlier than the source date (including the source date). The *day* is a string specifying a day of the week. You can specify *day* by either the first three characters of the day name or the full day name. The *day* can be quoted in either single or double quotes or quotes can be omitted.	prevday_col:date = previous_weekday_from_date [wed](date_col)
raw_from_string	(*string_col* (string))	raw	Returns a string in raw representation.	raw_col:raw = raw_from_string (string_col)
raw_length	(*raw_col* (raw))	int32	Returns the length of a raw field.	rawlength_col:int32 = raw_length (raw_col)
seconds_from_time	(*time_col* (time))	dfloat	Returns the seconds from a time field.	sec_col:dfloat = seconds_from_time (time_col)
seconds_since_from _timestamp	(*timestamp_col* (timestamp))	dfloat	Seconds since the time given by *timestamp.*	secsince_col:dfloat = seconds_since_from_timestamp (timestamp_col)
string_from_date	[*date_format*] (*date_col* (date))	string	Converts the date to a string representation using the specified *date_format*.	datestring_col:string = string_from_date [%dd-%mm-%yyyy] (date_col)
string_from_decimal	[fix_zero] (*decimal_col* (decimal)	string	Returns a string from a decimal.	string_col:string = string_from_decimal [fix_zero] (dec_col)
string_from_time	[*time_format*] (*time_col* (time))	string	Converts the time to a string representation using the specified *time_format*. The default time format is %hh:%nn:%ss.	timestring_col:string = string_from_time [%hh:%nn:%ss.] (time_col)

Conversion	Arguments	Output type	Description	Example
string_from _timestamp	[*timestamp_format*] (*timestamp_col* (timestamp))		Converts the timestamp to a string representation using the specified *timestamp _format*. The default timestamp format is %yyyy-%mm-%dd. %hh:%nn:%ss.	stringtimestamp_col:string = string_from_timestamp [%yyyy-%mm-%dd. %hh:%nn:%ss.] (timestamp_col)
string_from_ustring	(*string_col* (ustring))	string	Returns a string from a ustring.	string_col:string = string_from_ustring (ustring_col)
string_length	(*string_col* (string))	int32	Returns an int32 containing the length of a string.	length_col:int32 = string_length (string_col)
substring	[*startPosition,len*] (*string_col* (string))	string	Converts long strings to shorter strings by string extraction. The *startPosition* specifies the starting location of the substring; *len* specifies the substring length. If *startPosition* is positive, it specifies the byte offset into the string from the beginning of the string. If *startPosition* is negative, it specifies the byte offset from the end of the string.	shorstring_col:string = substring [5,10] (longstring_col)
time_from_midnight _seconds	(*dfloat_col* (dfloat))	time	Returns a time from aseconds-from-midnight field.	time_col:time = time_from_midnight_seconds (dfloat_col)
time_from_string	[*time_format*] (*string_col* (string))	time	Converts the string to a time representation using the specified *time_format*. The default time format is %hh:%nn:%ss.	time_col:time = time_from_string [%hh:%nn:%ss.] (string_col)
time_from_timestamp	(*timestamp_col* (timestamp))	time	Time from timestamp.	time_col:time = time_from_timestamp (timestamp_col)

Conversion	Arguments	Output type	Description	Example
time_from_ustring	(*string_col* (ustring))	time	Returns a time from a ustring.	time_col:time = time_from_ustring (string_col)
timestamp_from_date	[*time*](*date_col* (date))	time stamp	Timestamp from date. The *time* 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 time defaults to midnight.	timestamp_col:timestamp = timestamp_from_date [08:20:33] (date_col)
timestamp_from _seconds_since	(*secondssince_col* (dfloat))	time stamp	Timestamp from a seconds since value.	timestamp_col:timestamp = timestamp_from_seconds_since (secondssince_col)
timestamp_from _string	[*timestamp_format*] (*string_col* (string))	time stamp	Converts the string to a timestamp representation using the specified *timestamp _format*. By default, the string format is %yyyy-%mm-%dd hh:nn:ss.	timestamp_col:timestamp = timestamp_from_string [%yyyy-%mm-%dd hh:nn:ss] (string_col)
timestamp_from_time	[*date*](*time_col* (time))	time stamp	Timestamp from time. The *date* argument is required. It specifies the date portion of the timestamp and must be in the form *yyyy-mm-dd*.	timestamp_col:timestamp = timestamp_from_time [1958-08-18] (time_col)
timestamp_from _timet	(*timet_col* (int32))	time stamp	Timestamp from time_t. The source field must contain a timestamp as defined by the UNIX time_t representation.	timestamp_col:timestamp = timestamp_from_timet (timet_col)
timestamp_from _ustring	(*string_col* (ustring))	time stamp	Returns a timestamp from a ustring.	timestamp_col:timestamp = timestamp_from_ustring (string_col)

Conversion	Arguments	Output type	Description	Example
timet_from _timestamp	(*tstamp_col* (timestamp))	int32	Time_t from timestamp. The destination column contains a timestamp as defined by the UNIX time_t representation.	timet_col:int32 = timet_from_timestamp (tstamp_col)
uint64_from_decimal	[*r_type,* fix_zero] (*dec_col* (decimal))	uint64	Uint64 from decimal.	int_col:uint64 = uint64_from_decimal [ceil, fix_zero] (dec_col)
uppercase_string	(*string_col* (string))	string	Convert strings to all upper case. Non-alphabetic characters are ignored in the conversion.	string_col:string = uppercase_string (instring_col)
uppercase_ustring	(*string_col* (ustring))	ustring	Convert ustrings to all upper case. Non-alphabetic characters are ignored in the conversion.	ustring_col:string = uppercase_ustring (string_col)
u_raw_from_string	(*string_col* (ustring))	raw	Returns a raw from a ustring	raw_col:raw = u_raw_from_string (string_col)
ustring_from_date	(*date_col* (date))	ustring	Returns a ustring from a date.	string_col:ustring = ustring_from_date (date_col)
ustring_from_decimal	(*dec_col* (decimal))	ustring	Returns a ustring from a decimal.	string_col:ustring = ustring_from_decimal (dec_col)
ustring_from_string	(*string_col* (string))	ustring	Returns a ustring from a string.	string_col:ustring = ustring_from_string (string_col)
ustring_from_time	(*time_col* (time))	ustring	Returns a ustring from a time.	string_col:ustring = ustring_from_time (time_col)
ustring_from _timestamp	(*timestamp_col* (timestamp))	ustring	Returns a ustring from a timestamp.	string_col:ustring = ustring_from_timestamp (timestamp_col)
ustring_length	(*string_col* (ustring))	int32	Returns the length of a ustring.	length_col:int32 = ustring_length (string_col)

Conversion	Arguments	Output type	Description	Example
u_substring	[*startPosition*,*len*] (*string_col* (string))	ustring	Converts long ustrings to shorter ustrings by string extraction. The *startPosition* specifies the starting location of the substring; *len* specifies the substring length. If *startPosition* is positive, it specifies the byte offset into the string from the beginning of the string. If *startPosition* is negative, it specifies the byte offset from the end of the string.	shorstring_col:ustring = substring [5,10] (longstring_col)
weekday_from_date	[*originDay*] (*date_col* (date))	int8	Day of week from date. *originDay* 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 *originDay* can be either single- or double-quoted or the quotes can be omitted.	dow_int:int8 = [mon] (date_col)
year_day_from_date	(*date_col* (date))	int16	Day of year from date (returned value 1-366).	doy_col:int16 = year_day_from_date (date_col)
year_from_date	(*date_col* (date))	int16	Year from date.	year_col:int16 = year_from_date (date_col)
year_week_from_date	(*date_col* (date))	int8	Week of year from date.	week_col:int8 = year_week_from_date (date_col)

table_definition defines the rows of a string lookup table and has the following form:

```
{propertyList} ('string' = value; 'string' = 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* specifies a comma-separated list of strings associated with value; enclose each string in quotes.
- *value* specifies a comma-separated list of 16-bit integer values associated with *string*.

date_format is the standard date formatting string described in "Date and time formats" on page 31

R_type is a string representing the rounding type and should contain one of the following:
- **ceil**. Round the source field toward positive infinity. E.g, 1.4 -> 2, -1.6 -> -1.
- **floor**. Round the source field toward negative infinity. E.g, 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. E.g, 1.4 -> 1, 1.5 -> 2, -1.4 -> -1, -1.5 -> -2.
- **trunc_zero**. Discard any fractional digits to the right of the rightmost 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. E.g, 1.6 -> 1, -1.6 -> -1.

You can specify fix_zero for decimal source columns so that columns containing all zeros (by default illegal) are treated as a valid decimal with a value of zero.

Modify stage: Advanced tab

This tab allows you to specify the following:
- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. If you have an input data set, it adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request the next stage should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Modify stage: Input page

The Input page allows you to specify details about the incoming data set you are modifying. There is only one input link.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Modify stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Modify stage: Partitioning on input links

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the modify is performed.

By default the stage uses the auto partitioning method.

If the Modify stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Modify stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Modify stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Modify stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Modify stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Modify stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.

- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operation starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the modify operation is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

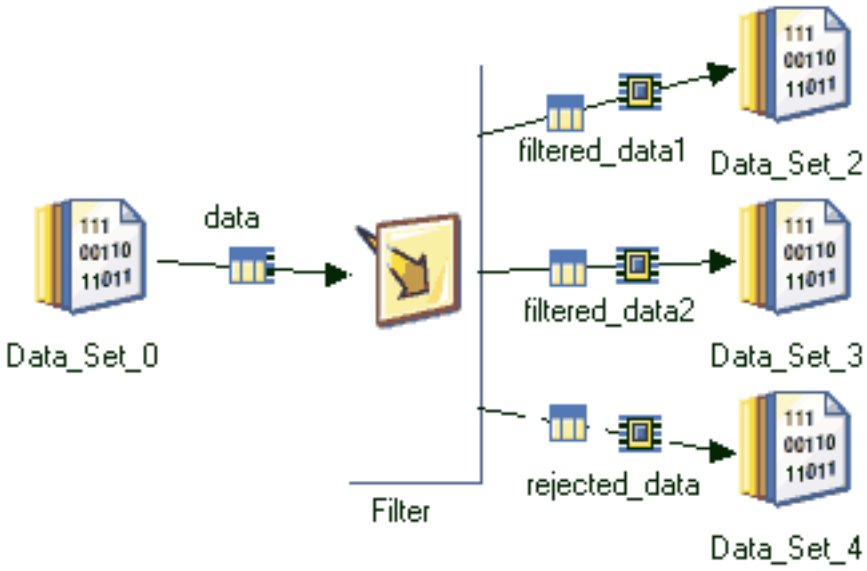
Modify stage: Output page

See "Stage Editors," for a general description of the output tabs.

Filter Stage

The Filter stage is a processing stage. It can have a single input link and a any number of output links and, optionally, a single reject link.

The Filter stage transfers, unmodified, the records of the input data set which satisfy the specified requirements and filters out all other records. You can specify different requirements to route rows down different output links. The filtered out records can be routed to a reject link, if required.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the input link carrying the data to be filtered.
- **Output Page**. This is where you specify details about the filtered data being output from the stage down the various output links.

Specifying the filter

About this task

The operation of the filter stage is governed by the expressions you set in the **Where** property on the Properties tab. You can use the following elements to specify the expressions:

- Input columns.
- Requirements involving the contents of the input columns.
- 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 the specified output link. The **Where** property supports standard SQL expressions, except when comparing strings.

When quoting in the filter, you should use single, not double, inverted commas.

Input data columns

If you specify a single column for evaluation, that column can be of any data type. Note that InfoSphere DataStage's treatment of strings differs slightly from that of standard SQL. If you compare columns they must be of the same or compatible data types. Otherwise, the operation terminates with an error. Compatible data types are those that InfoSphere DataStage converts by default. Regardless of any conversions the whole row is transferred unchanged to the output. If the columns are not compatible upstream of the filter stage, you can convert the types by using a Modify stage prior to the Filter stage.

Column 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.

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.

Note: The conversion of numeric data types might result in a loss of range and cause incorrect results. InfoSphere DataStage displays a warning message to that effect when range is lost.

Supported Boolean expressions and operators

The following list summarizes the Boolean expressions that are supported. In the list, BOOLEAN denotes any Boolean expression.

- true
- false
- six comparison operators: =, <>, <, >, <=, >=
- is null
- is not null
- like 'abc' (the second operand must be a regular expression)
- between (for example, A between B and C is equivalent to B <= A and A> = C)
- not BOOLEAN
- BOOLEAN is true
- BOOLEAN is false
- BOOLEAN is not true
- BOOLEAN is not false

Any of these can be combined using AND or OR.

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

InfoSphere DataStage sorts 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.

Examples

The following give some example **Where** properties.

Comparing two fields:

You want to compare columns number1 and number2. If the data in column number1 is greater than the data in column number2, the corresponding records are to be written to output link 2.

You enter the following in the **Where** property:

```
name1 > name2
```

You then select output link 2 in the dependent **Output Link** property. (You use the Link Ordering tab to specify the number order of the output links).

Testing for a null:

You want to test column serialno to see if it contains a null. If it does, you want to write the corresponding records to the output link.

You enter the following in the **Where** property:

```
serialno is null
```

In this example the stage only has one output link. You do not need to specify the **Output Link** property because the stage will write to the output link by default.

Evaluating input columns:

You want to evaluate each input row to see if these conditions prevail:

- EITHER all the following are true
  - Column number1 does not have the value 0
  - Column number2 does not have the value 3
  - Column number3 has the value 0
- OR column name equals the string ZAG

You enter the following in the **Where** property:

```
number1 <> 0 and number2 <> 3 and number3 = 0 or name = 'ZAG'
```

If these conditions are met, the stage writes the row to the output link.

Filter stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Filter stages in a job. This section specifies the minimum steps to take to get a Filter stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Filter stage:

- In the Stage page **Properties Tab**:
  - Supply the specifications that determine which records are accepted and which are filtered out. This is given in the form of a Where clause. You can multiple statements each applying to different links.
  - Specify which Where clause correspond to which output links.
  - Specify whether rows that fail to satisfy any of the Where clauses will be routed to a reject link.
  - Specify whether rows are output only for the first Where clause they satisfy, or for any clauses they satisfy.
- In the Stage page **Link Ordering Tab**, specify which order the output links are processed in. This is important where you specify that rows are only output for the first Where clause that they satisfy.
- Ensure that meta data has been defined for input link and output links, and reject link, if applicable.
- In the Output Page **Mapping Tab**, specify how the input columns of the data set being filtered map onto the columns of the various output links.

Filter stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The Link Ordering tab allows you to specify what order the output links are processed in. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Filter stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 42. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Predicates/Where clause	string	N/A	Y	Y	N/A
Predicates/ Output link	Output link	N/A	Y	N	Where clause
Options/Output rejects	True/False	False	Y	N	N/A
Options/Output rows only once	True/False	False	Y	N	N/A
Options/Nulls value	Less Than/Greater Than	Less Than	N	N	N/A

Predicates category:
Filter stage: Where clause

Specify a Where statement that a row must satisfy in order to be routed down this link. This is like an SQL Where clause, see "Specifying the Filter" for details.

Output link

Specify the output link corresponding to the Where clause.

Filter stage: Options category:
Output rejects

Set this to true to output rows that satisfy no Where clauses down the reject link (remember to specify which link is the reject link on the parallel job canvas).

Output rows only once

Set this to true to specify that rows are only output down the link of the first Where clause they satisfy. Set to false to have rows output down the links of all Where clauses that they satisfy.

Nulls value

Specify whether null values are treated as greater than or less than other values.

Filter stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts the setting of the previous stage.You can explicitly select **Set** or **Clear**. Select **Set** to request the stage should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Filter stage: Link Ordering tab

This tab allows you to specify the order in which output links are processed. This is important where you have set the Output rows only once property to True.

Filter stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Filter stage uses this when evaluating Where clauses. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Filter stage: Input page

The Input page allows you to specify details about the data set being filtered. There is only one input link.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Filter stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Filter stage: Partitioning on input links

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the filter is performed.

By default the stage uses the auto partitioning method.

If the Filter stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Filter stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Filter stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Filter stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Filter stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Filter stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the remove duplicates operation is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the Selected list and right-click to invoke the shortcut menu.

Filter stage: Output page

The Output page allows you to specify details about data output from the Filter stage. The stage can have any number of output links, plus one reject link, choose the one you want to work on from the **Output name** drop down list.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Filter stage and the output columns. The Advanced tab allows you to change the default buffering settings for the output links.

Details about Filter stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Filter stage: Mapping tab

For Filter stages the Mapping tab allows you to specify how the output columns are derived, that is, what filtered columns map onto them.

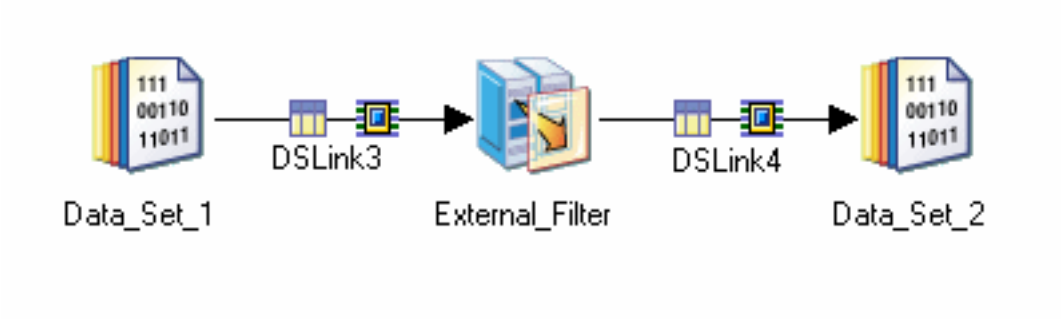
The left pane shows the filtered columns. These are read only and cannot be modified on this tab.

The right pane shows the output columns for the output link. This has a **Derivations** field where you can specify how the column is derived.You can fill it in by dragging copied columns over, or by using the Auto-match facility.

External Filter stage

The External Filter stage is a processing stage. It can have a single input link and a single output link.

The External Filter stage allows you to specify a UNIX command that acts as a filter on the data you are processing. An example would be to use the stage to *grep* a data set for a certain string, or pattern, and discard records which did not contain a match. This technique can be a quick and efficient way of filtering data.



Whitespace is stripped from the start and end of the data before the command is executed. To avoid this behavior, use an explicitly wrapped command that sets format options on the schema.

The stage editor has three pages:

- **Stage Page**. Use this page to specify general information about the stage.
- **Input Page**. Use this page to specify details about the input link carrying the data to be filtered.
- **Output Page**. Use this page to specify details about the filtered data being output from the stage.

External Filter stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include External Filter stages in a job. This section specifies the minimum steps to take to get an External Filter stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use an External Filter stage:

- In the Stage page **Properties Tab** specify the filter command the stage will use. Optionally add arguments that the command requires.

External Filter stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

External Filter stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 43. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Filter Command	string	N/A	Y	N	N/A
Options/ Arguments	string	N/A	N	N	N/A

External Filter stage: Options category:
Filter command

Specifies the filter command line to be executed and any command line options it requires. For example:

```
grep
```

Arguments

Allows you to specify any arguments that the command line requires. For example:

```
\(cancel\).*\1
```

Together with the *grep* command would extract all records that contained the string "cancel" twice and discard other records.

External Filter stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts the setting of the previous stage.You can explicitly select **Set** or **Clear**. Select **Set** to request the next stage should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

External Filter stage: Input page

The Input page allows you to specify details about the data set being filtered. There is only one input link.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about External Filter stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

External Filter stage: Partitioning on input links

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the filter is executed.

By default the stage uses the auto partitioning method.

If the External Filter stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the External Filter stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the External Filter stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the External Filter stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the External Filter stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the External Filter stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the remove duplicates operation is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

External Filter stage: Output page

The Output page allows you to specify details about data output from the External Filter stage. The stage can only have one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

See "Stage Editors," for a general description of these tabs.

Change Capture stage

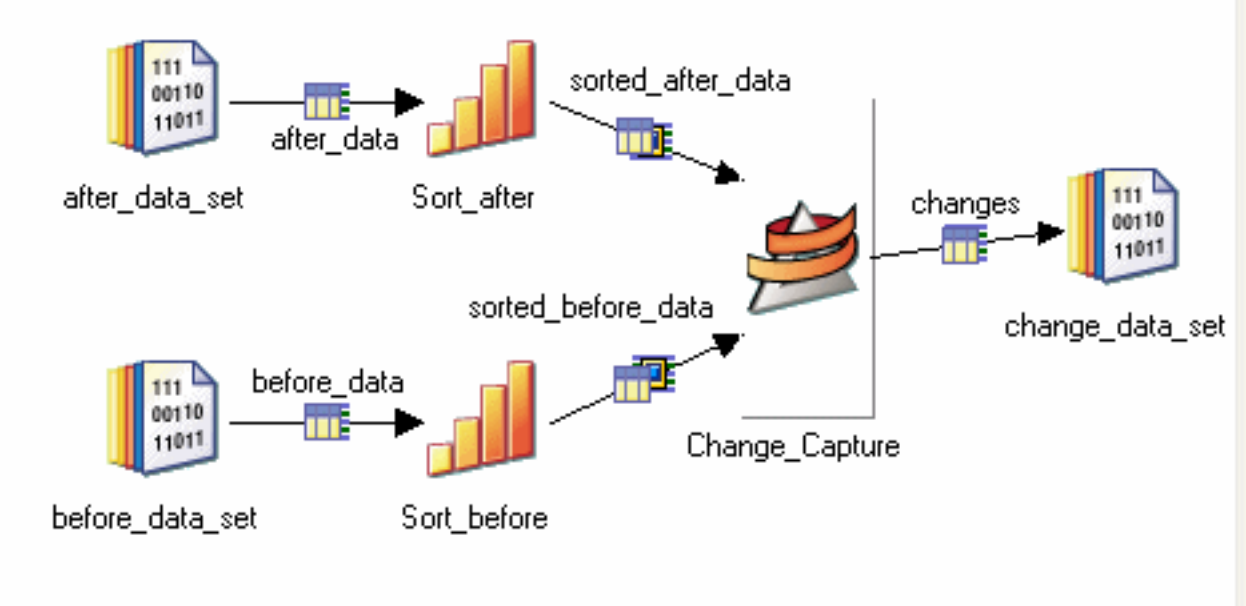
The Change Capture Stage is a processing stage. The stage compares two data sets and makes a record of the differences.

The Change Capture stage 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 stage produces a change data set, whose table definition is transferred from the after data set's table definition with the addition of one column: 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.

The compare is based on a set a set of key columns, rows from the two data sets are assumed to be copies of one another if they have the same values in these key columns. You can also optionally specify change values. If two rows have identical key columns, you can compare the value columns in the rows to see if one is an edited copy of the other.

The stage assumes that the incoming data is key-partitioned and sorted in ascending order. The columns the data is hashed on should be the key columns used for the data compare. You can achieve the sorting and partitioning using the Sort stage or by using the built-in sorting and partitioning abilities of the Change Capture stage.

You can use the companion Change Apply stage to combine the changes from the Change Capture stage with the original before data set to reproduce the after data set (see “Switch stage” on page 376).



The Change Capture stage is very similar to the Difference stage described in "Difference stage" on page 355.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data set having its duplicates removed.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Example Data

This example shows a *before* and *after* data set, and the data set that is output by the Change Capture stage when it has compared them.

This is the *before* data set:

Table 44. Before data set

bcol0	bcol1	bcol2	bcol3	bcol4
0	0	0	0	a
1	7	1	1	b
2	2	2	2	c
3	3	3	3	d
4	5	4	4	e
5	2	5	5	f
6	6	6	6	g
7	7	7	7	h
8	8	8	8	i
9	9	9	9	j

This is the *after* data set:

Table 45. After data set

bcol0	bcol1	bcol2	bcol3	bcol4
0	0	0	0	a
1	1	1	1	b
2	2	2	2	c
3	3	3	3	d
4	4	4	4	e
5	5	5	5	f
6	6	6	6	g
7	7	7	7	h
8	8	8	8	i
9	9	9	9	j

This is the data set output by the Change Capture stage (bcol4 is the key column, bcol1 the value column):

Table 46. Change data set

bcol0	bcol1	bcol2	bcol3	bcol4	change_code
1	1	1	1	b	3
4	4	4	4	e	3
5	5	5	5	f	3

The change_code indicates that, in these three rows, the bcol1 column in the after data set has been edited. The bcol1 column carries the edited value.

Change Capture stage: fast path

This section specifies the minimum steps to take to get a Change Capture stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Change Capture stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

1. Go to the **Properties Tab** on the Stage page.
2. Specify the key column. You can repeat this property to specify a composite key. *Before* and *after* rows are considered to be the same if they have the same value in the key column or columns.
3. Optionally specify one or more Value columns. This enables you to determine if an *after* row is an edited version of a *before* row.

   Note: You can also set the Change Mode property to have InfoSphere DataStage treat all columns not defined as keys treated as values, or all columns not defined as values treated as keys.
4. Specify whether the stage will output the changed row or drop it. You can specify this individually for each type of change (copy, delete, edit, or insert).
5. In the Stage page **Link Ordering Tab**, specify which of the two links carries the *before* data set and which carries the *after* data set.

6. If the two incoming data sets aren't already key partitioned on the key columns and sorted, set InfoSphere DataStage to do this on the **Input Page Partitioning Tab**.
7. In the **Output Page Mapping Tab**, specify how the change data columns are mapped onto the output link columns.

Change Capture stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The Link Ordering tab allows you to specify which input link carries the *before* data set and which the *after* data set. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Change Capture stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 47. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Change Keys/Key	Input Column	N/A	Y	Y	N/A
Change Keys/Case Sensitive	True/False	True	N	N	Key
Change Keys/Sort Order	Ascending/ Descending	Ascending	N	N	Key
Change Keys/Nulls Position	First/Last	First	N	N	Key
Change Values/Value	Input Column	N/A	N	Y	N/A
Change Values/Case Sensitive	True/False	True	N	N	Value
Options/Change Mode	Explicit Keys & Values/All keys, Explicit values/Explicit Keys, All Values	Explicit Keys & Values	Y	N	N/A
Options/Log Statistics	True/False	False	N	N	N/A
Options/Drop Output for Insert	True/False	False	N	N	N/A
Options/Drop Output for Delete	True/False	False	N	N	N/A
Options/Drop Output for Edit	True/False	False	N	N	N/A

Table 47. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Drop Output for Copy	True/False	True	N	N	N/A
Options/Code Column Name	string	N/A	N	N	N/A
Options/Copy Code	number	0	N	N	N/A
Options/Deleted Code	number	2	N	N	N/A
Options/Edit Code	number	3	N	N	N/A
Options/Insert Code	number	1	N	N	N/A

Change Capture stage: Change Keys category: Key

Specifies the name of a difference key input column. This property can be repeated to specify multiple difference key input columns. You can use the Column Selection dialog box to select several keys at once if required. Key has the following dependent properties:

- **Case Sensitive**

  Use this property to specify whether each key is case sensitive or not. It is set to True by default; for example, the values "CASE" and "case" would not be judged equivalent.
- **Sort Order**

  Specify ascending or descending sort order.
- **Nulls Position**

  Specify whether null values should be placed first or last.

Change Capture stage: Change Value category: Value

Specifies the name of a value input column. You can use the Column Selection dialog box to select values at once if required . Value has the following dependent properties:

- **Case Sensitive**

  Use this to property to specify whether each value is case sensitive or not. It is set to True by default; for example, the values "CASE" and "case" would not be judged equivalent.

Change Capture stage: Options category: Change mode

This mode determines how keys and values are specified. Choose Explicit Keys & Values to specify the keys and values yourself. Choose All keys, Explicit values to specify that value columns must be defined, but all other columns are key columns unless excluded. Choose Explicit Keys, All Values to specify that key columns must be defined but all other columns are value columns unless they are excluded.

Log statistics

This property configures the stage to display result information containing the number of input rows and the number of copy, delete, edit, and insert rows.

Drop output for insert

Specifies to drop (not generate) an output row for an insert result. By default, an output row is always created by the stage.

Drop output for delete

Specifies to drop (not generate) the output row for a delete result. By default, an output row is always created by the stage.

Drop output for edit

Specifies to drop (not generate) the output row for an edit result. By default, an output row is always created by the stage.

Drop output for copy

Specifies to drop (not generate) the output row for a copy result. By default, an output row is not created by the stage.

Code column name

Allows you to specify a different name for the output column carrying the change code generated for each record by the stage. By default the column is called change_code.

Copy code

Allows you to specify an alternative value for the code that indicates the *after* record is a copy of the *before* record. By default this code is 0.

Deleted code

Allows you to specify an alternative value for the code that indicates that a record in the *before* set has been deleted from the *after* set. By default this code is 2.

Edit code

Allows you to specify an alternative value for the code that indicates the *after* record is an edited version of the *before* record. By default this code is 3.

Insert Code

Allows you to specify an alternative value for the code that indicates a new record has been inserted in the *after* set that did not exist in the *before* set. By default this code is 1.

Change Capture stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.

- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pools or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Change Capture stage: Link Ordering tab

This tab allows you to specify which input link carries the *before* data set and which carries the *after* data set.

By default the first link added will represent the *before* set. To rearrange the links, choose an input link and click the up arrow button or the down arrow button.

Change Capture stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Change Capture stage uses this when it is determining the sort order for key columns. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Change Capture stage: Input page

The Input page allows you to specify details about the incoming data sets. The Change Capture expects two incoming data sets: a *before* data set and an *after* data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being compared. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Change Capture stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Change Capture stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is compared. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. In the case of the Change Capture stage, InfoSphere DataStage will determine if the incoming data is key partitioned. If it is, the Same method is used, if not, InfoSphere DataStage will hash partition the data and sort it. You could also explicitly choose hash and take advantage of the on-stage sorting.

If the Change Capture stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Change Capture stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Change Capture stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Change Capture stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Change Capture stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Change Capture stages. For the Change Capture stage, InfoSphere DataStage will ensure that the data is sorted as it is collected.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being compared. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.

- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Change Capture stage: Output page

The Output page allows you to specify details about data output from the Change Capture stage. The Change Capture stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Change Capture stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Change Capture stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Change Capture stage: Mapping tab

For the Change Capture stage the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them and which column carries the change code data.

The left pane shows the columns from the before/after data sets plus the change code column. These are read only and cannot be modified on this tab.

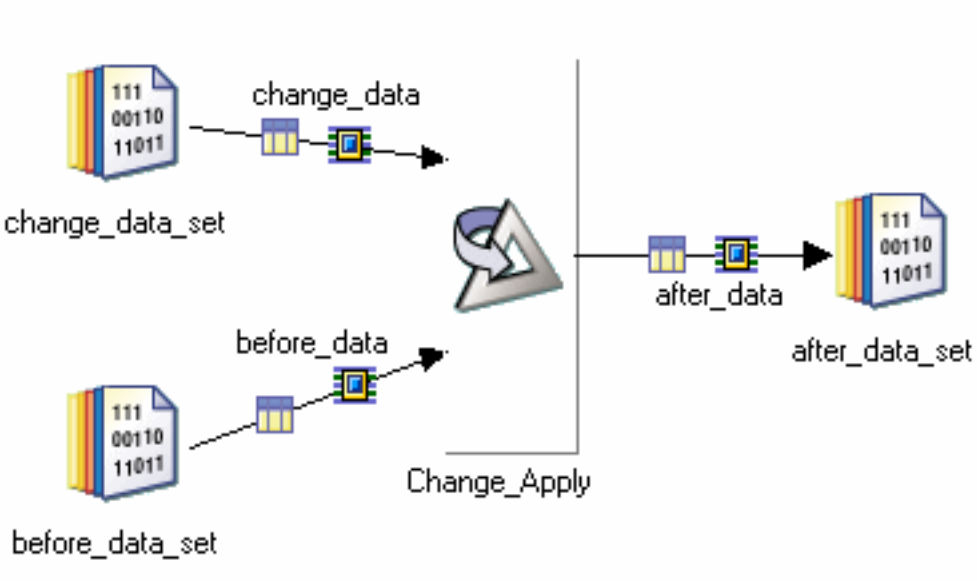
The right pane shows the output columns for each link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility. By default the data set columns are mapped automatically. You need to ensure that there is an output column to carry the change code and that this is mapped to the Change_code column.

Change Apply stage

The Change Apply stage is a processing stage. It takes the *change* data set, that contains the changes in the before and after data sets, from the Change Capture stage and applies the encoded change operations to a *before* data set to compute an *after* data set. (See “Change Capture stage” on page 339 for a description of the Change Capture stage.)

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

Note: The change input to Change Apply must have been output from Change Capture without modification. Because preserve-partitioning is set on the change output of Change Capture, you will be warned at run time if the Change Apply stage does not have the same number of partitions as the Change Capture stage. Additionally, both inputs of Change Apply are designated as partitioned using the Same partitioning method



.

The Change Apply stage reads a record from the *change* data set and from the *before* data set, compares their key column values, and acts accordingly:

- If the before keys come before the change keys in the specified sort order, the before record is copied to the output. The change record is retained for the next comparison.
- If the before keys are equal to the change keys, the behavior depends on the code in the change_code column of the change record:
  - Insert: The change record is copied to the output; the stage retains the same before record for the next comparison. If key columns are not unique, and there is more than one consecutive insert with the same key, then Change Apply applies all the consecutive inserts before existing records. This record order might be different from the after data set given to Change Capture.
  - Delete**:** The value columns of the before and change records are compared. If the value columns are the same or if the Check Value Columns on Delete is specified as False, the change and before records are both discarded; no record is transferred to the output. If the value columns are not the same, the before record is copied to the output and the stage retains the same change record for the next comparison. If key columns are not unique, the value columns ensure that the correct record is deleted. If more than one record with the same keys have matching value columns, the first-encountered record is deleted. This might cause different record ordering than in the after data set given to the Change Capture stage. A warning is issued and both change record and before record are discarded, that is, no output record results.
  - Edit: The change record is copied to the output; the before record is discarded. If key columns are not unique, then the first before record encountered with matching keys will be edited. This might be a different record from the one that was edited in the after data set given to the Change Capture stage. A warning is issued and the change record is copied to the output; but the stage retains the same before record for the next comparison.
  - Copy: The change record is discarded. The before record is copied to the output.
- If the before keys come after the change keys, behavior also depends on the change_code column:
  - **Insert.** The *change* record is copied to the output, the stage retains the same before record for the next comparison. (The same as when the keys are equal.)
  - **Delete**. A warning is issued and the *change* record discarded while the *before* record is retained for the next comparison.
  - **Edit** or **Copy**. A warning is issued and the *change* record is copied to the output while the *before* record is retained for the next comparison.

Note: If the before input of Change Apply is identical to the *before* input of Change Capture and either the keys are unique or copy records are used, then the output of Change Apply is identical to the *after* input of Change Capture. However, if the before input of Change Apply is not the same (different record contents or ordering), or the keys are not unique and copy records are not used, this is not detected and the rules described above are applied anyway, producing a result that might or might not be useful.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Example Data

This example shows a *before* and *change* data set, and the data set that is output by the Change Apply stage when it has compared them.

This is the *before* data set:

Table 48. Before data set

bcol0	bcol1	bcol2	bcol3	bcol4
0	0	0	0	a
1	7	1	1	b
2	2	2	2	c
3	3	3	3	d
4	5	4	4	e
5	2	5	5	f
6	6	6	6	g
7	7	7	7	h
8	8	8	8	i
9	9	9	9	j

This is the *change* data set, as output by a Change Capture stage:

Table 49. Change data set

bcol0	bcol1	bcol2	bcol3	bcol4	change_code
1	1	1	1	b	3
4	4	4	4	e	3
5	5	5	5	f	3

This is the *after* data set, output by the Change Apply stage (bcol4 is the key column, bcol1 the value column):

Table 50. After data set

bcol0	bcol1	bcol2	bcol3	bcol4
0	0	0	0	a
1	1	1	1	b

Table 50. After data set (continued)

bcol0	bcol1	bcol2	bcol3	bcol4
2	2	2	2	c
3	3	3	3	d
4	4	4	4	e
5	5	5	5	f
6	6	6	6	g
7	7	7	7	h
8	8	8	8	i
9	9	9	9	j

Change Apply stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Change Apply stages in a job. This section specifies the minimum steps to take to get a Change Apply stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Change Apply stage:

- In the Stage page **Properties Tab**:
  - Specify the key column. You can repeat this property to specify a composite key. *Before* and *change* rows are considered to be the same if they have the same value in the key column or columns.
  - Optionally specify one or more Value columns.

    (You can also set the Change Mode property to have InfoSphere DataStage treat all columns not defined as keys treated as values, or all columns not defined as values treated as keys.)
- In the Stage page **Link Ordering Tab**, specify which of the two links carries the *before* data set and which carries the *change* data set.
- In the **Output Page Mapping Tab**, specify how the change data columns are mapped onto the output link columns.

Change Apply stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Change Apply stage: Properties tab

The **Properties** tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 51. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Change Keys/Key	Input Column	N/A	Y	Y	N/A
Change Keys/Case Sensitive	True/False	True	N	N	Key
Change Keys/Sort Order	Ascending/ Descending	Ascending	N	N	Key
Change Keys/Nulls Position	First/Last	First	N	N	Key
Change Values/Value	Input Column	N/A	N	Y	N/A
Change Values/Case Sensitive	True/False	True	N	N	Value
Options/Change Mode	Explicit Keys & Values/All keys, Explicit values/Explicit Keys, All Values	Explicit Keys & Values	Y	N	N/A
Options/Log Statistics	True/False	False	N	N	N/A
Options/Check Value Columns on Delete	True/False	True	Y	N	N/A
Options/Code Column Name	string	N/A	N	N	N/A
Options/Copy Code	number	0	N	N	N/A
Options/Deleted Code	number	2	N	N	N/A
Options/Edit Code	number	3	N	N	N/A
Options/Insert Code	number	1	N	N	N/A

Change Apply stage: Change Keys category: Key

Specifies the name of a difference key input column. This property can be repeated to specify multiple difference key input columns. You can use the Column Selection dialog box to select several keys at once if required. Key has the following dependent properties:

- **Case Sensitive**

  Use this to property to specify whether each key is case sensitive or not. It is set to True by default; for example, the values "CASE" and "case" would not be judged equivalent.
- **Sort Order**

  Specify ascending or descending sort order.
- **Nulls Position**

Specify whether null values should be placed first or last.

Change Apply stage: Change Value category:
Value

Specifies the name of a value input column for an explanation of how Value columns are used). You can use the Column Selection dialog box to select several values at once if required. Value has the following dependent properties:

- **Case Sensitive**

  Use this to property to specify whether each value is case sensitive or not. It is set to True by default; for example, the values "CASE" and "case" would not be judged equivalent.

Change Apply stage: Options category:
Change mode

This mode determines how keys and values are specified. Choose Explicit Keys & Values to specify the keys and values yourself. Choose All keys, Explicit values to specify that value columns must be defined, but all other columns are key columns unless excluded. Choose Explicit Keys, All Values to specify that key columns must be defined but all other columns are value columns unless they are excluded.

Log statistics

This property configures the stage to display result information containing the number of input records and the number of copy, delete, edit, and insert records.

Check value columns on delete

Specifies that InfoSphere DataStage should not check value columns on deletes. Normally, Change Apply compares the value columns of delete *change* records to those in the *before* record to ensure that it is deleting the correct record.

Code column name

Allows you to specify that a different name has been used for the *change* data set column carrying the change code generated for each record by the stage. By default the column is called change_code.

Copy code

Allows you to specify an alternative value for the code that indicates a record copy. By default this code is 0.

Deleted code

Allows you to specify an alternative value for the code that indicates a record delete. By default this code is 2.

Edit code

Allows you to specify an alternative value for the code that indicates a record edit. By default this code is 3.

Insert code

Allows you to specify an alternative value for the code that indicates a record insert. By default this code is 1.

Change Apply stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the *Available Nodes* dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Change Apply stage: Link Ordering tab

This tab allows you to specify which input link carries the *before* data set and which carries the *change* data set.

By default the first link added will represent the *before* set. To rearrange the links, choose an input link and click the up arrow button or the down arrow button.

Change Apply stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Change Apply stage uses this when it is determining the sort order for key columns. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Change Apply stage: Input page

The **Input page** allows you to specify details about the incoming data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being compared. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Change Apply stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Change Apply stage: Partitioning tab

The change input to Change Apply should have been output from the Change Capture stage without modification and should have the same number of partitions. Additionally, both inputs of Change Apply are automatically designated as partitioned using the Same partitioning method.

The standard partitioning and collecting controls are available on the Change Apply stage, however, so you can override this behavior.

If the Change Apply stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override the default behavior. The exact operation of this tab depends on:

- Whether the Change Apply stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Change Apply stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Change Apply stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Change Apply stage, and will apply the Same method.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Change Apply stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the operation is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort

occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Change Apply stage: Output page

The Output page allows you to specify details about data output from the Change Apply stage. The Change Apply stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data.The Mapping tab allows you to specify the relationship between the columns being input to the Change Apply stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Change Apply stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Change Apply stage: Mapping tab

For the Change Apply stage the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them or how they are generated.

The left pane shows the common columns of the *before* and *change* data sets. These are read only and cannot be modified on this tab.

The right pane shows the output columns for the output link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility. By default the columns are mapped straight across.

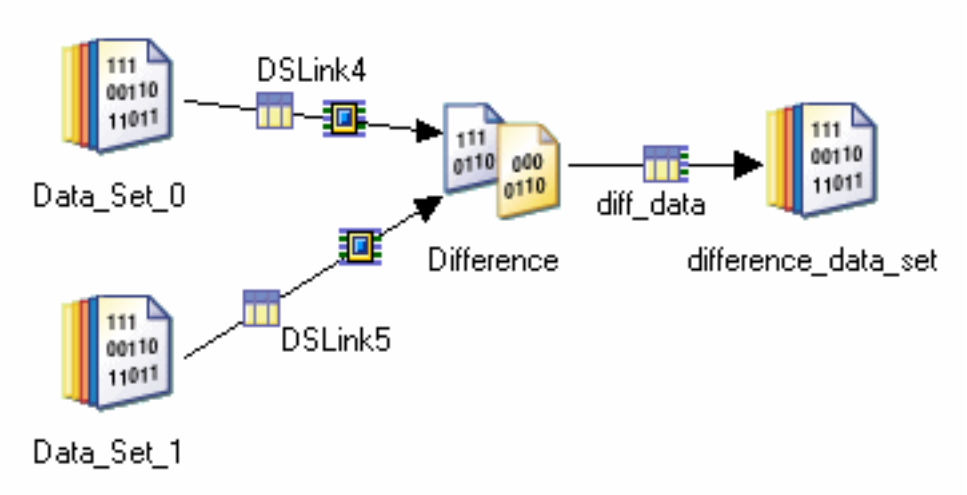
Difference stage

The Difference stage is a processing stage. It performs a record-by-record comparison of two input data sets, which are different versions of the same data set designated the *before* and *after* data sets. The Difference stage outputs a single data set whose records represent the difference between them. The stage assumes that the input data sets have been key-partitioned and sorted in ascending order on the key columns you specify for the Difference stage comparison. You can achieve this by using the Sort stage or by using the built in sorting and partitioning abilities of the Difference stage.

The comparison is performed based on a set of difference key columns. Two records are copies of one another if they have the same value for all difference keys. You can also optionally specify change values. If two records have identical key columns, you can compare the value columns to see if one is an edited copy of the other.

The Difference stage is similar, but not identical, to the Change Capture stage described in "Change Capture stage" on page 339. The Change Capture stage is intended to be used in conjunction with the Change Apply stage ("Change Apply stage" on page 347); it produces a change data set which contains changes that need to be applied to the *before* data set to turn it into the *after* data set. The Difference stage outputs the before and after rows to the output data set, plus a code indicating if there are differences. If the before and after data have the same column names, then one data set effectively overwrites the other data set and so you only see one set of columns in the output. Which data set is output is controlled by the settings on the Link Order tab and the Mapping tab. If your *before* and *after* data sets have different column names, columns from both data sets are available to be output as set on the Mapping tab. Any columns that are designated as key or value columns in the input data sets must have the same names.

The stage generates an extra column, Diff, which indicates the result of each record comparison.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data set having its duplicates removed.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Example data

This example shows a *before* and *after* data set, and the data set that is output by the Difference stage when it has compared them.

This is the *before* data set:

Table 52. Before data set

acol0	acol1	acol2	acol3	key
0	0	0	0	a
1	7	1	1	b
2	2	2	2	c
3	3	3	3	d
4	5	4	4	e
5	2	5	5	f
6	6	6	6	g
7	7	7	7	h

Table 52. Before data set (continued)

acol0	acol1	acol2	acol3	key
8	8	8	8	i
9	9	9	9	j

This is the *after* data set:

Table 53. After data set

acol0	acol1	acol2	acol3	key
0	0	0	0	a
1	1	1	1	b
2	2	2	2	c
3	3	3	3	d
4	4	4	4	e
5	5	5	5	f
6	6	6	6	g
7	7	7	7	h
8	8	8	8	i
9	9	9	9	j

This is the data set output by the Difference stage (Key is the key column, All non-key columns are values are set True, all other settings take the default):

Table 54. Output data set

acol0	acol1	acol2	acol3	key	diff
0	0	0	0	a	2
1	1	1	1	b	3
2	2	2	2	c	2
3	3	3	3	d	2
4	4	4	4	e	3
5	5	5	5	f	3
6	6	6	6	g	2
7	7	7	7	h	2
8	8	8	8	i	2
9	9	9	9	j	2

The diff column indicates that rows b, e, and f have been edited in the after data set (the rows output carry the data after editing).

Difference stage: fast path

This section specifies the minimum steps to take to get a Difference stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Difference stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

1. Go to the Stage page **Properties Tab**.
2. Specify the key column. You can repeat this property to specify a composite key. *Before* and *after* rows are considered to be the same if they have the same value in the key column or columns.
3. Optionally specify one or more Difference Value columns. This enables you to determine if an *after* row is an edited version of a *before* row.

   Note: You can also set the All non-Key columns are Values property to have InfoSphere DataStage treat all columns not defined as keys treated as values.
4. Specify whether the stage will output the changed row or drop it. You can specify this individually for each type of change (copy, delete, edit, or insert).
5. In the Stage page **Link Ordering Tab**, specify which of the two links carries the *before* data set and which carries the *after* data set.
6. If the two incoming data sets aren't already hash partitioned on the key columns and sorted, set InfoSphere DataStage to do this on the **Input Page Partitioning Tab**.
7.

Difference stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The Link Ordering tab allows you to specify which input link carries the *before* data set and which the *after* data set. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Difference stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 55. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Difference Keys/Key	Input Column	N/A	Y	Y	N/A
Difference Keys/Case Sensitive	True/False	True	N	N	Key
Difference Values/All non-Key Columns are Values	True/False	False	Y	N	N/A

Table 55. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Difference Values/Case Sensitive	True/False	True	N	N	All non-Key Columns are Values (when true)
Options/Tolerate Unsorted Inputs	True/False	False	N	N	N/A
Options/Log Statistics	True/False	False	N	N	N/A
Options/Drop Output for Insert	True/False	False	N	N	N/A
Options/Drop Output for Delete	True/False	False	N	N	N/A
Options/Drop Output for Edit	True/False	False	N	N	N/A
Options/Drop Output for Copy	True/False	False	N	N	N/A
Options/Copy Code	number	0	N	N	N/A
Options/Deleted Code	number	2	N	N	N/A
Options/Edit Code	number	3	N	N	N/A
Options/Insert Code	number	1	N	N	N/A

Difference stage: Difference Keys category:
Key

Specifies the name of a difference key input column. This property can be repeated to specify multiple difference key input columns. You can use the Column Selection dialog box to select several keys at once if required. Key has this dependent property:

- **Case Sensitive**

  Use this to property to specify whether each key is case sensitive or not. It is set to True by default; for example, the values "CASE" and "case" would not be judged equivalent.

Difference stage: Difference Values category:
All non-key columns are values

Set this to True to indicate that any columns not designated as difference key columns are value columns. It is False by default. The property has this dependent property:

- **Case Sensitive**

  Use this to property to specify whether each value is case sensitive or not. It is set to True by default; for example, the values "CASE" and "case" would not be judged equivalent. This property is only available if the All non-Key columns are values property is set to True.

Difference stage: Options category:

Tolerate unsorted inputs

Specifies that the input data sets are not sorted. This property allows you to process groups of records that might be arranged by the difference key columns but not sorted. The stage processed the input records in the order in which they appear on its input. It is False by default.

Log statistics

This property configures the stage to display result information containing the number of input records and the number of copy, delete, edit, and insert records. It is False by default.

Drop output for insert

Specifies to drop (not generate) an output record for an insert result. By default, an output record is always created by the stage.

Drop output for delete

Specifies to drop (not generate) the output record for a delete result. By default, an output record is always created by the stage.

Drop output for edit

Specifies to drop (not generate) the output record for an edit result. By default, an output record is always created by the stage.

Drop output for copy

Specifies to drop (not generate) the output record for a copy result. By default, an output record is always created by the stage.

Copy code

Allows you to specify an alternative value for the code that indicates the *after* record is a copy of the *before* record. By default this code is 2.

Deleted code

Allows you to specify an alternative value for the code that indicates that a record in the *before* set has been deleted from the *after* set. By default this code is 1.

Edit code

Allows you to specify an alternative value for the code that indicates the *after* record is an edited version of the *before* record. By default this code is 3.

Insert code

Allows you to specify an alternative value for the code that indicates a new record has been inserted in the *after* set that did not exist in the *before* set. By default this code is 0.

Difference stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the **Advanced** tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Difference stage: Link Ordering tab

This tab allows you to specify which input link carries the *before* data set and which carries the *after* data set.

By default the first link added will represent the before set. To rearrange the links, choose an input link and click the up arrow button or the down arrow button.

Difference stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Difference stage uses this when it is determining the sort order for key columns. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Difference stage: Input page

The Input page allows you to specify details about the incoming data sets. The Difference stage expects two incoming data sets: a *before* data set and an *after* data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being compared. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Difference stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Difference stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the operation is performed. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. For a Difference stage, InfoSphere DataStage checks to see if the incoming data is key-partitioned and sorted. If it is, the Same method is used, if not, InfoSphere DataStage will key partition the data and sort it. You could also explicitly choose hash or modulus partitioning methods and take advantage of the on-stage sorting.

If the Difference stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Difference stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Difference stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Difference stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Difference stage. If the incoming data is already key-partitioned and sorted, InfoSphere DataStage will use the Same method. Otherwise it will key partition and sort for you.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Difference stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available. For the Difference stage, InfoSphere DataStage will ensure that the data is sorted as it is collected.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.

- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the operation is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Difference stage: Output page

The Output page allows you to specify details about data output from the Difference stage. The Difference stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Difference stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Difference stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Difference stage: Mapping tab

For the Difference stage the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them or how they are generated.

The left pane shows the columns from the before/after data sets plus the DiffCode column. These are read only and cannot be modified on this tab.

The right pane shows the output columns for each link. This has a **Derivations** field where you can specify how the column is derived.You can fill it in by dragging input columns over, or by using the Auto-match facility. By default the data set columns are mapped automatically. You need to ensure that there is an output column to carry the change code and that this is mapped to the DiffCode column.

Compare stage

The Compare stage is a processing stage. It can have two input links and a single output link.

The Compare stage performs a column-by-column comparison of records in two presorted input data sets. You can restrict the comparison to specified key columns.

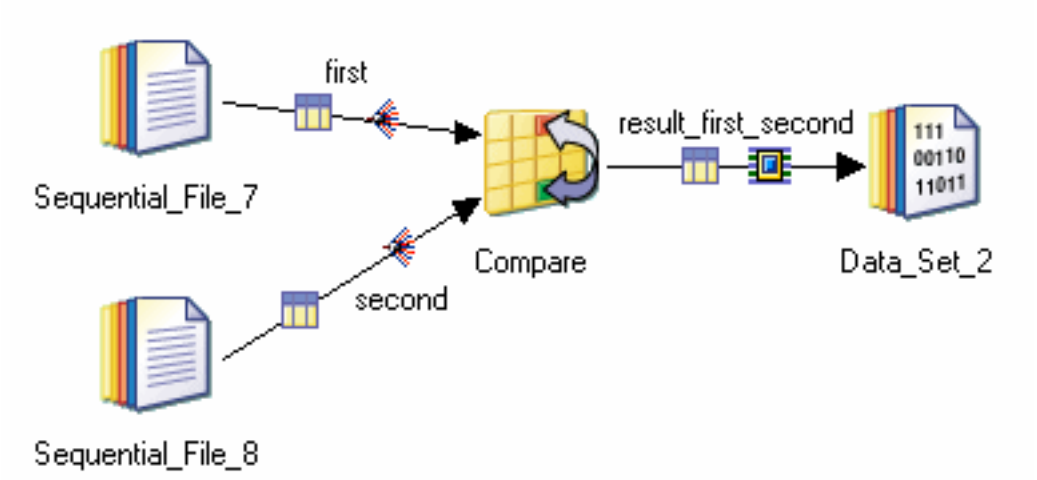
The Compare stage does not change the table definition, partitioning, or content of the records in either input data set. It transfers both data sets intact to a single output data set generated by the stage. The comparison results are also recorded in the output data set.

You can use runtime column propagation in this stage and allow InfoSphere DataStage to define the output column schema for you at runtime. The stage outputs a data set with three columns:

- **result**. Carries the code giving the result of the comparison.
- **first**. A subrecord containing the columns of the first input link.
- **second**. A subrecord containing the columns of the second input link.

If you specify the output link metadata yourself, you must define the columns carrying the data as subrecords of a parent column that you also define. InfoSphere DataStage will not let you specify two groups of identical column names, and so you make them subrecords to give them unique names such as first.col1 and second.col1. Specify metadata by doing the following steps:

1. Specify the parent column for the output data corresponding to the first input link, and set the SQL type to unknown.
2. Specify the actual columns that carry your data and make these subrecords of the parent column. Name each column first.*colname*, for example first.col1, first.col2 and so on. Make each column a subrecord by selecting the column, selecting **edit row** from the shortcut menu, and specifying a level number (for example, 03) for that column. (You can speed up this process by making the first column a subrecord and using the propagate values feature to make the remaining columns subrecords of the parent column.)
3. Specify the parent column for output data corresponding to the second input link, and set the SQL type to unknown.
4. Specify the actual columns that carry the data from the second input link, name them second.*colname* (for example, second.col1, second.col2) and make these subrecords of the parent column.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Example Data

This example shows two data sets being compared, and the data set that is output by the Compare stage when it has compared them.

This is the first data set:

Table 56. First data set

bcol0	bcol1	bcol2	bcol3	bcol4
0	0	0	0	a
1	7	1	1	b
2	2	2	2	c
3	3	3	3	d
4	5	4	4	e
5	2	5	5	f
6	6	6	6	g
7	7	7	7	h
8	8	8	8	i
9	9	9	9	j

This is the second data set:

Table 57. Second data set

bcol0	bcol1	bcol2	bcol3	bcol4
0	0	0	0	a
1	1	1	1	b
2	2	2	2	c
3	3	3	3	d
4	4	4	4	e
5	5	5	5	f
6	6	6	6	g
7	7	7	7	h
8	8	8	8	i
9	9	9	9	j

The stage compares on the Key columns bcol1 and bcol4. This is the output data set:

Table 58. Output data set

Result	First	Second								
	bcol0	bcol1	bcol2	bcol3	bcol4	bcol0	bcol1	bcol2	bcol3	bcol4
0	0	0	0	0	a	0	0	0	0	a
2	1	7	1	1	b	1	1	1	1	b
0	2	2	2	2	c	2	2	2	2	c
0	3	3	3	3	d	3	3	3	3	d
2	4	5	4	4	e	4	4	4	4	e

Table 58. Output data set (continued)

Result	First	Second								
	bcol0	**bcol1**	**bcol2**	**bcol3**	**bcol4**	**bcol0**	**bcol1**	**bcol2**	**bcol3**	**bcol4**
-1	5	2	5	5	f	5	5	5	5	f
0	6	6	6	6	g	6	6	6	6	g
0	7	7	7	7	h	7	7	7	7	h
0	8	8	8	8	i	8	8	8	8	i
0	9	9	9	9	j	9	9	9	9	j

Compare stage: fast path

This section specifies the minimum steps to take to get a Compare stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Compare stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

1. In the Stage page **Properties Tab**, check that the default settings are suitable for your requirements.
2. In the Stage page **Link Ordering Tab**, specify which of your input links is the first link and which is the second.

Compare stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Compare stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 59. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Abort On Difference	True/False	False	Y	N	N/A
Options/Warn on Record Count Mismatch	True/False	False	Y	N	N/A
Options/`Equals' Value	number	0	N	N	N/A

Table 59. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/`First is Empty' Value	number	1	N	N	N/A
Options/`Greater Than' Value	number	2	N	N	N/A
Options/`Less Than' Value	number	-1	N	N	N/A
Options/`Second is Empty' Value	number	-2	N	N	N/A
Options/Key	Input Column	N/A	N	Y	N/A
Options/Case Sensitive	True/False	True	N	N	Key

Compare stage: Options category:
Abort on difference

This property forces the stage to abort its operation each time a difference is encountered between two corresponding columns in any record of the two input data sets. This is False by default, if you set it to True you cannot set Warn on Record Count Mismatch.

Warn on record count mismatch

This property directs the stage 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. This is False by default, if you set it to True you cannot set Abort on difference.

`Equals' value

Allows you to set an alternative value for the code which the stage outputs to indicate two compared records are equal. This is 0 by default.

`First is empty' value

Allows you to set an alternative value for the code which the stage outputs to indicate the first record is empty. This is -2 by default.

`Greater than' value

Allows you to set an alternative value for the code which the stage outputs to indicate the first record is greater than the other. This is 1 by default.

`Less than' value

Allows you to set an alternative value for the code which the stage outputs to indicate the second record is greater than the other. This is -1 by default.

`Second is empty' value

Allows you to set an alternative value for the code which the stage outputs to indicate the second record is empty. This is -2 by default.

Key

Allows you to specify one or more key columns. Only these columns will be compared. Repeat the property to specify multiple columns. You can use the Column Selection dialog box to select several keys at once if required. The Key property has a dependent property:

- **Case Sensitive**

  Use this to specify whether each key is case sensitive or not, this is set to True by default, that is, the values "CASE" and "case" in would end up in different groups.

Compare stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Compare stage: Link Ordering tab

This tab allows you to specify which input link carries the *First* data set and which carries the *Second* data set. Which is categorized as first and which second affects the setting of the comparison code.

By default the first link added will represent the *First* set. To rearrange the links, choose an input link and click the up arrow button or the down arrow button.

Compare stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Compare stage uses this when it is determining the sort order for key columns. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Compare stage: Input page

The **Input page** allows you to specify details about the incoming data sets. The Compare stage expects two incoming data sets.

The **General** tab allows you to specify an optional description of the input link. The **Partitioning** tab allows you to specify how incoming data is partitioned before being compared. The **Columns** tab specifies the column definitions of incoming data. The **Advanced** tab allows you to change the default buffering settings for the input link.

Details about Compare stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Compare stage: Partitioning tab

If you are running the Compare stage in parallel you must ensure that the incoming data is suitably partitioned and sorted to make a comparison sensible.

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is compared. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Compare stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Compare stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Compare stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Compare stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Compare stage, and will ensure that incoming data is key partitioned and sorted.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.

- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Compare stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available. For the Compare stage, InfoSphere DataStage will ensure that the data is sorted as it is collected.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

If you are collecting data, the Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being collected and compared. The sort is always carried out within data partitions. The sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Compare stage: Output page

The Output page allows you to specify details about data output from the Compare stage. The Compare stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

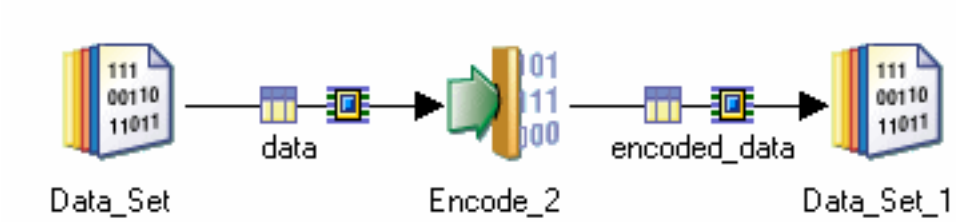
See "Stage Editors," for a general description of the tabs.

Encode Stage

The Encode stage is a processing stage. It encodes a data set using a UNIX encoding command, such as gzip, that you supply. The stage converts a data set from a sequence of records into a stream of raw binary data. The companion Decode stage reconverts the data stream to a data set (see “Decode stage” on page 374).

An encoded data set is similar to an ordinary one, and can be written to a data set stage. You cannot use an encoded data set as an input to stages that performs column-based processing or re-orders rows, but you can input it to stages such as Copy. You can view information about the data set in the data set viewer, but not the data itself. You cannot repartition an encoded data set, and you will be warned at runtime if your job attempts to do that.

As the output is always a single stream, you do not have to define meta data for the output link



.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Encode stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Encode stages in a job. This section specifies the minimum steps to take to get an Encode stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use an Encode stage:

- In the Stage Page **Properties Tab**, specify the UNIX command that will be used to encode the data, together with any required arguments. The command should expect its input from STDIN and send its output to STDOUT.

Encode stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Encode stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. This stage only has one property and you must supply a value for this. The property appears in the warning color (red by default) until you supply a value.

Table 60. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Command Line	Command Line	N/A	Y	N	N/A

Encode stage: Options category:
Command line

Specifies the command line used for encoding the data set. The command line must configure the UNIX command to accept input from standard input and write its results to standard output. The command must be located in your search path and be accessible by every processing node on which the Encode stage executes.

Encode stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Set** by default to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Encode stage: Input page

The Input page allows you to specify details about the incoming data sets. The Encode stage can only have one input link.

The General tab allows you to specify an optional description of the input link. The *Partitioning* tab allows you to specify how incoming data is partitioned before being encoded. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Encode stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Encode stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is encoded. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Encode stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Encode stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Encode stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Encode stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Encode stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Encode stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being encoded. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.

- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Encode stage: Output page

The Output page allows you to specify details about data output from the Encode stage. The Encode stage can have only one output link.

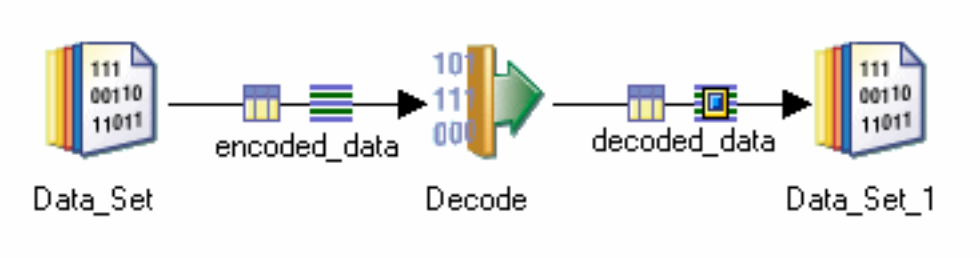
The General tab allows you to specify an optional description of the output link. The Columns tab allows you to specify column definitions for the data (although this is optional for an encode stage). The Advanced tab allows you to change the default buffering settings for the output link.

See "Stage Editors," for a general description of these tabs.

Decode stage

The Decode stage is a processing stage. It decodes a data set using a UNIX decoding command, such as gzip, that you supply. It converts a data stream of raw binary data into a data set. Its companion stage, Encode, converts a data set from a sequence of records to a stream of raw binary data (see “Encode Stage” on page 370).

As the input is always a single stream, you do not have to define meta data for the input link.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Decode stage: fast path

This section specifies the minimum steps to take to get a Decode stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Decode stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

In the Stage Page **Properties Tab**, specify the UNIX command that will be used to decode the data, together with any required arguments. The command should expect its input from STDIN and send its output to STDOUT.

Decode stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Decode stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. This stage only has one property and you must supply a value for this. The property appears in the warning color (red by default) until you supply a value.

Table 61. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Command Line	Command Line	N/A	Y	N	N/A

Decode stage: Options category: Command line

Specifies the command line used for decoding the data set. The command line must configure the UNIX command to accept input from standard input and write its results to standard output. The command must be located in the search path of your application and be accessible by every processing node on which the Decode stage executes.

Decode stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Decode stage: Input page

The Input page allows you to specify details about the incoming data sets. The Decode stage expects a single incoming data set.

The General tab allows you to specify an optional description of the input link. The *Partitioning* tab allows you to specify how incoming data is partitioned before being decoded. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Decode stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Decode stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is decoded. It also allows you to specify that the data should be sorted before being operated on.

The Decode stage partitions in Same mode and this cannot be overridden.

If the Decode stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Decode stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

Decode stage: Output page

The Output page allows you to specify details about data output from the Decode stage. The Decode stage can have only one output link.

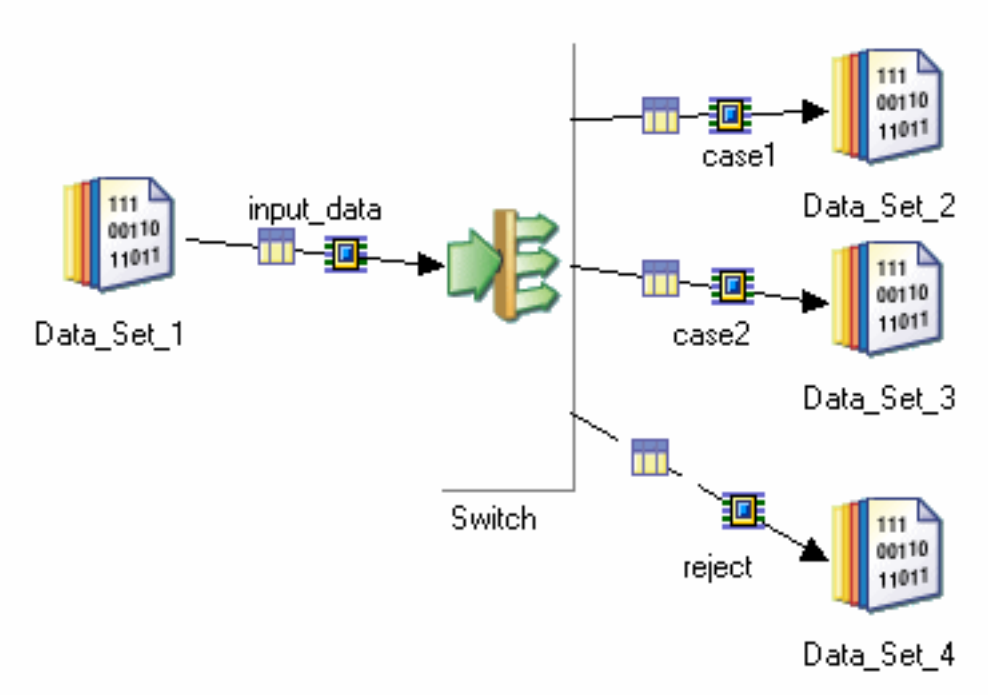
The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions for the decoded data.

See "Stage Editors," for a general description of the tabs.

Switch stage

The Switch stage is a processing stage. It can have a single input link, up to 128 output links and a single rejects link.

The Switch stage takes a single data set as input and assigns each input row to an output data set based on the value of a selector field. The Switch stage performs an operation 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. Rows that satisfy none of the cases are output on the rejects link.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting rows.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Switch stage example

The example Switch stage implements the following switch statement:

```
switch (selector)
{
   case 0:          // if selector = 0,
            // write record to output data set 0
            break;
   case 10:          // if selector = 10,
            // write record to output data set 1
            break;
   case 12:             // if selector = discard value (12)
               // skip record
               break;
   case default:             // if selector is invalid,
               // send row down reject link
};
```

The metadata input to the switch stage listed in the following table.

Table 62. Column definitions

Column name	SQL type
Select	Integer
col1	Char
col2	Char

Table 62. Column definitions (continued)

Column name	SQL type
col3	Char

In this example, the Select column is the selector. Each input row is assigned to an output data set based on the value of the selector.

To implement the switch statement, the properties of the stage are set to the following values.

Table 63. Example Switch stage properties

Property	Value
Selector	Select
Selector Mode	User-defined Mapping
Case	0=0
Case	10=1
Case	12=5
Discard Value	5

In this example, any rows that have a 0 in the Select column are sent to output link 0. Any rows that have a 10 in the Select column are sent to output link 1, and any rows that have a 12 in the Select column are discarded.

Switch stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Switch stages in a job. This section specifies the minimum steps to take to get a Switch stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Switch stage:

- In the Stage page **Properties Tab**, under the Input category choose the Selector mode:
  - **User-defined Mapping**. This is the default, and means that you must provide explicit mappings from case values to outputs. If you use this mode you specify the switch expression under the User-defined Mapping category.
  - **Auto**. This can be used where there are as many distinct selector values as there are output links.
  - **Hash**. The incoming rows are hashed on the selector column modulo the number of output links and assigned to an output link accordingly.

    In all cases you need to use the **Selector** property to specify the input column that the switch is performed on. You can also specify whether the column is case sensitive or not. The other properties depend on which mode you have chosen:
  - If you have chosen the User-defined mapping mode, under the User-defined Mapping category specify the case expression in the case property. Under the Option category, select the If not found property to specify what action the stage takes if the column value does not correspond to any of the cases. Choose from Fail to have the job fail, Drop to drop the row, or Output to output it on the reject link.

- If you have chosen the Auto mode, Under the Option category, select the If not found property to specify what action the stage takes if the column value does not correspond to any of the cases. Choose from Fail to have the job fail, Drop to drop the row, or Output to output it on the reject link.
- If you have chose the Hash mode there are no other properties to fill in.

- In the Stage page **Link Ordering Tab**, specify the order in which the output links are processed.
- In the Output page **Mapping Tab** check that the input columns are mapping onto the output columns as you expect. The mapping is carried out according to what you specified in the Properties tab.

Switch stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The Link Ordering tab allows you to specify what order the output links are processed in. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Switch stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 64. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Input/Selector	Input Column	N/A	Y	N	N/A
Input/Case Sensitive	True/False	True	N	N	Selector
Input/Selector Mode	User-defined mapping/Auto/ Hash	User-defined mapping	Y	N	N/A
User-defined Mapping/Case	String	N/A	Y (if Selector Mode = User-defined mapping)	Y	N/A
Options/If not found	Pathname	N/A	Y (if Column Method = Schema file)	N	N/A
Options/Discard Value	String	N/A	N	N	N/A

Switch stage: Input category: Selector

Specifies the input column that the switch applies to.

Case sensitive

Specifies whether the column is case sensitive or not.

Selector mode

Specifies how you are going to define the case statements for the switch. Choose between:

- **User-defined Mapping**. This is the default, and means that you must provide explicit mappings from case values to outputs. If you use this mode you specify the switch expression under the User-defined Mapping category.
- **Auto**. This can be used where there are as many distinct selector values as there are output links.
- **Hash**. The incoming rows are hashed on the selector column modulo the number of output links and assigned to an output link accordingly.

Switch stage: User-defined mapping category:
Case

This property appears if you have chosen a Selector Mode of User-defined Mapping. Specify the case expression in the case property. It has the following format:

Selector_Value[= *Output_Link_Label_Number*]

You must specify a selector value for each value of the input column that you want to direct to an output column. Repeat the Case property to specify multiple values. You can omit the output link label if the value is intended for the same output link as the case previously specified. For example, the case statements:

```
1990=0
1991
1992
1993=1
1994=1
```

would cause the rows containing the dates 1990, 1991, or 1992 in the selector column to be routed to output link 0, and the rows containing the dates 1993 to 1994 to be routed to output link 1.

Switch stage: Options category:
If not found

Specifies the action to take if a row fails to match any of the case statements. This does not appear if you choose a Selector Mode of Hash. Otherwise, choose between:

- **Fail**. Causes the job to fail.
- **Drop**. Drops the row.
- **Output**. Routes the row to the Reject link.

Discard value

You can use this property in conjunction with the case property to specify that rows containing certain values in the selector column will always be discarded. For example, if you defined the following case statement:

```
1995=5
```

and set the Discard Value property to 5, all rows containing 1995 in the selector column would be routed to link 5 which has been specified as the discard link and so will be dropped.

Switch stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Switch stage: Link Ordering tab

This tab allows you to specify which output links are associated with which link labels.

Switch stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Switch stage uses this when evaluating case statements. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Switch stage: Input page

The Input page allows you to specify details about the incoming data sets. The Switch stage expects one incoming data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being switched. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Switch stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Switch stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is switched. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Column Import stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Switch stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Switch stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Switch stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Switch stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Switch stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being imported. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Switch stage: Output page

The Output page allows you to specify details about data output from the Switch stage. The Switch stage can have up to 128 output links, and can also have a reject link carrying rows that have been rejected. Choose the link you are working on from the **Output name** drop-down list.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Switch stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output links.

Details about Switch stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Switch stage: Mapping tab

For the Switch stage the Mapping tab allows you to specify how the output columns are derived.

The left pane shows the columns that have been switched. These are read only and cannot be modified on this tab.

The right pane shows the output columns for each link.

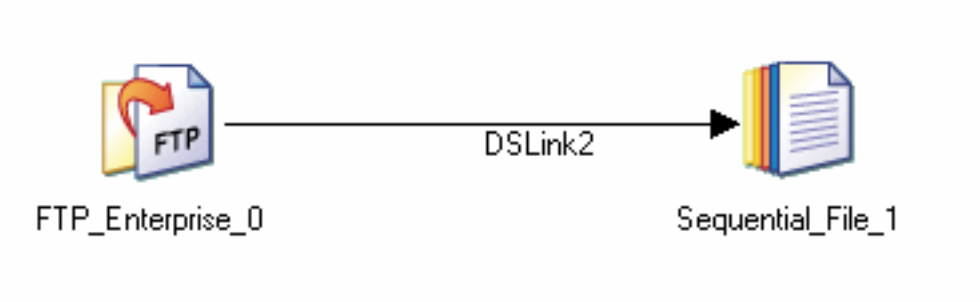
Switch stage: Reject link

You cannot change the details of a Reject link. The link uses the column definitions for the link rejecting the data rows.

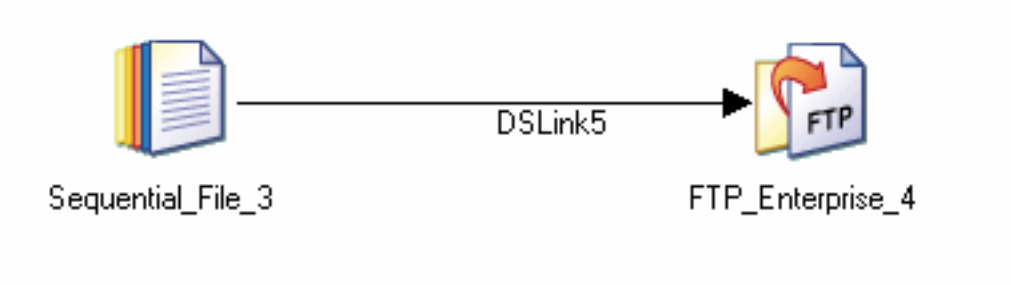
FTP Enterprise Stage

The FTP Enterprise stage transfers multiple files in parallel. These are sets of files that are transferred from one or more FTP servers into InfoSphere DataStage or from InfoSphere DataStage to one or more FTP servers. The source or target for the file is identified by a URI (Universal Resource Identifier). The FTP Enterprise stage invokes an FTP client program and transfers files to or from a remote host using the FTP Protocol.

Accessing Files from a Remote Host:



Transferring Files from a Local Sequential file to a Remote Host:



When you edit an FTP Enterprise stage, the FTP stage editor appears. This is based on the generic stage editor.

The stage editor has up to three pages, depending on whether you are accessing or transferring a file:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is present when you are accessing files from a remote host. Specify details about the input link here.
- **Output Page**. This is present when you are transferring data sets to a file. Specify details about the output link here.

Restart capability

You can specify that the FTP operation runs in restartable mode. To do this you set the stage properties as follows:

1. Specify a restartable mode of restartable transfer.
2. Specify a unique job id for the transfer
3. Optionally specify a checkpoint directory for the transfer 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 restartable mode set to restart transfer or abandon transfer. 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 an FTP stage with the restart transfer option to attempt the transfer again using the information in the restart directory. .

For get operations, InfoSphere 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 dataset 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.

FTP Enterprise stage: fast path

InfoSphere DataStage has many defaults, which means that it is easy to include FTP Enterprise stages in a job. This section specifies the minimum steps required to get an FTP stage functioning. InfoSphere DataStage provides a versatile user interface, with many shortcuts available to achieve a particular end. This section describes the basic method. You will learn where the shortcuts are when you get familiar with the product.

The steps required depend on whether you are using the FTP Enterprise stage to access or transfer a file.

Transferring Data to a Remote Host Using the FTP Enterprise Stage

About this task

In the **Input** page**Properties** tab:

Procedure

1. Specify the URI that the file will be put to. You can specify multiple URIs if required. You can specify an absolute or relative pathname, see "FTP Enterprise stage: Properties Tab" on page 386 for details of syntax.
2. Ensure column meta data has been specified for the files. (You can achieve this via a schema file if required.)

Accessing Data from a Remote Host Using the FTP Enterprise Stage

You can access Data from a Remote Host Using the FTP Enterprise Stage.

Procedure

1. Go to the **Properties** tab on the Output page.
2. Specify the URI that the file will be taken from via a get operation. You can specify multiple URIs if required. You can also use a wildcard character to retrieve multiple files. You can specify an absolute or relative pathname, see "FTP Enterprise stage: Properties Tab" on page 391 for details of syntax.
3. Ensure column meta data has been specified for the files. (You can achieve this via a schema file if required.)

FTP Enterprise stage: Stage Page

The **General** tab allows you to specify an optional description of the stage. The **Properties** tab lets you specify what the stage does. The **Advanced** tab allows you to specify how the stage executes. Use the **NLS Map** (National Language Support) tab to define a character set map for the FTP Enterprise stage.

FTP Enterprise stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the **Advanced** tab. In Sequential mode the entire data set is processed by the conductor node.

- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. The stage has a mandatory partitioning method of **Same**, this overrides the preserve partitioning flag and so the partitioning of the incoming data is always preserved.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the **Available Nodes** dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

FTP Enterprise stage: NLS Map Tab

Use the **NLS Map** (National Language Support) tab to define a character set map for the FTP Enterprise stage. This overrides the default character set map for the project or the job. You can specify that the map be supplied as a job parameter if required. You can also select Allow per-column mapping. You can specify character set maps for individual columns within the data processed by the FTP Enterprise stage. An extra property, NLS Map, appears in the Columns grid, but note that only ustring data types allow you to set an NLS Map value.

FTP Enterprise stage: Input Page

The **Input page** allows you to specify details about how the stage transfers files to a remote host using FTP. The FTP Enterprise stage can have only one input link, but this can write to multiple files.

The **General** tab allows you to specify an optional description of the input link. The **Properties** tab allows you to specify details of exactly what the link does. The **Partitioning** tab allows you to specify how incoming data is partitioned before being written to the file or files. The **Columns** tab specifies the column definitions of data being written. The **Advanced** tab allows you to change the default buffering settings for the input link.

Details about FTP Enterprise stage properties and partitioning are given in the following sections. See Chapter 4, “Stage editors,” on page 49, "Stage Editors," for a general description of the other tabs.

FTP Enterprise stage: Properties Tab

Use the **Properties tab** to specify properties, which determine what the stage actually does. The available properties are displayed in a tree structure. They are divided into categories to help you find your way around them. All the mandatory properties are included in the tree by default and cannot be removed. Required properties (that is, which do not have a default value) are shown in a warning color (red by default) but change to black when you have set a value.

The following table gives a quick reference list of the properties and their attributes.

Table 65. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Target/ URI	Pathname	N/A	Y	Y	N/A
Target/URI/ Open command	String	N/A	N	N	URI

Table 65. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Connection /ftp command	Pathname	ftp	N	N	N/A
Connection/ Password	Encrypted	N/A	N	Y	N/A
Connection/ User Name	String	N/A	N	Y	N/A
Options/ Force Parallelism	Yes/No	No	N	N	N/A
Options/ Overwrite	Yes/No	No	N	N	N/A
Options/ Restartable mode	Abandon Transfer/ Restart Transfer/ Restartable Transfer	Restartable Transfer	N	Y	N/A
Options/ Restartable Mode/ Checkpointdir	Pathname	N/A	N	N	Restartable Mode
Options/ Restartable Mode/ Job Id	Integer	1	N	N	Restartable Mode
Options/ Schema file	Pathname	N/A	N	N	N/A
Options/ Transfer Type	Binary/ASCII	Binary	N	N	N/A
Transfer Protocol/ Transfer Mode	FTP/Secure FTP (SFTP)	FTP	N	N	N/A

Target Category:

Specify URI and, if required, Open command.

URI

Is a pathname connecting the Stage to a target file on a remote host. It has the Open dependent property. You can repeat this property to specify multiple URIs. You can specify an absolute or a relative pathname.

The syntax for a relative path is:
`ftp://host/path/filename`

Where *path* is the relative path of the user's home directory.

The syntax for an absolute path is:
`ftp://host//path/filename`

While connecting to the mainframe system, the syntax for an absolute path is:
`ftp://host/\'path.filename\'`

The syntax for connecting to the mainframe system through USS (Unix System Services) is:
`ftp://host//path/filename`

Where *path* is the absolute path of the user's home directory.

Open command

Is required if you perform any operation besides navigating to the directory where the file exists. There can be multiple Open commands. This is a dependent property of URI.

Connection Category:

Specify the **ftp Command, Password** and **User Name**. Only one ftp command can be specified, but multiple passwords and user names can be specified.

ftp command

Is an optional command that you can specify if you do not want to use the default ftp command. For example, you could specify /opt/gnu/bin/wuftp. You can enter the path of the command (on the engine tier host) directly in this field. You can also specify a job parameter if you want to be able to specify the ftp command at run time.

User Name

Specify the user name for the transfer. You can enter it directly in this field, or you can specify a job parameter if you want to be able to specify the user name at run time. You can specify multiple user names. User1 corresponds to URI1 and so on. When the number of users is less than the number of URIs, the last user name is set for remaining URIs

If no User Name is specified, the FTP Enterprise Stage tries to use .netrc file in the home directory.

Password

Enter the password in this field. You can also specify a job parameter if you want to be able to specify the password at run time. Specify a password for each user name. Password1 corresponds to URI1. When the number of passwords is less than the numbers of URIs, the last password is set for the remaining URIs.

Transfer Protocol

Select the type of FTP service to transfer files between computers. You can choose either FTP or Secure FTP (SFTP).

FTP

Select this option if you want to transfer files using the standard FTP protocol. This is a nonsecure protocol. By default FTP enterprise stage uses this protocol to transfer files.

Secure FTP (SFTP)

Select this option if you want to transfer files between computers in a secured channel. Secure FTP (SFTP) uses the SSH (Secured Shell) protected channel for data transfer between computers over a nonsecure network such as a TCP/IP network. Before you can use SFTP to transfer files, you should configure the

SSH connection without any pass phrase for RSA authentication.

Options Category:
Force Parallelism

You can set either Yes or No. In general, the FTP Enterprise stage tries to start as many processes as needed to transfer the *n* files in parallel. However, you can force the parallel transfer of data by specifying this property to yes. This allows *m* number of processes at a time where *m* is the number specified in InfoSphere DataStage configuration file. If *m* is less than *n*, then the stage waits to transfer the first *m* files and then start the next *m* until *n* files are transferred.

When you set **Force Parallelism** to **Yes**, you should only give one URI.

Overwrite

Set this option to have any existing files overwritten by this transfer.

Restartable Mode

When you specify a restartable mode of Restartable transfer, InfoSphere DataStage creates a directory for recording information about the transfer in a restart directory. If the transfer fails, you can run an identical job with the restartable mode property set to Restart transfer, which will reattempt the transfer. If the transfer repeatedly fails, you can run an identical job with the restartable mode option set to Abandon transfer, which will delete the restart directory. For more details, see "Restartability" .
Restartable mode has the following dependent properties:

- **Job Id**
  Identifies a restartable transfer job. This is used to name the restart directory.
- **Checkpoint directory**
  Optionally specifies a checkpoint directory to contain restart directories. If you do not specify this, the current working directory is used.

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*.

Schema file

Contains a schema for storing data. Setting this option overrides any settings on the **Columns** tab. You can enter the path name of a schema file, or specify a job parameter, so the schema file name can be specified at run time.

Transfer Type

Select a data transfer type to transfer files between computers. You can select either the Binary or ASCII mode of data transfer. The default data transfer mode is binary.

FTP Enterprise stage: Partitioning Tab

Use the **Partitioning** tab to specify details about how the incoming data is partitioned or collected before it is operated on. Use it also to specify whether the data should be sorted before being operated on. By default, most stages partition in Auto mode. This mode attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. If the stage is operating in sequential mode, it will first collect the data before writing it to the file using the default Auto collection method.

Use the **Partitioning** tab to override this default behavior. The exact operation of this tab depends on whether:

- The stage is set to execute in parallel or sequential mode.
- The preceding stage in the job is set to execute in parallel or sequential mode.

If the FTP Enterprise stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the FTP Enterprise stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the FTP Enterprise stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the FTP Enterprise stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The **Partitioning** tab also allows you to specify that data arriving on the input link should be sorted before being written to the file or files. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.

- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

- You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

FTP Enterprise stage: Output Page

The **Output page** allows you to specify details about how the FTP Enterprise stage transfers one or more files from a remote host using the FTP protocol. The FTP Enterprise stage can have only one output link, but this can transfer multiple files.

The **General** tab allows you to specify an optional description of the output link. The **Properties** tab allows you to specify details of exactly what the link does. The **Columns** tab specifies the column definitions of the data. The **Advanced** tab allows you to change the default buffering settings for the output link.

Details about FTP Enterprise stage properties and formatting are given in the following sections. See "Stage Editors," for a general description of the other tabs.

FTP Enterprise stage: Properties Tab

Use the **Properties tab** to specify properties, which determine what the stage actually does. The available properties are displayed in a tree structure. They are divided into categories to help you find your way around them. All the mandatory properties are included in the tree by default and cannot be removed. Required properties (that is, which do not have a default value) are shown in a warning color (red by default) but change to black when you have set a value.

The following table gives a quick reference list of the properties and their attributes.

Table 66. Properties

Category/Property	Values	Default	Mandatory?	Repeats?	Dependent of
Source/URI	Pathname	N/A	Y	Y	N/A
Source/URI/Open Command	String	N/A	N	N	URI
Connection/ftp command	Pathname	ftp	N	N	N/A
Connection/ Password	Encrypted	N/A	N	Y	N/A
Connection/ User Name	String	N/A	N	Y	N/A
Options/Force Parallelism	Yes/No	No	N	N	N/A
Options/Restartable Mode	Abandon Transfer/Restart Transfer/ Restartable Transfer	Restartable Transfer	N	Y	N/A
Options/Restartable Mode/ Checkpointdir	Pathname	N/A	N	N	Restartable Mode

Table 66. Properties (continued)

Category/Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Restartable Mode/ Job Id	Integer	1	N	N	Restartable Mode
Options/Schema file	Pathname	N/A	N	N	N/A
Options/Transfer Type	Binary/ASCII	Binary	N	N	N/A
Transfer ProtocolTransfer Mode	FTP/Secure FTP (SFTP)	FTP	N	N	N/A

Source Category:

Specify URI and, if required, Open command.

URI

Is a pathname connecting the Stage to a source file on a remote host. It has the Open dependent property. You can repeat this property to specify multiple URIs, or you can use a wildcard in the path to retrieve a number of files. You can specify an absolute or a relative pathname.

The syntax for a relative path is:
`ftp://host/path/filename`

Where *path* is the relative path of the user's home directory.

The syntax for an absolute path is:
`ftp://host//path/filename`

While connecting to the main frame system, the syntax for an absolute path is:
`ftp://host/\'path.filename\'`

The syntax for connecting to the main frame system through USS is:
`ftp://host//path/filename`

Where *path* is the absolute path of the user's home directory.

Open command

Is required if you perform any operation besides navigating to the directory where the file exists. There can be multiple Open commands. This is a dependent property of **URI**.

Connection Category:

Specify the **ftp Command, Password** and **User Name**. Only one ftp command can be specified, but multiple passwords and user names can be specified.

ftp command

Is an optional command that you can specify if you do not want to use the default ftp command. For example, you could specify /opt/gnu/bin/wuftp. You can enter the path of the command (on the engine tier host) directly in this field. You can also specify a job parameter if you want to be able to specify the ftp command at run time.

User Name

Specify the user name for the transfer. You can enter it directly in this field, or you can specify a job parameter if you want to be able to specify the user name at run time. You can specify multiple user names. User1 corresponds to URI1 and so on. When the number of users is less than the number of URIs, the last user name is set for remaining URIs

If no User Name is specified, the FTP Enterprise Stage tries to use .netrc file in the home directory.

Password

Enter the password in this field. You can also specify a job parameter if you want to be able to specify the password at run time. Specify a password for each user name. Password1 corresponds to URI1. When the number of passwords is less than the numbers of URIs, the last password is set for the remaining URIs.

Transfer Protocol

Select the type of FTP service to transfer files between computers. You can choose either FTP or Secure FTP (SFTP).

FTP

Select this option if you want to transfer files using the standard FTP protocol. This is a nonsecure protocol. By default FTP Enterprise Stage uses this protocol to transfer files.

Secure FTP (SFTP)

Select this option if you want to transfer the files between computers in a secured channel. Secure FTP (SFTP) uses the SSH (Secured Shell) protected channel for data transfer between computers over a non-secure network such as a TCP/IP network. Before you can use SFTP to transfer files, you should configure the SSH connection without any pass phrase for RSA authentication.

Options Category:
Force Parallelism

You can set either Yes or No. In general, the FTP Enterprise stage tries to start as many processes as needed to transfer the *n* files in parallel. However, you can force the parallel transfer of data by specifying this property to yes. This allows *m* number of processes at a time where *m* is the number specified in InfoSphere DataStage configuration file. If *m* is less than *n*, then the stage waits to transfer the first *m* files and then start the next *m* until *n* files are transferred.

When you set **Force Parallelism** to **Yes**, you should only give one URI.

Restartable Mode

When you specify a restartable mode of Restartable transfer, InfoSphere DataStage creates a directory for recording information about the transfer in a restart directory. If the transfer fails, you can run an identical job with the restartable mode property set to Restart transfer, which will reattempt the transfer. If the transfer repeatedly fails, you can run an identical job with the restartable mode option set to Abandon transfer, which will delete the restart directory. For more details, see “Restart capability” on page 384.

For get operations, InfoSphere 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 dataset from the temporary folder path.

Restartable mode has the following dependent properties:

- **Job Id**

  Identifies a restartable transfer job. This is used to name the restart directory.
- **Checkpoint directory**

  Optionally specifies a checkpoint directory to contain restart directories. If you do not specify this, the current working directory is used.

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*.

Schema file

Contains a schema for storing data. Setting this option overrides any settings on the **Columns** tab. You can enter the path name of a schema file, or specify a job parameter, so the schema file name can be specified at run time.

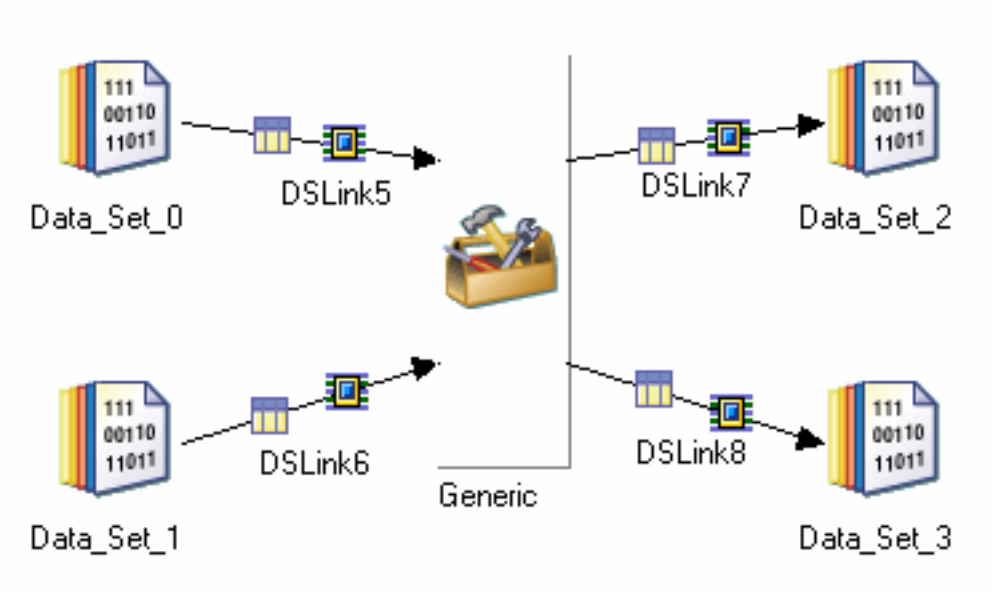
Transfer Type

Select a data transfer type to transfer files between computers. You can select either the Binary or ASCII mode of data transfer. The default data transfer type is binary.

Generic stage

The Generic stage is a processing stage. It has any number of input links and any number of output links.

The Generic stage allows you to call an Orchestrate operator from within a InfoSphere DataStage stage and pass it options as required. See *InfoSphere DataStage Parallel Job Advanced Developer Guide* for more details about Orchestrate operators.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Generic stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Generic stages in a job. This section specifies the minimum steps to take to get a Generic stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Generic stage:

- In the Stage page **Properties Tab**:
  - Specify the name of the Orchestrate operator the stage will call.
  - Specify the name of any options the operator requires, and set its value. This can be repeated to specify multiple options.
- In the Stage page **Link Ordering Tab**, order your input and output links so they correspond to the required link number.

Generic stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Generic stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 67. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Operator	Orchestrate operator	N/A	Y	N	N/A
Options/Option name	String	N/A	N	Y	N/A
Options/Option Value	String	N/A	N	N	Option name

Generic stage: Options category: Operator

Specify the name of the Orchestrate operator the stage will call.

Option name

Specify the name of an option the operator requires. This has a dependent property:

- **Option Value**
  The value the option is to be set to.

Generic stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Generic stage: Link Ordering tab

This tab allows you to specify how input and output links correspond to link labels.

To rearrange the links, choose an output link and click the up arrow button or the down arrow button.

Generic stage: Input page

The Input page allows you to specify details about the incoming data sets. The Generic stage can accept multiple incoming data sets. Select the link whose details you are looking at from the **Input name** drop-down list.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being operated on. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Generic stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Generic stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is operated on. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Generic stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Generic stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Generic stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Generic stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method of the Generic stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Generic stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being operated on. The sort is always carried out within data partitions. If the stage is partitioning

incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Generic stage: Output page

The Output page allows you to specify details about data output from the Generic stage. The Generic stage can have any number of output links. Select the link whose details you are looking at from the **Output name** drop-down list.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output links.

See "Stage Editors," for a general description of these tabs.

Surrogate Key Generator stage

The Surrogate Key Generator stage is a processing stage that generates surrogate key columns and maintains the key source.

A surrogate key is a unique primary key that is not derived from the data that it represents, therefore changes to the data will not change the primary key. In a star schema database, surrogate keys are used to join a fact table to a dimension table.

The Surrogate Key Generator stage can have a single input link, a single output link, both an input link and an output link, or no links. Job design depends on the purpose of the stage.

You can use a Surrogate Key Generator stage to perform the following tasks:

- Create or delete the key source before other jobs run
- Update a state file with a range of key values
- Generate surrogate key columns and pass them to the next stage in the job
- View the contents of the state file

Generated keys are unsigned 64-bit integers. The key source can be a state file or a database sequence. If you are using a database sequence, the sequence must be created by the Surrogate Key stage. You cannot use a sequence previously created outside of DataStage.

You can use the Surrogate Key Generator stage to update a state file, but not a database sequence. Sequences must be modified with database tools.

Creating the key source

You can create the surrogate key source by adding a Surrogate Key Generator stage to a job by itself, with no input or output links.

About this task

You must create a key source before you can use it in a job. The key source can be a state file or a database sequence. If you are using a database sequence, the sequence must be created by the Surrogate Key stage. You cannot use a sequence previously created outside of DataStage.

Procedure

1. Open the Surrogate Key Generator stage editor.
2. On the Properties tab, define the Key Source properties:
   a. Set the **Key Source Action** property to `Create`.
   b. Type or browse for the source name.
   c. Select the source type.
3. If the source type is a database sequence, complete some additional steps on the Properties tab:
   a. Define the Database Type properties.
   b. Add the **Sequence Initial Value** property to the Options group, and specify the value to initialize the sequence.
   c. If you want to control the block size for key ranges, add the **Sequence Block Size** property to the Options group, set this property to `User specified`, and specify a value for the block size.
4. Optional: On the Advanced tab, change the processing settings for the stage.
5. Click **OK** to save your changes and to close the Surrogate Key Generator stage editor.

Deleting the key source

To delete the surrogate key source, design a job that uses a Surrogate Key Generator stage by itself, with no input or output links.

Procedure

1. Open the Surrogate Key Generator stage editor.
2. On the Properties tab, define the Key Source properties:
   a. Set the **Key Source Action** property to `Delete`.
   b. Type or browse for the source name.
   c. Select the source type.
   d. If the source type is a database sequence, define the Database Type properties.
3. Optional: On the Advanced tab, change the processing settings for the stage.
4. Click **OK** to save your changes and to close the Surrogate Key Generator stage editor.

Updating the state file

To update the state file, add a Surrogate Key Generator stage to a job with a single input link from another stage. If the state file does not exist, you can optionally create it in the same job.

Before you begin

Before you edit the Surrogate Key Generator stage, you must edit the stage on the input link. In the data input stage, specify details about the data source, such as the table name and connection properties, and define the column metadata.

About this task

You cannot update database sequences. Use database tools to update sequences.

Procedure

1. Open the Surrogate Key Generator stage editor.
2. On the Stage page, define the stage properties:
   a. Click the **Properties** tab.
   b. Select the input column to update the state file. The input column usually is the surrogate key column.
   c. Set the **Key Source Update Action** property to `Update`, or if the state file does not exist and you want to create it, select `Create and Update`.
   d. Type or browse for the source name.
   e. Set the **Source Type** property to `Flat File`.
   f. Optional: On the Advanced tab, change the processing settings for the stage.
3. Optional: On the Input page, define the input data to the stage:
   a. On the Partitioning tab, change the partition settings for the input link.
   b. On the Advanced tab, change the buffer settings for the input link.
4. Click **OK** to save your changes and to close the Surrogate Key Generator stage editor.

Generating surrogate keys

To generate surrogate keys, add a Surrogate Key Generator stage to a job with a single output link to another stage.

About this task

If you want to pass input columns to the next stage in the job, the Surrogate Key Generator stage can also have an input link.

Procedure

1. Open the Surrogate Key Generator stage editor.
2. On the Stage page, define the stage properties:
   a. Click the **Properties** tab.
   b. Type a name for the surrogate key column in the **Generated Output Column Name** property.
   c. Type or browse for the source name.
   d. Select the source type.
   e. If the source type is a database sequence, define the Database Type properties.
   f. If the key source is a flat file, specify how keys are generated:
      - To generate keys in sequence from the highest value that was last used, set the **Generate Key from Last Highest Value** property to `Yes`. Any gaps in the key range are ignored.
      - To specify a value to initialize the key source, add the **File Initial Value** property to the Options group, and specify the start value for key generation.
      - To control the block size for key ranges, add the **File Block Size** property to the Options group, set this property to `User specified`, and specify a value for the block size.
   g. If there is no input link, add the **Number of Records** property to the Options group, and specify how many records to generate.
   h. Optional: On the Advanced tab, change the processing settings for the stage.
3. Optional: If the stage has an input link, on the Input page, define the input data:
   a. On the Partitioning tab, change the partition settings for the input link.

b. On the Advanced tab, change the buffer settings for the input link.

4. On the Output page, define the output data:
   a. On the Mapping tab, map the surrogate key column to the output link. If the stage has an input link, you can also map input columns to the output link.
   b. Optional: On the Advanced tab, change the buffer settings for the output link.
5. Click **OK** to save your changes and to close the Surrogate Key Generator stage editor.

Slowly Changing Dimension stage

The Slowly Changing Dimension (SCD) stage is a processing stage that works within the context of a star schema database. The SCD stage has a single input link, a single output link, a dimension reference link, and a dimension update link.

The SCD stage reads source data on the input link, performs a dimension table lookup on the reference link, and writes data on the output link. The output link can pass data to another SCD stage, to a different type of processing stage, or to a fact table. The dimension update link is a separate output link that carries changes to the dimension. You can perform these steps in a single job or a series of jobs, depending on the number of dimensions in your database and your performance requirements.

SCD stages support both SCD Type 1 and SCD Type 2 processing:

SCD Type 1
: Overwrites an attribute in a dimension table.

SCD Type 2
: Adds a new row to a dimension table.

Each SCD stage processes a single dimension and performs lookups by using an equality matching technique. If the dimension is a database table, the stage reads the database to build a lookup table in memory. If a match is found, the SCD stage updates rows in the dimension table to reflect the changed data. If a match is not found, the stage creates a new row in the dimension table. All of the columns that are needed to create a new dimension row must be present in the source data.

Input data to SCD stages must accurately represent the order in which events occurred. You might need to presort your input data by a sequence number or a date field. If a job has multiple SCD stages, you must ensure that the sort order of the input data is correct for each stage.

If the SCD stage is running in parallel, the input data must be hash partitioned by key. Hash partitioning allows all records with the same business key to be handled by the same process. The SCD stage divides the dimension table across processes by building a separate lookup table for each process.

Job design using a Slowly Changing Dimension stage

Each SCD stage processes a single dimension, but job design is flexible. You can design one or more jobs to process dimensions, update the dimension table, and load the fact table.

Processing dimensions
: You can create a separate job for each dimension, one job for all dimensions, or several jobs, each of which has several dimensions.

Updating dimensions
: You can update the dimension table as the job runs by linking the SCD stage to a database stage, or you can update the dimension table later by sending dimension changes to a flat file that you use in a separate job. Actual dimension changes are applied to the lookup table in memory and are mirrored to the dimension update link, giving you the flexibility to handle a series of changes to the same dimension row.

Loading the fact table

You can load the fact table as the final step of the job that updates the last dimension, or in a separate job.

The following figures show a series of jobs, where the first job performs the dimension lookup, the second job performs the dimension table update, and the third job loads the fact table.

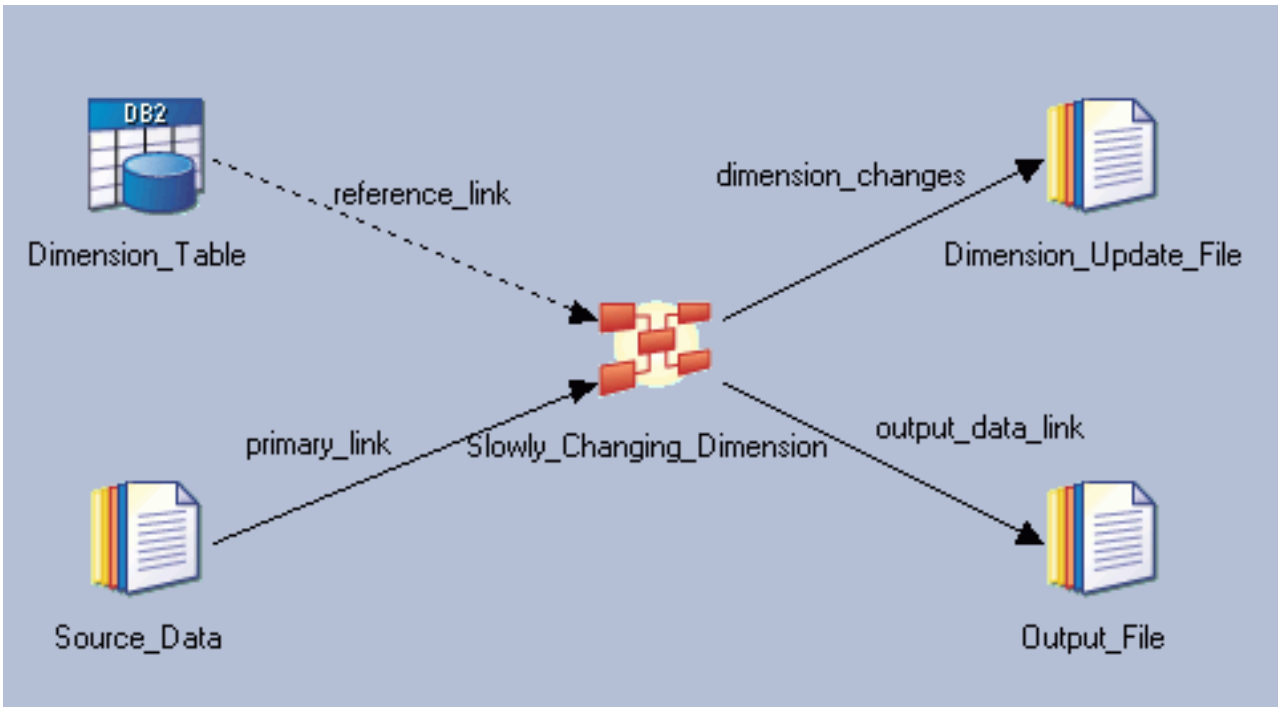


Figure 17. This job performs the dimension lookup.

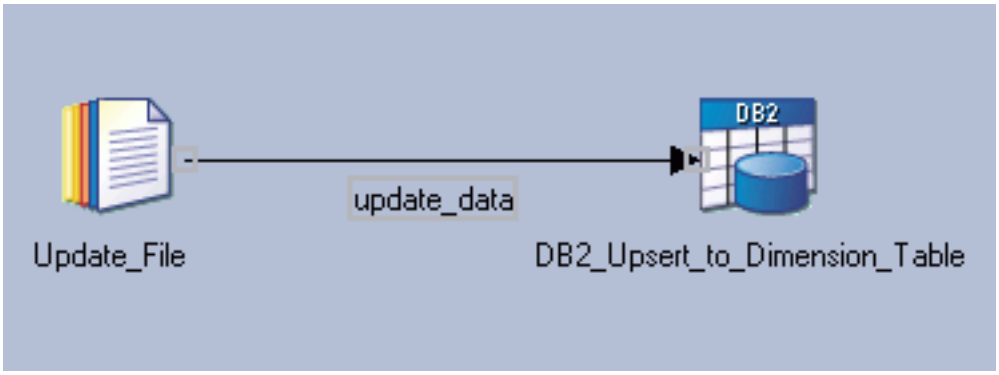


Figure 18. This job updates the dimension table.

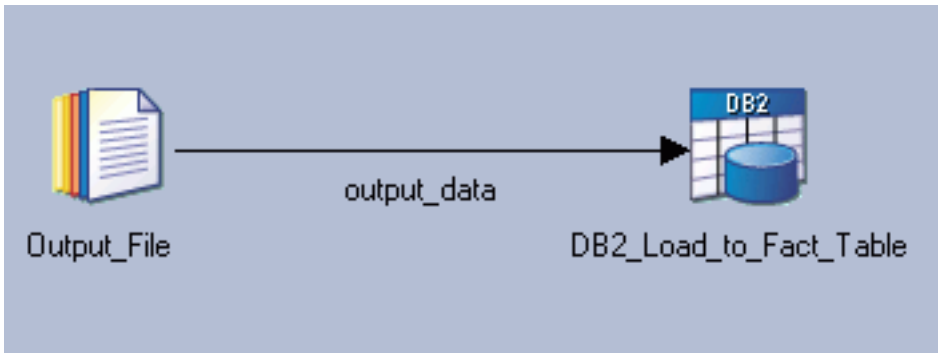


Figure 19. This job loads the fact table.

The job design shown in these figures minimizes the use of database facilities. Job 1 builds a lookup table in memory for the dimension, so the database connection is active only when the table is being created. Both the output data and the dimension update records are written to flat files. Job 2 and Job 3 use these files to update the dimension table and to load the fact table later.

This series of jobs represents a single dimension table. If you have multiple dimensions, each has a Job 1 and a Job 2. The output of the last Job 1 is the input to Job 3.

Another strategy is to combine the first two jobs into a single step, as shown in the following figure:

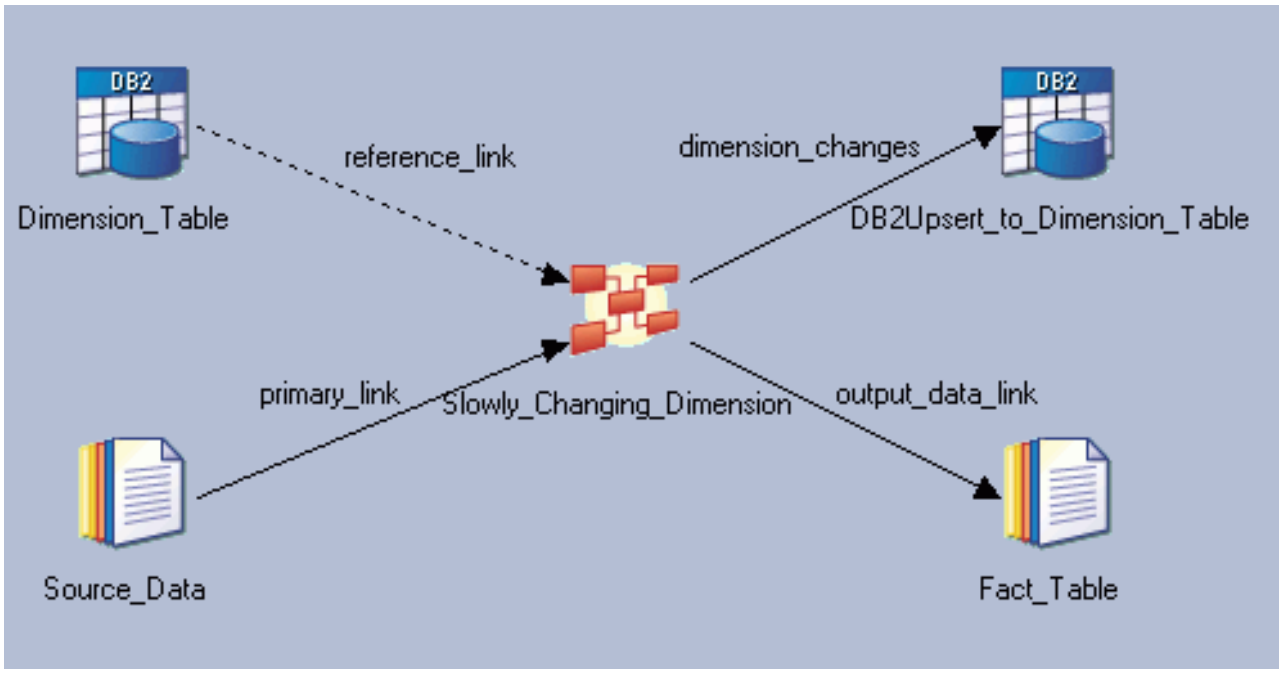


Figure 20. This job performs the dimension lookup and updates the dimension table.

Here the SCD stage provides the necessary column information to the database stage so that it can generate the correct INSERT and UPDATE SQL statements to update the dimension table. By contrast, the design in the first three figures requires you to save your output columns from the SCD stage in Job 1 as a table definition in the repository. You must then load columns from this table definition into the database stage in Job 2.

Purpose codes in a Slowly Changing Dimension stage

Purpose codes are an attribute of dimension columns in SCD stages. Purpose codes are used to build the lookup table, to detect dimension changes, and to update the dimension table.

Building the lookup table

The SCD stage uses purpose codes to determine how to build the lookup table for the dimension lookup. If a dimension has only Type 1 columns, the stage builds the lookup table by using all dimension rows. If any Type 2 columns exist, the stage builds the lookup table by using only the current rows. If a dimension has a Current Indicator column, the stage uses the derivation value of this column on the Dim Update tab to identify the current rows of the dimension table. If a dimension does not have a Current Indicator column, then the stage uses the Expiration Date column and its derivation value to identify the current rows. Any dimension columns that are not needed are not used. This technique minimizes the amount of memory that is required by the lookup table.

Detecting dimension changes

Purpose codes are also used to detect dimension changes. The SCD stage compares Type 1 and Type 2 column values to source column values to determine whether to update an existing row, insert a new row, or expire a row in the dimension table.

Updating the dimension table

Purpose codes are part of the column metadata that the SCD stage propagates to the dimension update link. You can send this column metadata to a database stage in the same job, or you can save the metadata on the Columns tab and load it into a database stage in a different job. When the database stage uses the auto-generated SQL option to perform inserts and updates, it uses the purpose codes to generate the correct SQL statements.

For more information, see “Selecting purpose codes” on page 406 and “Purpose code definitions” on page 406.

Surrogate keys in a Slowly Changing Dimension stage

Surrogate keys are used to join a dimension table to a fact table in a star schema database.

When the SCD stage performs a dimension lookup, it retrieves the value of the existing surrogate key if a matching record is found. If a match is not found, the stage obtains a new surrogate key value by using the derivation of the Surrogate Key column on the Dim Update tab. If you want the SCD stage to generate new surrogate keys by using a key source that you created with a Surrogate Key Generator stage, you must use the NextSurrogateKey function to derive the Surrogate Key column. If you want to use your own method to handle surrogate keys, you should derive the Surrogate Key column from a source column.

You can replace the dimension information in the source data stream with the surrogate key value by mapping the Surrogate Key column to the output link.

For more information, see “Specifying information about a key source” on page 407 and “Creating derivations for dimension columns” on page 407.

Editing a Slowly Changing Dimension stage

To edit an SCD stage, you must define how the stage should look up data in the dimension table, obtain surrogate key values, update the dimension table, and write data to the output link.

Before you begin

Before you edit the SCD stage, you must edit the stages on the input links:

- In the source stage, specify the file name for the data source and define the column metadata. You should also verify that the source data is sorted correctly.
- In the dimension reference stage, specify the table name and connection properties for the dimension table, and define the column metadata.

Procedure

1. Open the SCD stage editor.
2. On the Stage page, define the general stage properties:
   a. On the General tab, select the output link. The SCD stage will use the other link to update the dimension table. By default the first output link that you connect to the SCD stage is set as the output link.
   b. Optional: On the Advanced tab, change the processing settings.
3. On the Input page, define the input data to the stage:
   a. Select the reference link in the **Input name** field.
   b. On the Lookup tab, define the match condition to use for the dimension lookup.
   c. On the Lookup tab, select the purpose codes for the dimension columns.
   d. If you are using a Surrogate Key Generator stage to maintain the key source, specify information about the key source on the Surrogate Key tab.
   e. Optional: On the Advanced tab, change the buffering settings.
   f. Optional: On the Partitioning tab, change the partitioning settings. The partitioning mode is set to Auto by default. The SCD stage always uses the Hash method to partition records, based on the keys that are defined by the lookup condition.
4. On the Output page, define the output data from the stage:
   a. On the Dim Update tab, create column derivations to specify how to detect and apply changes to dimension records.
   b. On the Output Map tab, map data from the input links to the output link.
   c. Optional: If you are updating the dimension table in a separate job, save the column definitions on the Columns tab as a table definition in the repository. You can load the columns from the saved table definition into the database stage of another job. The database stage will use these columns definitions and SCD purpose codes to generate the correct INSERT and UPDATE SQL statements to update the dimension table.
   d. Optional: On the Advanced tab, change the buffering settings.
5. Click **OK** to save your changes and to close the SCD stage editor.

What to do next

After you edit the SCD stage, you must edit the stages on the output links:

- If you are updating the dimension table in the current job, specify the name and connection properties for the dimension table, set the **Write Method** property to **Upsert**, and set the **Upsert Mode** property to **Auto-generated Update & Insert**. You do not need to define column metadata because the SCD stage propagates the column definitions from the dimension reference link to the dimension update link.
- If you are loading the fact table in the current job, specify the table name and connection properties for the target database, and select any write method. The column metadata is already defined by the mappings that you specified in the SCD stage.

Defining the match condition

The match condition specifies how the SCD stage should perform the dimension lookup.

About this task

You must associate at least one pair of columns, but you can define multiple pairs if required. When you define more than one pair, the SCD stage combines the match conditions. A successful lookup requires all associated pairs of columns to match.

Procedure

1. On the Input page, select the reference link in the **Input name** field.
2. Click the **Lookup** tab.
3. Drag a source column from the left pane to a dimension column in the right pane. A relationship line is drawn between the two columns to represent the match condition.

Selecting purpose codes

Purpose codes specify how the SCD stage should process dimension data.

About this task

Purpose codes apply to columns on the dimension reference link and on the dimension update link. Select purpose codes according to the type of columns in a dimension:

- If a dimension contains a Type 2 column, you must select a Current Indicator column, an Expiration Date column, or both. An Effective Date column is optional. You cannot assign Type 2 and Current Indicator to the same column.
- If a dimension contains only Type 1 columns, no Current Indicator, Effective Date, Expiration Date, or SK Chain columns are allowed.

Procedure

1. On the Input page, select the reference link in the **Input name** field.
2. Click the **Lookup** tab.
3. In the right pane, select a purpose code for each column. You assign purpose codes in different ways depending on whether you want to assign one to a single column or to multiple columns:
   - For a single column, click the **Purpose** arrow next to the column to select a purpose code from a list.
   - For multiple columns with the same purpose code, select the columns and click **Set purpose code** from the pop-up menu. In the Set Purpose Code window, select the purpose code that you want to apply to the selected columns.

Results

After you select purpose codes on the Lookup tab, the SCD stage automatically propagates the column definitions and purpose codes to the Dim Update tab.

Purpose code definitions:

The SCD stage provides nine purpose codes to support dimension processing.

(blank)
The column has no SCD purpose. This purpose code is the default.

Surrogate Key
The column is a surrogate key that is used to identify dimension records.

Business Key
The column is a business key that is typically used in the lookup condition.

Type 1
The column is an SCD Type 1 field. SCD Type 1 column values are always current. When changes occur, the SCD stage overwrites existing values in the dimension table.

Type 2
The column is an SCD Type 2 field. SCD Type 2 column values represent a point in time. When changes occur, the SCD stage creates a new dimension row.

Current® Indicator (Type 2)
The column is the current record indicator for SCD Type 2 processing. Only one Current Indicator column is allowed.

Effective Date (Type 2)
The column is the effective date for SCD Type 2 processing. Only one Effective Date column is allowed.

Expiration Date (Type 2)
The column is the expiration date for SCD Type 2 processing. An Expiration Date column is required if there is no Current Indicator column, otherwise it is optional.

SK Chain
The column is used to link a record to the previous record or the next record by using the value of the Surrogate Key column. Only one Surrogate Key column can exist if you have an SK Chain column.

Specifying information about a key source

If you created a key source with a Surrogate Key Generator stage, you must specify how the SCD stage should use the source to generate surrogate keys.

About this task

The key source can be a flat file or a database sequence. The key source must exist before the job runs. If the key source is a flat file, the file must be accessible from all nodes that run the SCD stage.

Procedure

1. On the Input page, select the reference link in the **Input name** field.
2. Click the **Surrogate Key** tab.
3. In the **Source type** field, select the source type.
4. In the **Source name** field, type the name of the key source, or click the arrow button to browse for a file or to insert a job parameter. If the source is a flat file, type the name and fully qualified path of the state file, such as `C:\SKG\ProdDim`. If the source is a database sequence, type the name of the sequence, such as `PRODUCT_KEY_SEQ`.
5. Provide additional information about the key source according to the type:
   - If the source is a flat file, specify information in the **Flat File** area.
   - If the source is a database sequence, specify information in the **DB sequence** area.

What to do next

Calls to the key source are made by the NextSurrogateKey function. On the Dim Update tab, create a derivation that uses the NextSurrogateKey function for the column that has a purpose code of Surrogate Key. The NextSurrogateKey function returns the value of the next surrogate key when the SCD stage creates a new dimension row.

Creating derivations for dimension columns

By creating derivations for the dimension columns, you specify how to update the dimension table, including the values to use for new records and when records should expire.

About this task

Every dimension column must have a derivation. Columns with a purpose code of Current Indicator or Expiration Date must also have an Expire derivation. The following requirements apply:

- Columns with a purpose code of Type 1 or Type 2 must be derived from a source column.
- Columns with a purpose code of Current Indicator or Expiration Date must be derived from a literal value.
- Columns with a purpose code of SK Chain must be derived by using either the PrevSKChain function or the NextSKChain function.

Procedure

1. On the Output page, select the dimension update link in the **Output name** field.
2. Click the **Dim Update** tab.
3. Create column derivations by using any of these methods:
   - Drag source columns from the left pane to the **Derivation** field in the right pane
   - Use the column auto-match facility (right-click the link header and select **Auto Match**)
   - Define an expression by using the expression editor (double-click the **Derivation** or **Expire** field)

   Relationship lines show which dimension columns are derived from source columns, either directly or as part of an expression.

Mapping data to the output link

Mappings specify how to write data from the input links to the output link.

About this task

You can map input data and dimension data to the output link. Dimension data is mapped from the lookup table in memory, so new rows and changed data are available for output.

Procedure

1. On the Output page, select the output link in the **Output name** field.
2. Click the **Output Map** tab.
3. Create mappings by using any of these methods:
   - Drag columns from the left pane to the **Derivation** field in the right pane
   - Use the column auto-match facility (right-click the link header and select **Auto Match**)
   - Define an expression by using the expression editor (double-click the **Derivation** field)

   Relationship lines show which input columns are being mapped to the output link.

Dimension update action

The SCD stage updates the dimension table based on the results of the dimension lookup, plus the detection of a change in value for at least one SCD column.

Changes to Type 2 columns take precedence over changes to Type 1 columns. The following table describes how the SCD stage determines the update action for the dimension table:

Table 68. Determining update action

Lookup result	Type of column change	Update action
Dimension row not found	Not applicable	The stage creates a new dimension row by using all of the column derivations that are defined for the dimension update link on the Dim Update tab.
Dimension row found	Type 2 column value changed, regardless of whether any Type 1 column value changed	The stage expires the current dimension row and creates a new row. This maintains the history that is required for Type 2 processing. To expire a row, the stage updates only the columns with a purpose code of Current Indicator and Expiration Date by using their Expire expressions on the Dim Update tab. To create a new row, the stage derives all of the columns from the Derivation expressions on the Dim Update tab.
Dimension row found	Type 1 column value changed, but no Type 2 column value changed	The stage updates only the columns with a purpose code of Type 1 by using their Derivation expressions on the Dim Update tab.
Dimension row found	No column values changed	No updates are needed.

Pivot Enterprise stage

The Pivot Enterprise stage is a processing stage that pivots data horizontally and vertically.

The Pivot Enterprise stage is in the Processing section of the Palette pane.

Figure 21. Pivot Enterprise stage is in the Processing section of the Palette pane

Horizontal pivoting maps a set of columns in an input row to a single column in multiple output rows. The output data of the horizontal pivot action typically has fewer columns, but more rows than the input data. With vertical pivoting, you can map several sets of input columns to several output columns.

Vertical pivoting maps a set of rows in the input data to single or multiple output columns. The array size determines the number of rows in the output data. The output data of the vertical pivot action typically has more columns, but fewer rows than the input data.

For horizontal and vertical pivoting, you can also include any columns that are in the input data in the output data.

Specifying a horizontal pivot operation

Use the Pivot Enterprise stage to horizontally pivot data to map sets of input columns onto single output columns.

About this task

You can design a horizontal pivot operation in a Pivot Enterprise stage in two ways:

- You can specify the horizontal pivot operation and then map the columns onto the output columns.
- If the output columns are already defined, you can use the output columns to specify the horizontal pivot operation.

Specifying a horizontal pivot operation and mapping output columns

You can specify the horizontal pivot operation and then map the resulting columns onto output columns.

About this task

Use horizontal pivot operations to map rows into output columns.

Procedure

1. Design your basic job by doing the following tasks:
   a. Place a Pivot Enterprise stage on your job design canvas.
   b. Place the stage that will supply the input data on the canvas on the left of the Pivot Enterprise stage.
   c. Place the stage that is the target for the output data on the canvas on the right of the Pivot Enterprise stage.
   d. Link the three stages together.
2. Configure the stage that provides the input data.
3. Open the Pivot Enterprise stage and click the **Properties** tab.
4. Specify the **Pivot type** as **Horizontal** on the **Pivot Action** tab.
5. Specify the horizontal pivot operation on the **Pivot Definitions** tab of the Stage page by doing the following tasks:
   a. In the **Name** field, type the name of the output column that will contain the pivoted data (the pivot column).
   b. Specify the SQL type and, if necessary (for example if the SQL type is decimal), the length and scale for the pivoted data.
   c. Double-click the **Derivation** field to open the Column Selection window.
   d. In the **Available Columns** list, select the columns that you want to combine in the pivot column.
   e. Click the right arrow to move the selected column to the **Selected Columns** list.
   f. Click **OK** to return to the **Pivot** tab.
   g. If you want the stage to number the pivoted rows by generating a pivot index column, select **Pivot Index**.
6. Click the **Output** page to go to the **Mapping** tab.
7. Specify your output data by doing the following tasks:
   a. Select your pivot column or columns from the **Columns** table in the right pane and drag them over to the output link table in the left pane.
   b. Drag all the columns that you want to appear in your output data from the available columns on the left side of the window to the available output columns on the right side of the window.
   c. Click the **Columns** tab to view the data that the stage will output when the job runs.
8. Click **OK** to save your pivot operation and close the Pivot Enterprise stage.

Specifying a horizontal pivot operation by using output columns

You can specify the horizontal pivot operation by using output columns if the output columns have already been defined.

About this task

If you are designing a job where the target data has already been defined, you can define the horizontal pivot operation by using the target data output columns.

Procedure

1. Design your basic job by doing the following tasks:
   a. Place a Pivot Enterprise stage on your job design canvas.
   b. Place the stage that will supply the input data on the canvas on the left of the Pivot Enterprise stage.
   c. Place the stage that is the target for the output data on the canvas on the right of the Pivot Enterprise stage.
   d. Link the three stages together.
2. Configure the stage that provides the input data.
3. Configure the target stage so that the target data is defined.
4. Open the Pivot Enterprise stage and click the **Properties** tab.
5. Specify the **Pivot type** as **Horizontal** on the **Pivot Action** tab.
6. Specify the horizontal pivot operation on the **Pivot Definitions** tab of the Stage page by doing the following tasks:
   a. Click **Load** to load all the defined output columns into the **Pivot Definitions** tab
   b. Double-click the **Derivation** field of the pivot column to open the "Column Select" window.
   c. In the **Available Columns** list, select the columns that you want to combine in the pivot column.
   d. Click the right arrow to move the selected column to the **Selected Columns** list.
   e. Click **OK** to return to the **Pivot Definitions** tab.
7. Click **Output** to go to the **Mapping** tab of the Output page.
8. Specify your output data by doing the following tasks:
   a. Select your pivot column or columns from the **Columns** table in the right pane and drag them over to the output link table in the left pane.
   b. Drag all the columns that you want to appear in your output data from the available columns on the left side of the window to the available output columns on the right side of the window.
   c. Click the **Columns** tab to view the data that the stage will output when the job runs.
9. Click **OK** to save your pivot operation and close the Pivot Enterprise stage.

Example of horizontally pivoting data

In this example, the Pivot Enterprise stage is set up to horizontally pivot some data.

You can generate a pivot index that will assign an index number to each row within sets of horizontally pivoted data. The following tables provide examples of data before and after a horizontal pivot operation.

Table 69. Input data for a simple horizontal pivot operation

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 70. Output data for a simple horizontal pivot operation

REPID	last_name	Q1sales	Pivot_index
100	Smith	1234.08	0
100	Smith	1456.80	1

Table 70. Output data for a simple horizontal pivot operation (continued)

REPID	last_name	Q1sales	Pivot_index
100	Smith	1578.00	2
101	Yamada	1245.20	0
101	Yamada	1765.00	1
101	Yamada	1934.22	2

Table 71. Input data for a horizontal pivot operation with multiple pivot columns

REPID	last_name	Q1sales	Q2sales	Q3sales	Q4sales
100	Smith	4268.88	5023.90	4321.99	5077.63
101	Yamada	4944.42	5111.88	4500.67	4833.22

Table 72. Output data for a horizontal pivot operation with multiple pivot columns

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 following screen capture shows the settings for a horizontal pivot on the **Pivot Definition** tab of the Stage page.

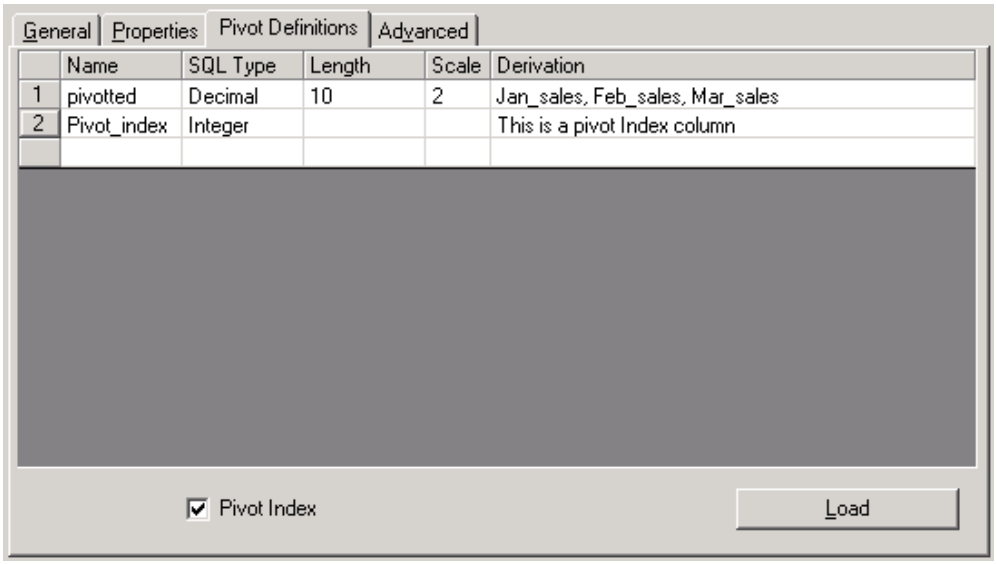


Figure 22. Example of horizontal pivot settings on the Pivot Definitions tab

The following screen capture shows the settings for a horizontal pivot operation on the **Mapping** tab of the Output page.

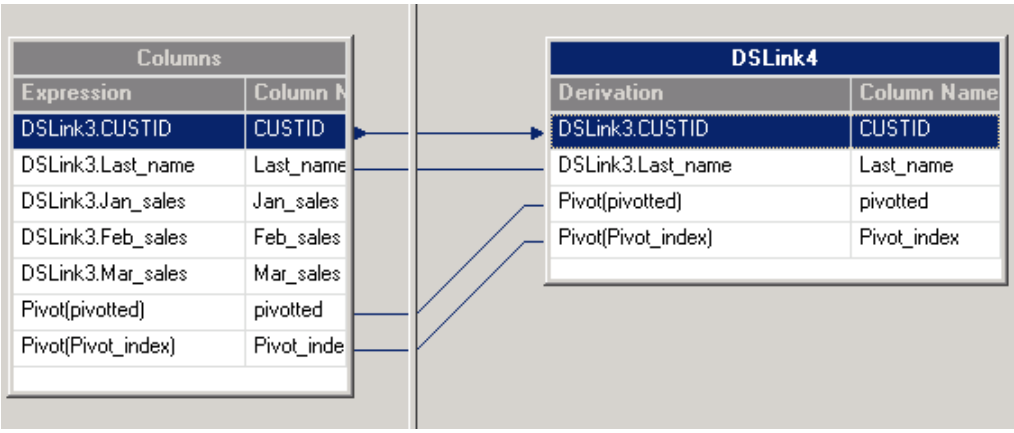


Figure 23. Mapping tab

Specifying a vertical pivot operation

Use the Pivot Enterprise stage to vertically pivot data and then map the resulting columns onto the output columns.

About this task

You can design a vertical pivot operation in a Pivot Enterprise stage and then map the resulting columns onto the output columns.

Specifying a vertical pivot operation and mapping output columns

You can specify the vertical pivot operation and then map the resulting columns onto output columns.

About this task

You can configure a pivot operation to vertically pivot data and then map the resulting columns onto output columns.

Procedure

1. Design your basic job by doing the following tasks:
   a. Place a Pivot Enterprise stage on your job design canvas.
   b. Place the stage that will supply the input data on the canvas on the left of the Pivot Enterprise stage.
   c. Place the stage that is the target for the output data on the canvas on the right of the Pivot Enterprise stage.
   d. Link the three stages together.
2. Configure the stage that provides the input data.
3. Open the Pivot Enterprise stage and click the **Properties** tab.
4. Select the **Pivot Type** property and select **Vertical** for the **Pivot Type**.
5. Specify the vertical pivot operation on the **Pivot Definitions** tab of the Stage page by doing the following tasks:
   a. Click **Load** to load the input data.
   b. Select **GroupBy** for each column that you want to group in the output.
   c. Select **Pivot** for each column that you want to pivot from rows to columns. You cannot pivot columns that have **GroupBy** selected.

d. Double-click the **Aggregation functions required for this column** field to open the Pivot - Function Select window.
e. In the **Available Columns** list, select the columns that you want to aggregate in the pivot column and click **Save**.
f. Specify the **Array Size** to specify the number of sets of pivoted data that will be generated for each output row. For example, specify an array size of 7 to generate a row of pivoted data in 7 columns, with each column for a day of the week.
g. If you want the stage to generate a pivot index column to number the pivoted columns, select **Pivot Index**.
h. Click **OK** to return to the **Pivot** tab.

6. Click **Output** to go to the **Mapping** tab of the Output page.
7. Specify your output data by doing the following tasks:
   a. Select your pivot column or columns from the **Columns** table in the right pane and drag them over to the output link table in the left pane.
   b. Drag all the columns that you want to appear in your output data from the available columns on the left side of the window to the available output columns on the right side of the window.
   c. Click the **Columns** tab to view the data that the stage will output when the job runs.
8. Click **OK** to save your pivot operation and close the Pivot Enterprise stage.

Example of vertically pivoting data

In this example, the Pivot Enterprise stage is set up to vertically pivot some data and aggregate a monthly average of sales.

The following tables provide examples of data before and after a vertical pivot operation.

Table 73. Input data for vertical pivot operation

REPID	last_name	Q_sales
100	Smith	1234.08
100	Smith	1456.80
100	Smith	1578.00
101	Yamada	1245.20
101	Yamada	1765.00
101	Yamada	1934.22

Table 74. Out put data for vertical pivot operation

REPID	last_name	Q_sales (January)	Q_sales1 (February)	Q_sales2 (March)	Q_sales_average
100	Smith	1234.08	1456.80	1578.00	1412.96
101	Yamada	1245.20	1765.00	1934.22	1648.14

The following screen capture shows the settings for a vertical pivot on the **Pivot Definitions** tab of the Stage page.

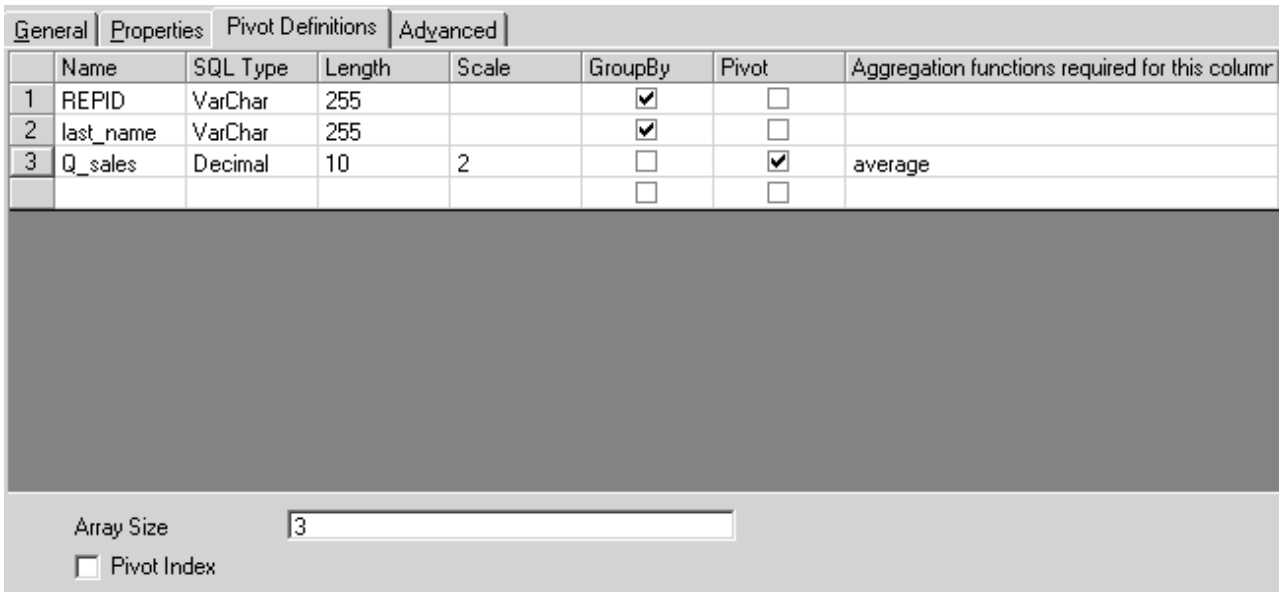


Figure 24. Example of vertical pivot settings on the Pivot Definitions tab

The following screen capture shows the settings for a vertical pivot operation on the **Mapping** tab of the Output page.

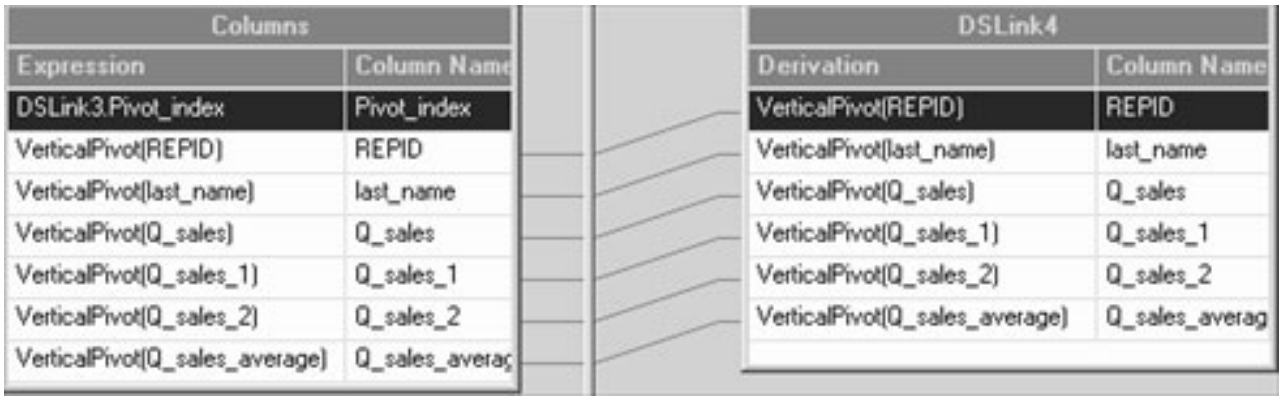


Figure 25. Mapping tab

Pivot Enterprise stage: Properties tab

Specify **Pivot Type** as **Horizontal** or **Vertical** on the **Properties** tab.

Use the Properties tab to specify whether the stage pivots data horizontally or vertically.

Pivot Type

Specify **Horizontal** to map columns into rows or **Vertical** to map rows into columns.

Specifying execution options

You can specify whether your stage runs sequentially or in parallel, whether underlying operators can be combined into a single process, and whether the job should try to maintain data in current partitions.

About this task

The execution options are set on the **Advanced** tab of the Stage page.

Procedure

1. Specify the sequential or parallel execution in the **Execution Mode** list. If the execution mode for a particular stage cannot be changed, the list is not displayed.
2. Specify the combinability mode by selecting an option from the **Combinability mode** list. Select one of the following options:

Option	Description
Auto	Use the default combination setting for this operator.
Combinable	Ignore the default setting and combine operators if possible.
Do not combine	Never combine operators.

3. Specify whether the stage requires that the data partitioning is preserved by the next stage in the job by selecting one of the following options from the **Preserve partitioning** list:

Option	Description
Set	Specifies that the next stage must preserve data partitioning if possible.
Clear	Specifies the partitioning method that the next stage uses is not relevant.
Propagate	Specifies that the stage uses the option that the previous stage in the job has used.

Specifying where the stage runs

You can specify on which processing nodes the stage processes are run, and constrain the stage to use certain resources.

About this task

You can limit the stage to running on a group of nodes that have been defined as a node pool in the configuration file, or you can define a node map that is local to the stage. You can also limit the stage to using a set of resources defined in a resource pool in the configuration file.

Procedure

1. Open the **Advanced** tab of the Stage page.
2. To make the stage run on a predefined node pool or use a predefined resource pool:
   a. Select the configuration file that defines your node and resource pools from the list.
   b. Select a **Node pool** or **Resource pool** from the **Constraint** list.
   c. If you are specifying a resource pool, select a **Type** for a resource pool
   d. Select the name of the pool that you are limiting execution to.
   e. Repeat steps a - d to specify more pools.
3. To define a node map of nodes the stage can run on:
   a. Select **Node map constraint**.
   b. In the text field, type in, or browse for, the names of the nodes to which execution will be limited.
4. Click **OK** to close the stage editor, or select another page or tab to open.

Specifying partitioning or collecting methods

You can specify how the data is collected or partitioned before it is processed.

Partitioning data

About this task

If the stage is running in parallel mode, it processes the data in partitions. By default, the partitioning method is set to Auto. You can override the default behavior.

Procedure

1. Open the **Partitioning** tab of the Input page.
2. Select a partitioning method from the list:

Option	Description
(Auto)	InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for most stages.
DB2	Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
Entire	Each file written to receives the entire data set.
Hash	The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
Modulus	The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
Random	The records are partitioned randomly, based on the output of a random number generator.
Round Robin	The records are partitioned on a round robin basis as they enter the stage.
Same	Preserves the partitioning already in place.
Range	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. Requires extra properties to be set. Access these properties by clicking the properties button.

3. If you selected the hash or modulus partitioning methods, specify a key by clicking on one or more of the columns in the **Available** list. The selected column or columns appear in the **Selected** list.

Collecting data

You can specify a collecting method.

About this task

If the stage runs sequentially, and the previous stage in the job runs in parallel, then the data is collected before being written. By default, the collecting method is set to Auto. You can override the default behavior.

Procedure

1. Open the **Partitioning** tab of the Input page.
2. Select a collecting method from the list:

Option	Description
(Auto)	This is the default collection method for the Sequential File stage. Normally, when you are using Auto mode, InfoSphere DataStage will read any row from any input partition as it becomes available.
Ordered	Reads all rows from the first partition, then all rows from the second partition, and so on.
Round Robin	Reads a row from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
Sort Merge	Reads rows in an order based on one or more columns of the row. This requires you to select a collecting key column from the **Available** list.

3. If you selected the Sort Merge collecting method, specify a collecting key by clicking on one or more of the columns in the **Available** list. The selected column or columns appear in the **Selected** list.

Specifying a sort operation

You can specify that the stage sorts data before processing it. You specify the sort operation on the **Partitioning** tab of the Input page to sort the data as it enters the stage.

Before you begin

You cannot sort data if you are using the Auto partitioning method.

About this task

You specify one or more columns in the input data to act as sorting keys. The operation sorts the key columns, and associated rows, into the specified alphanumeric order. If you require a more complex sort operation, you can use the Sort stage.

Procedure

1. Select **Perform Sort**.
2. Select **Stable** to preserve previously sorted data sets.
3. Select **Unique** to specify that, if multiple rows have identical sorting key values, only one row is retained. If **Stable** is also selected, the first row is retained.
4. Select one or more columns in the **Available** list to move them to the **Selected** list. These columns form the sort key.
5. Right-click on each key column in the **Selected** list and select from the following options:

Option	Description
Sort direction	Select **Ascending** or **Descending** as required.
Case sensitivity	Select **Case sensitive** or **Case insensitive** as required.
Sort as EBCDIC	Select this option to sort characters as EBCDIC rather than ASCII.
Nulls position	Select **First** or **Last** to specify whether Null values should be sorted to the first or the last position.
Usage	If you are using a keyed partitioning method, specify whether the current column is to be used for sorting, partitioning, or both.

6. If National Language Support is enabled, click the properties button in the sort box to open the **NLS Locale** tab of the Sort Properties window. You can view the current default collate convention, and select another one if required.

Checksum stage

Use the Checksum stage to generate a checksum value from the specified columns in a row and add the checksum to the row.

You can use the checksum value to check the validity of each row when it is written to the data target. If the checksum value does not equate to the columns from which it was generated, then the data is corrupt and is no longer valid.

Typically you create the Checksum stage in one job to add the checksum column, and then use a Checksum stage in another job to check the validity of the data.

The Checksum stage is in the Processing section of the palette.

Adding a checksum column to your data

You can add a checksum column to your data by adding a Checksum stage in your data flow.

Procedure

1. Place a Checksum stage on your job design canvas and position it so that it receives data from your data source, and writes it to your data target.
2. Open the checksum stage editor and optionally supply values for the following properties:

Option	Description
Computation Mode	By default set to **Use all columns**. You can also choose to explicitly exclude columns, or explicitly include them. If you choose either of these options, then you must fill in additional properties to specify the columns to exclude, or to include.
Buffer Output Column Name	By default this property is not set. You can use the property to specify that the buffer used in the generation of the checksum should be included in an additional column with the output data. The buffer contains the actual values in the columns used to generate the checksum.
Checksum Output Column Name	By default, the output column that contains the checksum value is named Checksum, but you can specify an alternative name in this property.

3. Open the **Mapping** tab of the **Output** page and specify how your output columns are derived from your input columns, and the data generated by the stage.

Properties for Checksum Stage

In most cases, you can use the default values for the properties on the **Properties** tab.

Default properties

These properties take default values unless you set the properties explicitly.

Computation Mode

Use this property to specify whether the checksum value is generated from all available columns (the default), all columns except the ones specified, or only the columns specified.

If you select **Use all columns except those specified**, the following property is displayed:

Exclude Column
Select a column from the **Exclude columns** list. You can repeat this property to exclude multiple columns.

If you select **Use only columns specified**, the following property is displayed:

Compute Column
Select a column from the **Compute columns** list. You can repeat this property to include multiple columns in the computation of the checksum.

Optional properties

You can set these properties to further control how the checksum stage operates.

Buffer Output Column Name
Use this property to specify the name of the output column that contains the buffer that the checksum algorithm was run with. If you do not specify this property, no output column is generated.

Checksum Output Column Name
Use this property to specify the name of the output column that contains the checksum value. If you do not specify a name, the column is named "Checksum".

Mapping output columns

You use the **Mapping** tab to specify which columns are output by the Checksum stage.

About this task

The **Mapping** tab lists the columns that are input to the stage and the columns that are generated by the stage. You can select columns from this list to build a list of columns to be output by the stage.

The **Mapping** tab is synchronized with the **Columns** tab on the Output page, so that the **Columns** tab is automatically populated when you map your columns. If your **Columns** tab already contains column definitions, these definitions are displayed on the **Mapping** tab.

You can map all the columns in a single operation, or you can drag columns one at a time and drop each one onto an existing output column.

Procedure

1. In the right pane of the **Mapping** tab, select one or more of the columns that you want to output from the stage.
2. Drag the column or columns over to the right pane and release the mouse button when the cursor changes shape.

Specifying execution options

You can specify whether your stage runs sequentially or in parallel, whether underlying operators can be combined into a single process, and whether the job should try to maintain data in current partitions.

About this task

The execution options are set on the **Advanced** tab of the Stage page.

Procedure

1. Specify the sequential or parallel execution in the **Execution Mode** list. If the execution mode for a particular stage cannot be changed, the list is not displayed.
2. Specify the combinability mode by selecting an option from the **Combinability mode** list. Select one of the following options:

Option	Description
Auto	Use the default combination setting for this operator.
Combinable	Ignore the default setting and combine operators if possible.
Do not combine	Never combine operators.

3. Specify whether the stage requires that the data partitioning is preserved by the next stage in the job by selecting one of the following options from the **Preserve partitioning** list:

Option	Description
Set	Specifies that the next stage must preserve data partitioning if possible.
Clear	Specifies the partitioning method that the next stage uses is not relevant.
Propagate	Specifies that the stage uses the option that the previous stage in the job has used.

Specifying where the stage runs

You can specify on which processing nodes the stage processes are run, and constrain the stage to use certain resources.

About this task

You can limit the stage to running on a group of nodes that have been defined as a node pool in the configuration file, or you can define a node map that is local to the stage. You can also limit the stage to using a set of resources defined in a resource pool in the configuration file.

Procedure

1. Open the **Advanced** tab of the Stage page.
2. To make the stage run on a predefined node pool or use a predefined resource pool:
   a. Select the configuration file that defines your node and resource pools from the list.
   b. Select a **Node pool** or **Resource pool** from the **Constraint** list.
   c. If you are specifying a resource pool, select a **Type** for a resource pool
   d. Select the name of the pool that you are limiting execution to.
   e. Repeat steps a - d to specify more pools.
3. To define a node map of nodes the stage can run on:
   a. Select **Node map constraint**.
   b. In the text field, type in, or browse for, the names of the nodes to which execution will be limited.
4. Click **OK** to close the stage editor, or select another page or tab to open.

Specifying partitioning or collecting methods

You can specify how the data is collected or partitioned before it is processed.

Partitioning data

About this task

If the stage is running in parallel mode, it processes the data in partitions. By default, the partitioning method is set to Auto. You can override the default behavior.

Procedure

1. Open the **Partitioning** tab of the Input page.
2. Select a partitioning method from the list:

Option	Description
(Auto)	InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for most stages.
DB2	Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
Entire	Each file written to receives the entire data set.
Hash	The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
Modulus	The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
Random	The records are partitioned randomly, based on the output of a random number generator.
Round Robin	The records are partitioned on a round robin basis as they enter the stage.
Same	Preserves the partitioning already in place.
Range	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. Requires extra properties to be set. Access these properties by clicking the properties button.

3. If you selected the hash or modulus partitioning methods, specify a key by clicking on one or more of the columns in the **Available** list. The selected column or columns appear in the **Selected** list.

Collecting data

You can specify a collecting method.

About this task

If the stage runs sequentially, and the previous stage in the job runs in parallel, then the data is collected before being written. By default, the collecting method is set to Auto. You can override the default behavior.

Procedure

1. Open the **Partitioning** tab of the Input page.
2. Select a collecting method from the list:

Option	Description
(Auto)	This is the default collection method for the Sequential File stage. Normally, when you are using Auto mode, InfoSphere DataStage will read any row from any input partition as it becomes available.
Ordered	Reads all rows from the first partition, then all rows from the second partition, and so on.
Round Robin	Reads a row from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
Sort Merge	Reads rows in an order based on one or more columns of the row. This requires you to select a collecting key column from the **Available** list.

3. If you selected the Sort Merge collecting method, specify a collecting key by clicking on one or more of the columns in the **Available** list. The selected column or columns appear in the **Selected** list.

Specifying a sort operation

You can specify that the stage sorts data before processing it. You specify the sort operation on the **Partitioning** tab of the Input page to sort the data as it enters the stage.

Before you begin

You cannot sort data if you are using the Auto partitioning method.

About this task

You specify one or more columns in the input data to act as sorting keys. The operation sorts the key columns, and associated rows, into the specified alphanumeric order. If you require a more complex sort operation, you can use the Sort stage.

Procedure

1. Select **Perform Sort**.
2. Select **Stable** to preserve previously sorted data sets.
3. Select **Unique** to specify that, if multiple rows have identical sorting key values, only one row is retained. If **Stable** is also selected, the first row is retained.
4. Select one or more columns in the **Available** list to move them to the **Selected** list. These columns form the sort key.
5. Right-click on each key column in the **Selected** list and select from the following options:

Option	Description
Sort direction	Select **Ascending** or **Descending** as required.
Case sensitivity	Select **Case sensitive** or **Case insensitive** as required.
Sort as EBCDIC	Select this option to sort characters as EBCDIC rather than ASCII.
Nulls position	Select **First** or **Last** to specify whether Null values should be sorted to the first or the last position.
Usage	If you are using a keyed partitioning method, specify whether the current column is to be used for sorting, partitioning, or both.

6. If National Language Support is enabled, click the properties button in the sort box to open the **NLS Locale** tab of the Sort Properties window. You can view the current default collate convention, and select another one if required.

Chapter 7. Cleansing your Data

Use the stages in the Data Quality section of the palette to cleanse your data.

The Data Quality stages are described in detail in the documentation that describes data cleansing with InfoSphere QualityStage jobs.

Chapter 8. Restructuring Data

Use the stages in the restructure section of the palette to restructure complex data.

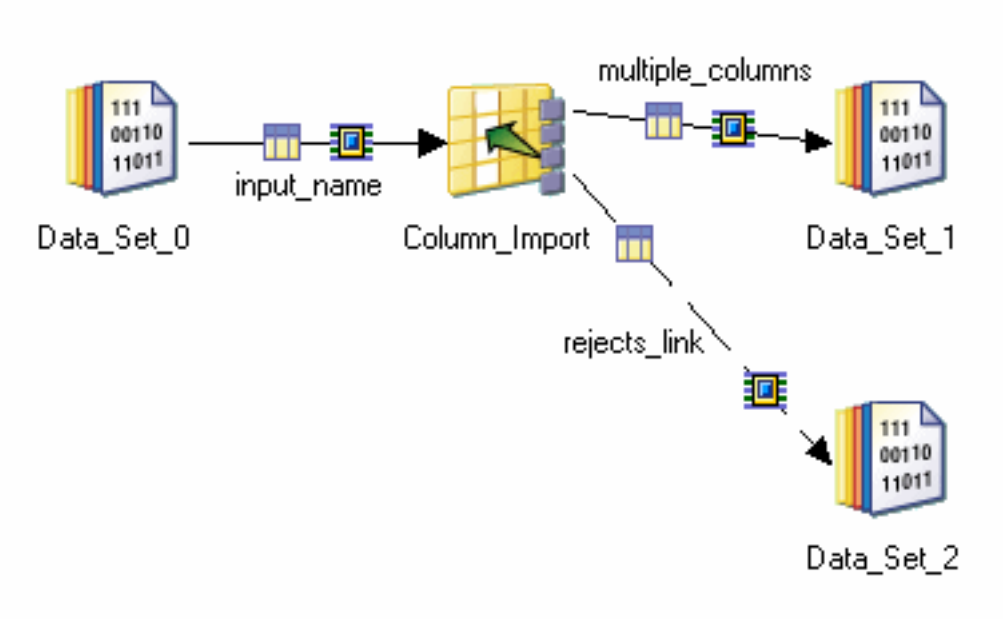
Column Import stage

The Column Import stage is a restructure stage. It can have a single input link, a single output link and a single rejects link. The complement to this stage is the Column Export stage, described in "Column Export stage" on page 441.

The Column Import stage imports data from a single column and outputs it to one or more columns. You would typically use it to divide data arriving in a single column into multiple columns. The data would be fixed-width or delimited in some way to tell the Column Import stage where to make the divisions. The input column must be a string or binary data, the output columns can be any data type.

You supply an import table definition to specify the target columns and their types. This also determines the order in which data from the import column is written to output columns. Information about the format of the incoming column (for example, how it is delimited) is given in the Format tab of the Output page. You can optionally save reject records, that is, records whose import was rejected, and write them to a rejects link.

In addition to importing a column you can also pass other columns straight through the stage. So, for example, you could pass a key column straight through.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Examples

This section gives examples of input and output data from a Column Import stage to give you a better idea of how the stage works.

In this example the Column Import stage extracts data from 16-byte raw data field into four integer output fields. The input data set also contains a column which is passed straight through the stage. The example assumes that the job is running sequentially. The metadata is as follows:

Table 75. Metadata

Column name	Key	SQL type
keycol	Yes	Char
col_to_import		Binary

These are the rows from the input data set:

Keycol col_to_import

a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

b 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01

c 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02

d 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03

e 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04

f 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05 05

g 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06 06

h 07 07 07 07 07 07 07 07 07 07 07 07 07 07 07 07

i 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08

j 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09

The import table definition can either be supplied on the Output Page Columns tab or in a schema file. For the example, the definition would be:

Table 76. Import table definition

Column name	Key	SQL type
keycol	Yes	Char
col1		Integer
col2		Integer
col3		Integer
col4		Integer

You have to give InfoSphere DataStage information about how to treat the imported data to split it into the required columns. This is done on the Output page **Format Tab**. For this example, you specify a data format of binary to ensure that the contents of col_to_import are interpreted as binary integers, and that the data has a field delimiter of none

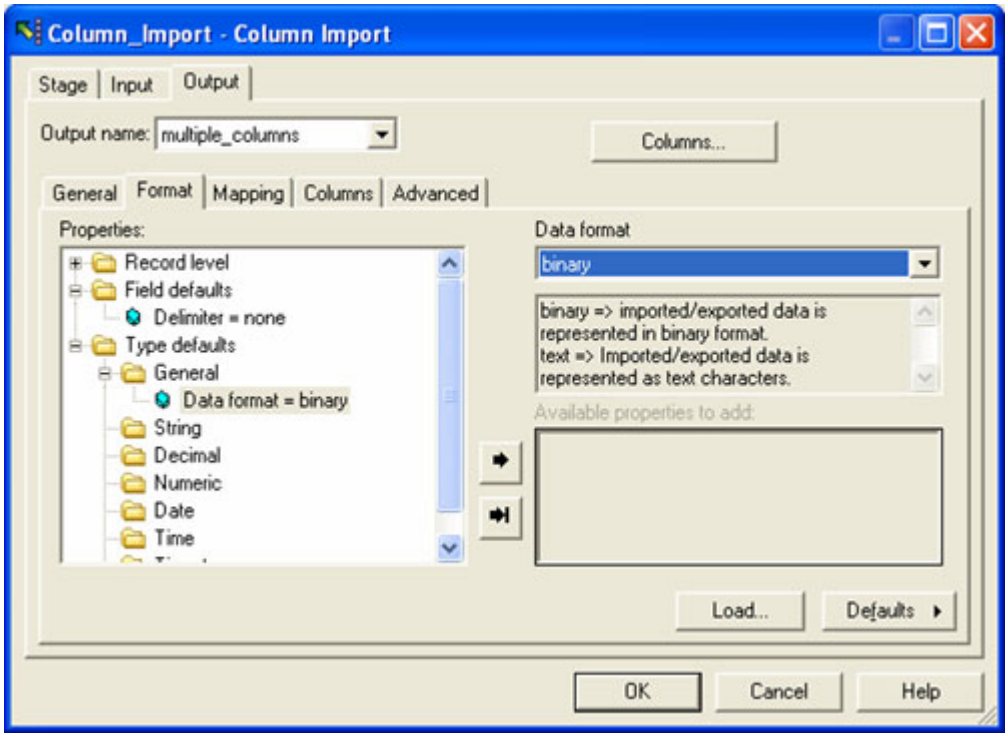


Figure 26. Format tab

.

The properties of the Column Import stage are set as follows:

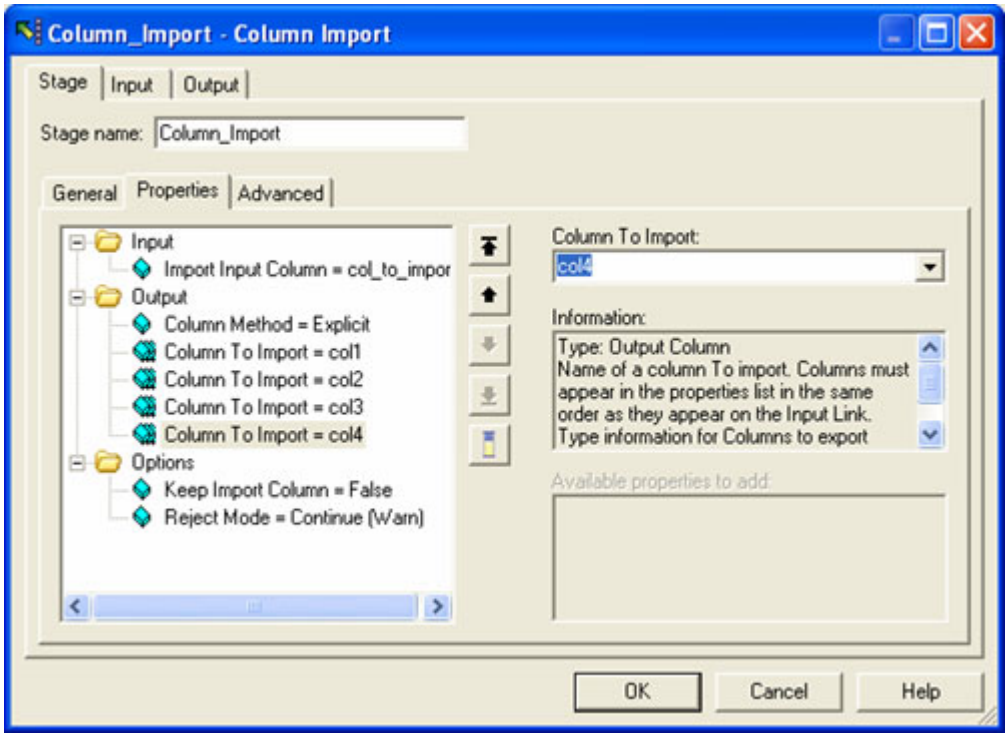


Figure 27. Properties tab

The output data set will be:

Table 77. Output data set

col1	col2	col3	col4	key
0	0	0	0	a
16843009	16843009	16843009	16843009	b
33686018	33686018	33686018	33686018	c
50529027	50529027	50529027	50529027	d
67372036	67372036	67372036	67372036	e
84215045	84215045	84215045	84215045	f
101058054	101058054	101058054	101058054	g
117901063	117901063	117901063	117901063	h
134744072	134744072	134744072	134744072	i
151587081	151587081	151587081	151587081	j

Column Import stage: fast path

This section specifies the minimum steps to take to get a Column Import stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Column Import stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

1. Go to the Stage page **Properties Tab**, under the Input category.
2. Specify the column that you are importing. Under the Output category:
   a. Choose the Column method, this is Explicit by default, meaning you specify explicitly choose output columns as destinations. The alternative is to specify a schema file.
   b. If you are using the Explicit method, choose the output column(s) to carry your imported input column. Repeat the Column to Import property to specify all the columns you need.
   c. If you are using the Schema File method, specify the schema file that gives the output column details.
3. In the Output page **Format Tab** specify the format of the column you are importing. This informs InfoSphere DataStage about data format and enables it to divide a single column into multiple columns.
4. In the Output page **Mapping Tab** check that the input columns are mapping onto the output columns as you expect. The mapping is carried out according to what you specified in the **Properties** tab.

Column Import stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Column Import stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 78. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Input/Import Input Column	Input Column	N/A	Y	N	N/A
Output/Column Method	Explicit/Schema File	Explicit	Y	N	N/A
Output/Column to Import	Output Column	N/A	Y (if Column Method = Explicit)	Y	N/A
Output/Schema File	Pathname	N/A	Y (if Column Method = Schema file)	N	N/A
Options/Keep Input Column	True/False	False	N	N	N/A
Options/Reject Mode	Continue (warn) /Output/Fail	Continue	N	N	N/A

Column Import stage: Input category:
Import input column

Specifies the name of the column containing the string or binary data to import.

Column Import stage: Output category:
Column method

Specifies whether the columns to import should be derived from column definitions on the Output page **Columns** tab (Explicit) or from a schema file (Schema File).

Column to import

Specifies an output column. The meta data for this column determines the type that the import column will be converted to. Repeat the property to specify multiple columns. You can use the Column Selection dialog box to select multiple columns at once if required . You can specify the properties for each column using the Parallel tab of the Edit Column Meta dialog box (accessible from the shortcut menu on the columns grid of the output Columns tab). The order of the Columns to Import that you specify should match the order on the Columns tab.

Schema File

Instead of specifying the source data type details via output column definitions, you can use a schema file (note, however, that if you have defined columns on the **Columns** tab, you should ensure these match the schema file). Type in a pathname or browse for a schema file.

Column Import stage: Options category:

Keep Input Column

Specifies whether the original input column should be transferred to the output data set unchanged in addition to being imported and converted. Defaults to False.

Reject mode

The values of this property specify the following actions:

- **Fail.** The stage fails when it encounters a record whose import is rejected.
- **Output.** The stage continues when it encounters a reject record and writes the record to the reject link.
- **Continue.** The stage is to continue but report failures to the log file.

Column Import stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Column Import stage: Input page

The Input page allows you to specify details about the incoming data sets. The Column Import stage expects one incoming data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being imported. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Column Import stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Column Import stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is imported. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Column Import stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Column Import stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Column Import stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Column Import stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Column Import stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Column Import stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being imported. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.

- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Column Import stage: Output page

The Output page allows you to specify details about data output from the Column Import stage. The Column Import stage can have only one output link, but can also have a reject link carrying records that have been rejected.

The General tab allows you to specify an optional description of the output link. The Format tab allows you to specify details about how data in the column you are importing is formatted so the stage can divide it into separate columns. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Column Import stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output links.

Details about Column Import stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Output link format tab

The Format tab allows you to supply information about the format of the column you are importing. You use it in the same way as you would to describe the format of a flat file you were reading. The tab has a similar format to the Properties tab.

To change individual properties, select a property type from the main tree then add the properties you want to set to the tree structure by clicking on them in the **Available properties to add** window. You can then set a value for that property in the Property Value box. Pop-up help for each of the available properties appears if you hover the mouse pointer over it.

Any property that you set on this tab can be overridden at the column level by setting properties for individual columns on the Edit Column Metadata dialog box (see Columns Tab).

This description uses the terms "record" and "row" and "field" and "column" interchangeably.

The following sections list the property types and properties available for each type.

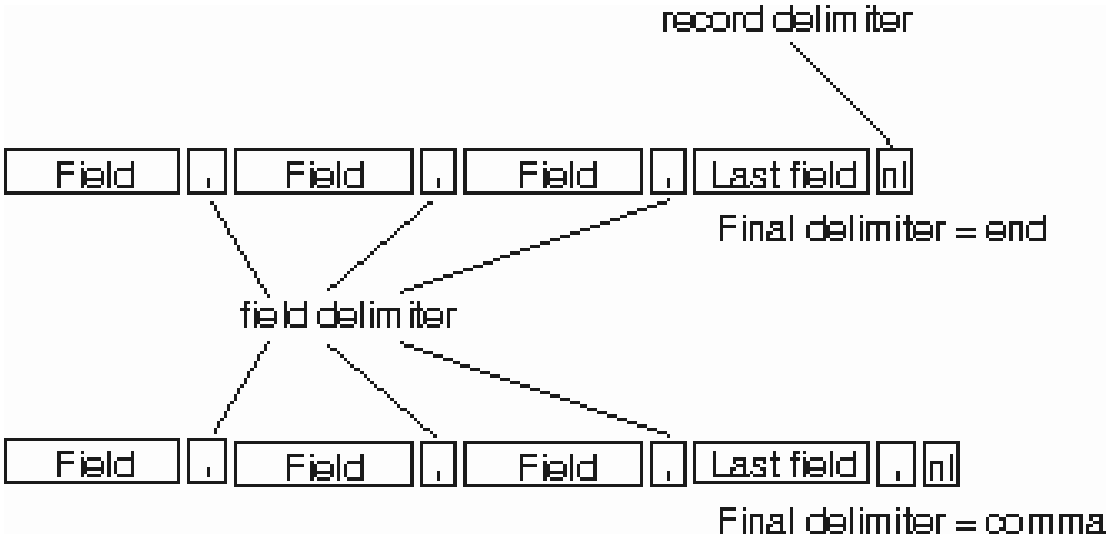
Record level

These properties define details about how data records are formatted in the flat file. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Fill char.** Does not apply to output links.
- **Final delimiter string**. Specify the string written after the last column of a record in place of the column delimiter. Enter one or more characters, this precedes the record delimiter if one is used. Mutually exclusive with Final delimiter, which is the default. For example, if you set Delimiter to comma and Final delimiter string to `, ` (comma space - you do not need to enter the inverted

commas) all fields are delimited by a comma, except the final field, which is delimited by a comma followed by an ASCII space character. InfoSphere DataStage skips the specified delimiter string when reading the file.

- **Final delimiter**. Specify the single character written after the last column of a record in place of the field delimiter. Type a character or select one of whitespace, end, none, null, tab, or comma. InfoSphere DataStage skips the specified delimiter string when reading the file. See the following diagram for an illustration.
  - **whitespace**. The last column of each record will not include any trailing white spaces found at the end of the record.
  - **end**. The last column of each record does not include the field delimiter. This is the default setting.
  - **none**. The last column of each record does not have a delimiter, used for fixed-width fields.
  - **null**. The last column of each record is delimited by the ASCII null character.
  - **comma**. The last column of each record is delimited by the ASCII comma character.
  - **tab**. The last column of each record is delimited by the ASCII tab character.



- **Intact**. The intact property specifies an identifier of a partial schema. A partial schema specifies that only the column(s) named in the schema can be modified by the stage. All other columns in the row are passed through unmodified. The file containing the partial schema is specified in the Schema File property on the **Outputs** tab. This property has a dependent property:
  - **Check intact**. Select this to force validation of the partial schema as the file or files are imported. Note that this can degrade performance.
- **Record delimiter string**. Specify the string at the end of each record. Enter one or more characters. This is mutually exclusive with Record delimiter, which is the default, and record type and record prefix.
- **Record delimiter**. Specify the single character at the end of each record. Type a character or select one of the following:
  - UNIX Newline (the default)
  - null

    (To specify a DOS newline, use the Record delimiter string property set to "\R\N" or choose **Format as** > **DOS line terminator** from the shortcut menu.)

    Record delimiter is mutually exclusive with Record delimiter string, Record prefix, and record type.
- **Record length**. Select Fixed where fixed length fields are being read. InfoSphere DataStage calculates the appropriate length for the record. Alternatively specify the length of fixed records as number of bytes. This is not used by default (default files are comma-delimited).

- **Record Prefix**. Specifies that a variable-length record is prefixed by a 1-, 2-, or 4-byte length prefix. It is set to 1 by default. This is mutually exclusive with Record delimiter, which is the default, and record delimiter string and record type.
- **Record type**. Specifies that data consists of variable-length blocked records (varying) or implicit records (implicit). If you choose the implicit property, data is written as a stream with no explicit record boundaries. The end of the record is inferred when all of the columns defined by the schema have been parsed. The varying property allows you to specify one of the following IBM blocked or spanned formats: V, VB, VS, VBS, or VR.

  This property is mutually exclusive with Record length, Record delimiter, Record delimiter string, and Record prefix and by default is not used.

Field Defaults

Defines default properties for columns read from the file or files. These are applied to all columns, but can be overridden for individual columns from the Columns tab using the Edit Column Metadata dialog box. A common reason to override a property for an individual column occurs when reading comma-separated values (CSV) files. CSV files often enclose fields in quotes when the fields might contain a special character, such as the field delimiter. In this case, the **Quote** property for the columns in question should be overridden.

Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Actual field length**. Specifies the actual number of bytes to skip if the field's length equals the setting of the null field length property.
- **Delimiter**. Specifies the trailing delimiter of all fields in the record. Type an ASCII character or select one of whitespace, end, none, null, comma, or tab. InfoSphere DataStage skips the delimiter when reading.
  - **whitespace**. Whitespace characters at the end of a column are ignored, that is, are not treated as part of the column.
  - **end**. The end of a field is taken as the delimiter, that is, there is no separate delimiter. This is not the same as a setting of `None' which is used for fields with fixed-width columns.
  - **none**. No delimiter (used for fixed-width).
  - **null**. ASCII Null character is used.
  - **comma**. ASCII comma character is used.
  - **tab**. ASCII tab character is used.
- **Delimiter string**. Specify the string at the end of each field. Enter one or more characters. This is mutually exclusive with Delimiter, which is the default. For example, specifying `, ` (comma space - you do not need to enter the inverted commas) specifies each field is delimited by `, ` unless overridden for individual fields. InfoSphere DataStage skips the delimiter string when reading.
- **Null field length**. The length in bytes of a variable-length field that contains a null. When a variable-length field is read, a length of null field length in the source field indicates that it contains a null. This property is mutually exclusive with null field value.
- **Null field value**. Specifies the value given to a null field if the source is set to null. Can be a number, string, or C-type 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 \x*hh*, where each *h* is a hexadecimal digit 0 - F. You must use this form to encode non-printable byte values.

  This property is mutually exclusive with Null field length and Actual length. For a fixed width data representation, you can use Pad char (from the general section of Type defaults) to specify a repeated trailing character if the value you specify is shorter than the fixed width of the field.

  You can specify a list of null values that a column could contain that represent null. To do this you specify a separator character in the dependent **Null field value separator** property, and then use this separator to delimit the null values in the **Null field value** property. For example, if you set **Null field**

value separator to contain the slash character (/), then you could specify NULL/null/NUL/nul to specify that any of these strings could represent a null value in this column.

- **Null field value separator**

  This is a dependent property of **Null field value**. You can specify a separator that can be used in the **Null field value** property to specify a range of values that could represent the null value. You can specify a number, string, or C-type literal escape character (as for **Null field value**) as a separator, but a single character such as a comma (,) or slash (/) character is the best choice. You must only specify a separator if you specify multiple values in **Null field value**; specifying a separator without using it will cause a runtime error.
- **Prefix bytes**. 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. InfoSphere DataStage reads the length prefix but does not include the prefix as a separate field in the data set it reads from the file.

  This property is mutually exclusive with the Delimiter, Quote, and Final Delimiter properties, which are used by default.
- **Print field**. This property is intended for use when debugging jobs. Set it to have InfoSphere DataStage produce a message for every field it reads. The message has the format:

  `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.
- **Quote**. Specifies that variable length fields are enclosed in single quotes, double quotes, or another character or pair of characters. Choose Single or Double, or enter a character. This is set to double quotes by default.

  When reading, InfoSphere DataStage ignores the leading quote character and reads all bytes up to but not including the trailing quote character.
- **Vector prefix**. For fields that are variable length vectors, specifies that a 1-, 2-, or 4-byte prefix contains the number of elements in the vector. You can override this default prefix for individual vectors.

  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 this property to indicate the prefix length. InfoSphere DataStage reads the length prefix but does not include it as a separate field in the data set. By default, the prefix length is assumed to be one byte.

Type Defaults

These are properties that apply to all columns of a specific data type unless specifically overridden at the column level. They are divided into a number of subgroups according to data type.

General

These properties apply to several data types (unless overridden at column level):

- **Byte order**. Specifies how multiple byte data types (except string and raw data types) are ordered. Choose from:
  - little-endian. The high byte is on the right.
  - big-endian. The high byte is on the left.
  - native-endian. As defined by the native format of the machine. This is the default.
- **Data Format**. Specifies the data representation format of a field. Applies to fields of all data types except string, ustring, and raw and to record, subrec or tagged fields containing at least one field that is neither string nor raw. Choose from:

- binary
- text (the default)

  A setting of binary has different meanings when applied to different data types:
- For decimals, binary means packed.
- For other numerical data types, binary means "not text".
- For dates, binary is equivalent to specifying the julian property for the date field.
- For time, binary is equivalent to 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. A binary timestamp specifies that two 32-but integers are written.

  By default data is formatted as text, as follows:
- For the date data type, text specifies that the data read, contains a text-based date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system.
- For the decimal data type: a 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. The destination string format is: [+ | -]ddd.[ddd] and any precision and scale arguments are ignored.
- For numeric fields (int8, int16, int32, uint8, uint16, uint32, sfloat, and dfloat): InfoSphere DataStage assumes that numeric fields are represented as text.
- For the time data type: text specifies that the field represents time in the text-based form %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system.
- For the timestamp data type: text specifies a text-based timestamp in the form %yyyy-%mm-%dd %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).

- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.
- **Field width**. The number of bytes in a field represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.

  If you specify neither field width nor field max width, numeric fields written 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)
- **Pad char**. This property is ignored for output links.
- **Character set**. Specifies the character set. Choose from ASCII or EBCDIC. The default is ASCII. Applies to all data types except raw and ustring and record, subrec, or tagged containing no fields other than raw or ustring.

String

These properties are applied to columns with a string data type, unless overridden at column level.

- **Export EBCDIC as ASCII**. Not relevant for output links.

- **Import ASCII as EBCDIC**. Select this to specify that ASCII characters are read as EBCDIC characters.

Decimal

These properties are applied to columns with a decimal data type unless overridden at column level.

- **Allow all zeros**. Specifies whether to treat a packed decimal column containing all zeros (which is normally illegal) as a valid representation of zero. Select Yes or No. The default is No.
- **Decimal separator**. Specify the ASCII character that acts as the decimal separator (period by default).
- **Packed**. Select an option to specify what the decimal columns contain, choose from:
  - Yes to specify that the decimal fields contain data in packed decimal format (the default). This has the following sub-properties:

    Check. Select Yes to verify that data is packed, or No to not verify.

    Signed. Select Yes to use the existing sign when reading decimal fields. Select No to write a positive sign (0xf) regardless of the fields' actual sign value.
  - No (separate) to specify that they contain unpacked decimal with a separate sign byte. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - No (zoned) to specify that they contain an unpacked decimal in either ASCII or EBCDIC text. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - No (overpunch) to specify that the field has a leading or end 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 has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
- **Precision**. Specifies the precision of a packed decimal. Enter a number.
- **Rounding**. Specifies how to round the source field to fit into the destination decimal when reading a source field to a decimal. Choose from:
  - **up (ceiling)**. Truncate source column towards positive infinity. This mode corresponds to the IEEE 754 Round Up mode. For example, 1.4 becomes 2, -1.6 becomes -1.
  - **down (floor)**. Truncate source column towards negative infinity. This mode corresponds to the IEEE 754 Round Down mode. For example, 1.6 becomes 1, -1.4 becomes -2.
  - **nearest value**. Round the source column towards the nearest representable value. This mode corresponds to the COBOL ROUNDED mode. For example, 1.4 becomes 1, 1.5 becomes 2, -1.4 becomes -1, -1.5 becomes -2.
  - **truncate towards zero**. This is the default. Discard fractional digits to the right of the right-most fractional digit supported by 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. Using this method 1.6 becomes 1, -1.6 becomes -1.
- **Scale**. Specifies the scale of a source packed decimal.

Numeric

These properties apply to integer and float fields unless overridden at column level.

- **C_format**. Perform non-default conversion of data from string data to a integer or floating-point. This property specifies a C-language format string used for reading integer or floating point strings. This is passed to *sscanf*(). For example, specifying a C-format of %x and a field width of 8 ensures that a 32-bit integer is formatted as an 8-byte hexadecimal string.
- **In_format**. Format string used for conversion of data from string to integer or floating-point data This is passed to *sscanf*(). By default, InfoSphere DataStage invokes the C *sscanf*() function to convert a

numeric field formatted as a string to either integer or floating point data. If this function does not output data in a satisfactory format, you can specify the in_format property to pass formatting arguments to *sscanf*().

- **Out_format**. This property is not relevant for output links.

Date

These properties are applied to columns with a date data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Days since**. Dates are written as a signed integer containing the number of days since the specified date. Enter a date in the form %*yyyy*-%*mm*-%*d*d or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
- **Format string**. The string format of a date. By default this is %yyyy-%mm-%dd. For details about the format, see "Date formats" on page 31.
- **Is Julian**. Select this to specify that dates are written as a numeric value containing the Julian day. A Julian day specifies the date as the number of days from 4713 BCE January 1, 12:00 hours (noon) GMT.

Time

These properties are applied to columns with a time data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Format string**. Specifies the format of columns representing time as a string. By default this is %hh-%mm-%ss. For details about the format, see "Time formats" on page 35
- **Is midnight seconds**. Select this to specify that times are written as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.

Timestamp

These properties are applied to columns with a timestamp data type unless overridden at column level.

- **Format string**. Specifies the format of a column representing a timestamp as a string. The format combines the format for date strings and time strings. See "Date formats" on page 31 and "Time formats" on page 35.

Column Import stage: Mapping tab

For the Column Import stage the **Mapping** tab allows you to specify how the output columns are derived.

The left pane shows the columns the stage is deriving from the single imported column. These are read only and cannot be modified on this tab.

The right pane shows the output columns for each link.

You can maintain the automatic mappings of the generated columns when using this stage.

Column Import stage: Reject link

You cannot change the details of a Reject link. The link uses the column definitions for the link rejecting the data records.

Using RCP With Column Import Stages

Runtime column propagation (RCP) allows InfoSphere DataStage to be flexible about the columns you define in a job. If RCP is enabled for a project, you can just define the columns you are interested in

using in a job, but ask InfoSphere DataStage to propagate the other columns through the various stages. So such columns can be extracted from the data source and end up on your data target without explicitly being operated on in between.

Columns you are importing do not have inherent column definitions, and so InfoSphere DataStage cannot always tell where there are extra columns that need propagating. You can only use RCP on Column Import stages if you have used the Schema File property to specify a schema which describes all the columns in the column. You need to specify the same schema file for any similar stages in the job where you want to propagate columns. Stages that will require a schema file are:

- Sequential File
- File Set
- External Source
- External Target
- Column Import
- Column Export

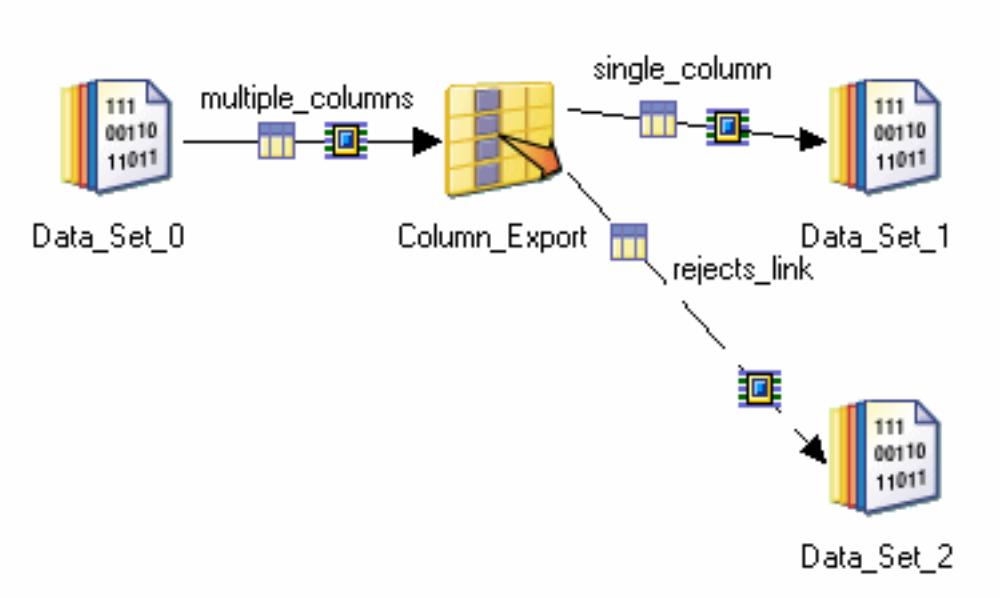
Column Export stage

The Column Export stage is a restructure stage. It can have a single input link, a single output link and a single rejects link.

The Column Export stage exports data from a number of columns of different data types into a single column of data type ustring, string, or binary. It is the complementary stage to Column Import (see "Column Import stage" on page 427).

The input data column definitions determine the order in which the columns are exported to the single output column. Information about how the single column being exported is delimited is given in the Formats tab of the Input page. You can optionally save reject records, that is, records whose export was rejected.

In addition to exporting a column you can also pass other columns straight through the stage. So, for example, you could pass a key column straight through.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Examples

This section gives examples of input and output data from a Column Export stage to give you a better idea of how the stage works.

In this example the Column Export stage extracts data from three input columns and outputs two of them in a single column of type string and passes the other through. The example assumes that the job is running sequentially. The column definitions for the input data set are as follows:

Table 79. Column definitions

Column name	SQL type	Length	Scale
value	Decimal	5	2
SN	SmallInt	1	
code	Char	4	

The following are the rows from the input data set:

Table 80. Input data set

value	SN	code
000.00	0	aaaa
001.00	1	bbbb
002.00	2	cccc
003.00	3	dddd
004.00	4	eeee
005.00	5	ffff
006.00	6	gggg
007.00	7	hhhh
008.00	8	iiii
009.00	9	jjjj

The import table definition is supplied on the **Output Page Columns** tab. For the example, the definition would be:

Table 81. Column definitions for output

Column name	SQL type	Length	Scale
code	Char	4	
exported_col	Char		

You have to give InfoSphere DataStage information about how to delimit the exported data when it combines it into a single column. This is done on the Input page **Format Tab**. For this example, you specify a data format of text, a Field Delimiter of comma, and a Quote type of double.

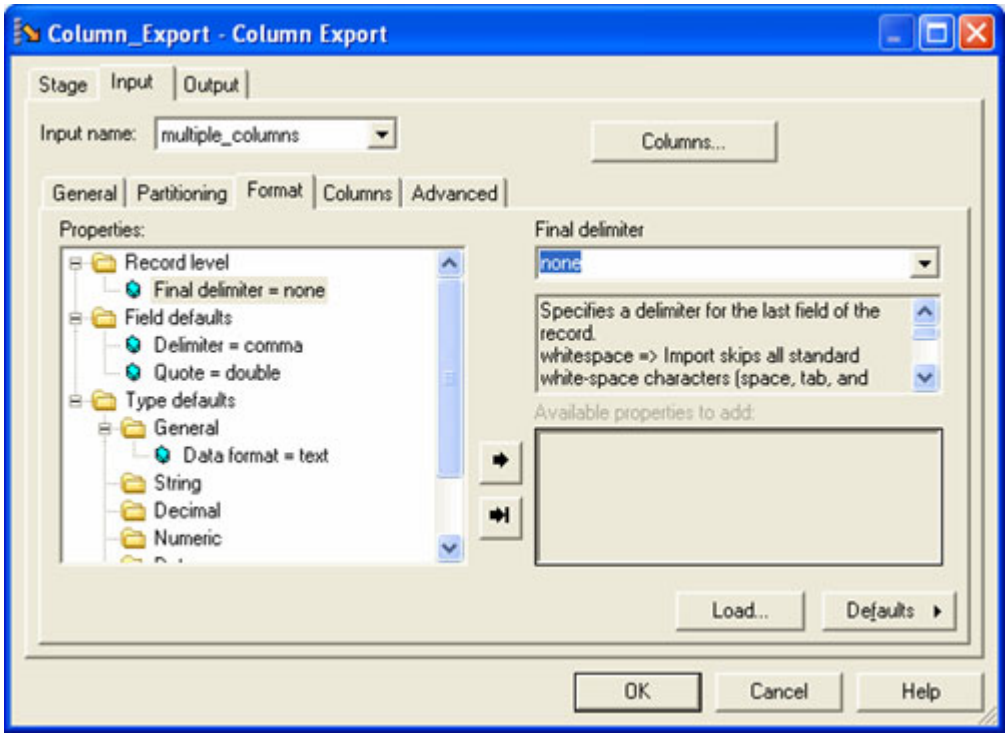


Figure 28. Format tab

.

The Properties of the Column Export stage are set as follows:

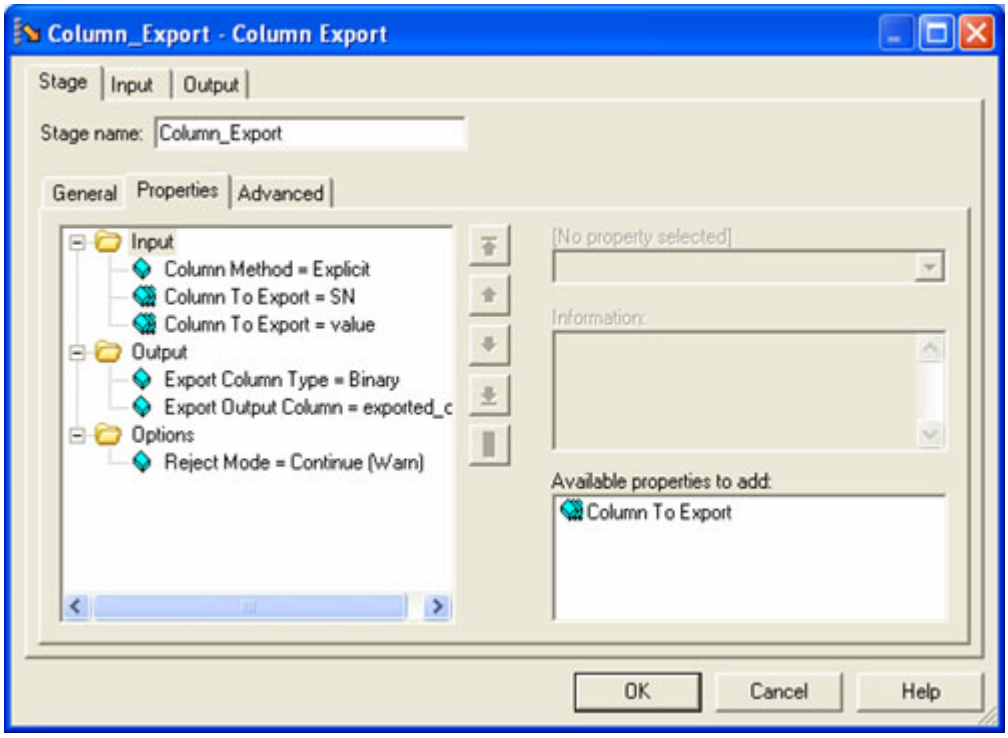


Figure 29. Properties tab

The output data set will be:

exported_co
code

("20 0 0 0.00" , "0")
aaaa

("20 0 0 1.00" , "1")
bbbb

("20 0 0 2.00" , "2")
cccc

("20 0 0 3.00" , "3")
dddd

("20 0 0 4.00" , "4")
eeee

("20 0 0 5.00" , "5")
ffff

("20 0 0 6.00" , "6")
gggg

("20 0 0 7.00" , "7")
hhhh

("20 0 0 8.00" , "8")
iiii

("20 0 0 9.00" , "9")
jjjj

Column Export stage: fast path

This section specifies the minimum steps to take to get a Column Export stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Column Export stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

1. Go to the Stage page **Partitioning Tab**, under the Input category.
2. Choose the Column method, this is Explicit by default, meaning you specify explicitly choose input columns as sources. The alternative is to specify a schema file.
   -
   -
3. If you are using the Explicit method, choose the input column(s) to carry your exported input column. Repeat the Column to Export property to specify all the columns you need.
4. If you are using the Schema File method, specify the schema file that gives the output column details. Under the Output category:
   a. Choose the Export Column Type. This is Binary by default, but you can also choose VarChar or Unicode VarChar. This specifies the format of the column you are exporting to.
   b. Specify the column you are exporting to in the Export Output Column property.

5. In the Input page **Format Tab** specify the format of the column you are exporting. This informs InfoSphere DataStage about delimiters and enables it to combine multiple columns into a single column with delimiters.
6. In the Output page **Mapping Tab** check that the input columns are mapping onto the output columns as you expect. The mapping is carried out according to what you specified in the Properties tab.

Column Export stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Column Export stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 82. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Export Output Column	Output Column	N/A	Y	N	N/A
Options/Export Column Type	Binary/ VarChar/Unicode VarChar	Binary	N	N	N/A
Options/Reject Mode	Continue (warn) /Output	Continue	N	N	N/A
Options/Column to Export	Input Column	N/A	N	Y	N/A
Options/Schema File	Pathname	N/A	N	N	N/A

Column Export stage: Options category:
Export output column

Specifies the name of the single column to which the input column or columns are exported.

Export column type

Specify either binary, VarChar (string), or Unicode VarChar (ustring).

Reject mode

The values of this property specify the following actions:

- **Output.** The stage continues when it encounters a reject record and writes the record to the rejects link.
- **Continue(warn).** The stage is to continue but report failures to the log file.

Column to export

Specifies an input column the stage extracts data from. The format properties for this column can be set on the Format tab of the Input page. Repeat the property to specify multiple input columns. You can use

the Column Selection dialog box to select multiple columns at once if required. The order of the Columns to Export that you specify should match the order on the Columns tab. If it does not, the order on the Columns tab overrides the order of the properties.

Schema file

Instead of specifying the source data details via input column definitions, you can use a schema file (note, however, that if you have defined columns on the Columns tab, you should ensure these match the schema file). Type in a pathname or browse for a schema file.

Column Export stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Column Export stage: Input page

The Input page allows you to specify details about the incoming data sets. The Column Export stage expects one incoming data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being exported. The Format tab allows you to specify details how data in the column you are exporting will be formatted. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Column Export stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Column Export stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is exported. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Column Export stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Column Export stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Column Export stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Column Export stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Column Export stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag columns.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Column Export stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being exported. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Column Export stage: Format tab

The Format tab allows you to supply information about the format of the column you are exporting. You use it in the same way as you would to describe the format of a flat file you were writing. The tab has a similar format to the Properties tab.

Select a property type from the main tree then add the properties you want to set to the tree structure by clicking on them in the **Available properties to add** window. You can then set a value for that property in the Property Value box. Pop up help for each of the available properties appears if you over the mouse pointer over it.

This description uses the terms "record" and "row" and "field" and "column" interchangeably.

The following sections list the Property types and properties available for each type.

Record level

These properties define details about how data records are formatted in the flat file. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

Fill char

Specify an ASCII character or a value in the range 0 to 255. You can also choose Space or Null from a drop-down list. This character is used to fill any gaps in a written record caused by column positioning properties. Set to 0 by default (which is the NULL character). For example, to set it to space you could also type in the space character or enter 32. Note that this value is restricted to one byte, so you cannot specify a multi-byte Unicode character.

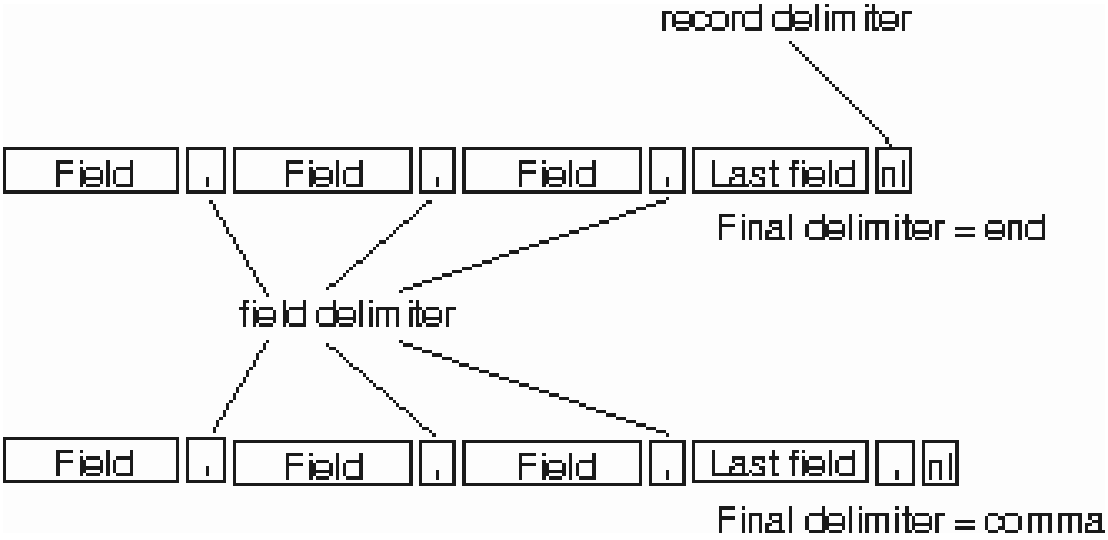
Final delimiter string

Specify a string to be written after the last column of a record in place of the column delimiter. Enter one or more characters, this precedes the record delimiter if one is used. Mutually exclusive with Final delimiter, which is the default. For example, if you set Delimiter to comma and Final delimiter string to `, ` (comma space - you do not need to enter the inverted commas) all fields are delimited by a comma, except the final field, which is delimited by a comma followed by an ASCII space character.

Final delimiter

Specify a single character to be written after the last column of a record in place of the field delimiter. Type a character or select one of whitespace, end, none, null, tab, or comma. See the following diagram for an illustration.

- **whitespace**. The last column of each record will not include any trailing white spaces found at the end of the record.
- **end**. The last column of each record does not include the field delimiter. This is the default setting.
- **none**. The last column of each record does not have a delimiter; used for fixed-width fields.
- **null**. The last column of each record is delimited by the ASCII null character.
- **comma**. The last column of each record is delimited by the ASCII comma character.
- **tab**. The last column of each record is delimited by the ASCII tab character.



When writing, a space is now inserted after every field except the last in the record. Previously, a space was inserted after every field including the last. (If you want to revert to the pre-release 7.5 behavior of inserting a space after the last field, set the APT_FINAL_DELIM_COMPATIBLE environment variable.

Intact

The intact property specifies an identifier of a partial schema. A partial schema specifies that only the column(s) named in the schema can be modified by the stage. All other columns in the row are passed through unmodified. The file containing the partial schema is specified in the **Schema File** property on the **Properties** tab. This property has a dependent property, Check intact, but this is not relevant to input links.

Record delimiter string

Specify a string to be written at the end of each record. Enter one or more characters. This is mutually exclusive with Record delimiter, which is the default, record type and record prefix.

Record delimiter

Specify a single character to be written at the end of each record. Type a character or select one of the following:

- UNIX Newline (the default)
- null

(To implement a DOS newline, use the Record delimiter string property set to "\R\N" or choose**Format as** > **DOS line terminator** from the shortcut menu.)

Note: Record delimiter is mutually exclusive with Record delimiter string, Record prefix, and Record type.

Record length

Select **Fixed** where fixed length fields are being written. InfoSphere DataStage calculates the appropriate length for the record. Alternatively specify the length of fixed records as number of bytes. This is not used by default (default files are comma-delimited). The record is padded to the specified length with either zeros or the fill character if one has been specified.

Record Prefix

Specifies that a variable-length record is prefixed by a 1-, 2-, or 4-byte length prefix. It is set to 1 by default. This is mutually exclusive with Record delimiter, which is the default, and record delimiter string and record type.

Record type

Specifies that data consists of variable-length blocked records (varying) or implicit records (implicit). If you choose the implicit property, data is written as a stream with no explicit record boundaries. The end of the record is inferred when all of the columns defined by the schema have been parsed. The varying property allows you to specify one of the following IBM blocked or spanned formats: **V**, **VB**, **VS**, **VBS**, or **VR**.

This property is mutually exclusive with Record length, Record delimiter, Record delimiter string, and Record prefix and by default is not used.

Field defaults

Defines default properties for columns written to the file or files. These are applied to all columns written, but can be overridden for individual columns from the **Columns** tab using the Edit Column Metadata dialog box. Where you can enter a character, this can usually be an ASCII character or a multi-byte Unicode character (if you have NLS enabled). The available properties are:

- **Actual field length**. Specifies the number of bytes to fill with the Fill character when a field is identified as null. When InfoSphere DataStage identifies a null field, it will write a field of this length full of Fill characters. This is mutually exclusive with Null field value.
- **Delimiter**. Specifies the trailing delimiter of all fields in the record. Type an ASCII character or select one of whitespace, end, none, null, comma, or tab.
  - **whitespace**. Whitespace characters at the end of a column are ignored, that is, are not treated as part of the column.
  - **end**. The end of a field is taken as the delimiter, that is, there is no separate delimiter. This is not the same as a setting of `None' which is used for fields with fixed-width columns.
  - **none**. No delimiter (used for fixed-width).
  - **null**. ASCII Null character is used.
  - **comma**. ASCII comma character is used.
  - **tab**. ASCII tab character is used.
- **Delimiter string**. Specify a string to be written at the end of each field. Enter one or more characters. This is mutually exclusive with Delimiter, which is the default. For example, specifying `, ` (comma space - you do not need to enter the inverted commas) would have each field delimited by `, ` unless overridden for individual fields.
- **Null field length**. The length in bytes of a variable-length field that contains a null. When a variable-length field is written, InfoSphere DataStage writes a length value of null field length if the field contains a null. This property is mutually exclusive with null field value.
- **Null field value**. Specifies the value written to null field if the source is set to null. Can be a number, string, or C-type 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 \x*hh*, where each *h* is a hexadecimal digit 0 - F. You must use this form to encode non-printable byte values.

This property is mutually exclusive with Null field length and Actual length. For a fixed width data representation, you can use Pad char (from the general section of Type defaults) to specify a repeated trailing character if the value you specify is shorter than the fixed width of the field.

Null field value has a sub property named Null field value separator. This is intended for output data, and should be ignored on Format tabs belonging to input links.

- **Prefix bytes**. Specifies that each column in the data file is prefixed by 1, 2, or 4 bytes containing, as a binary value, either the column's length or the tag value for a tagged 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. InfoSphere DataStage inserts the prefix before each field.

  This property is mutually exclusive with the Delimiter, Quote, and Final Delimiter properties, which are used by default.
- **Print field**. This property is not relevant for input links.
- **Quote**. Specifies that variable length fields are enclosed in single quotes, double quotes, or another character or pair of characters. Choose **Single** or **Double**, or enter a character. This is set to double quotes by default.

  When writing, InfoSphere DataStage inserts the leading quote character, the data, and a trailing quote character. Quote characters are not counted as part of a field's length.
- **Vector prefix**. For fields that are variable length vectors, specifies a 1-, 2-, or 4-byte prefix containing the number of elements in the vector. You can override this default prefix for individual vectors.

  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 this property to indicate the prefix length. InfoSphere DataStage inserts the element count as a prefix of each variable-length vector field. By default, the prefix length is assumed to be one byte.

Type defaults

These are properties that apply to all columns of a specific data type unless specifically overridden at the column level. They are divided into a number of subgroups according to data type.

General

These properties apply to several data types (unless overridden at column level):

- **Byte order**. Specifies how multiple byte data types (except string and raw data types) are ordered. Choose from:
  - little-endian. The high byte is on the right.
  - big-endian. The high byte is on the left.
  - native-endian. As defined by the native format of the machine. This is the default.
- **Data Format**. Specifies the data representation format of a field. Applies to fields of all data types except string, ustring, and raw and to record, subrec or tagged fields containing at least one field that is neither string nor raw. Choose from:
  - binary
  - text (the default)

    A setting of binary has different meanings when applied to different data types:
  - For decimals, binary means packed.
  - For other numerical data types, binary means "not text".
  - For dates, binary is equivalent to specifying the julian property for the date field.
  - For time, binary is equivalent to 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. A binary timestamp specifies that two 32-but integers are written.

By default data is formatted as text, as follows:

– For the date data type, text specifies that the data to be written contains a text-based date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
– For the decimal data type: a 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. The destination string format is: [+ | -]ddd.[ddd] and any precision and scale arguments are ignored.
– For *numeric* fields (int8, int16, int32, uint8, uint16, uint32, sfloat, and dfloat): InfoSphere DataStage assumes that numeric fields are represented as text.
– For the time data type: text specifies that the field represents time in the text-based form %hh:%nn:%ss or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
– For the *timestamp* data type: text specifies a text-based timestamp in the form *%yyyy-%mm-%dd %hh:%nn:%ss* or in the default date format if you have defined a new one on an NLS system.

- **Field max width**. The maximum number of bytes in a column represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using a fixed-width character set, you can calculate the length exactly. If you are using variable-length character set, calculate an adequate maximum width for your fields. Applies to fields of all data types except date, time, timestamp, and raw; and record, subrec, or tagged if they contain at least one field of this type.
- **Field width**. The number of bytes in a field represented as a string. Enter a number. This is useful where you are storing numbers as text. If you are using 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; and record, subrec, or tagged if they contain at least one field of this type.

  If you specify neither field width nor field max width, numeric fields written 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)

  Important: If you are using Unicode character columns, you must calculate the field length in bytes and specify that value in the Field Width column property.
- **Pad char**. Specifies the pad character used when strings or numeric values are written to an external string representation. Enter a character (single-byte for strings, can be multi-byte for ustrings) or choose null or space. The pad character is used when the external string representation is larger than required to hold the written field. In this case, the external string is filled with the pad character to its full length. Space is the default. Applies to string, ustring, and numeric data types and record, subrec, or tagged types if they contain at least one field of this type.
- **Character set**. Specifies the character set. Choose from ASCII or EBCDIC. The default is ASCII. Applies to all data types except raw and ustring and record, subrec, or tagged containing no fields other than raw or ustring.

String

These properties are applied to columns with a string data type, unless overridden at column level.

- **Export EBCDIC as ASCII**. Select this to specify that EBCDIC characters are written as ASCII characters. Applies to fields of the string data type and record, subrec, or tagged fields if they contain at least one field of this type.
- **Import ASCII as EBCDIC**. Not relevant for input links.

Decimal

These properties are applied to columns with a decimal data type unless overridden at column level.

- **Allow all zeros**. Specifies whether to treat a packed decimal column containing all zeros (which is normally illegal) as a valid representation of zero. Select **Yes** or **No**. The default is **No**.
- **Decimal separator.** Specify the ASCII character that acts as the decimal separator (period by default).
- **Packed**. Select an option to specify what the decimal columns contain, choose from:
  - **Yes** to specify that the decimal columns contain data in packed decimal format (the default). This has the following sub-properties:

    Check. Select **Yes** to verify that data is packed, or **No** to not verify.

    Signed. Select **Yes** to use the existing sign when writing decimal columns. Select **No** to write a positive sign (0xf) regardless of the columns' actual sign value.
  - **No (separate)** to specify that they contain unpacked decimal with a separate sign byte. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - **No (zoned)** to specify that they contain an unpacked decimal in either ASCII or EBCDIC text. This has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
  - **No (overpunch)** to specify that the field has a leading or end 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 has the following sub-property:

    Sign Position. Choose leading or trailing as appropriate.
- **Precision**. Specifies the precision where a decimal column is written in text format. Enter a number. When a decimal is written to a string representation, InfoSphere DataStage 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. When they are defined, InfoSphere DataStage truncates or pads the source decimal to fit the size of the destination string. If you have also specified the field width property, InfoSphere DataStage truncates or pads the source decimal to fit the size specified by field width.
- **Rounding**. Specifies how to round a decimal column when writing it. Choose from:
  - up (ceiling). Truncate source column towards positive infinity. This mode corresponds to the IEEE 754 Round Up mode. For example, 1.4 becomes 2, -1.6 becomes -1.
  - down (floor). Truncate source column towards negative infinity. This mode corresponds to the IEEE 754 Round Down mode. For example, 1.6 becomes 1, -1.4 becomes -2.
  - nearest value. Round the source column towards the nearest representable value. This mode corresponds to the COBOL ROUNDED mode. For example, 1.4 becomes 1, 1.5 becomes 2, -1.4 becomes -1, -1.5 becomes -2.
  - truncate towards zero. This is the default. Discard fractional digits to the right of the right-most fractional digit supported by 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. Using this method 1.6 becomes 1, -1.6 becomes -1.
- **Scale**. Specifies how to round a source decimal when its precision and scale are greater than those of the destination. By default, when the InfoSphere DataStage writes 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, InfoSphere DataStage truncates or pads the source decimal to fit the size of the destination string. If you have also specified the field width property, InfoSphere DataStage truncates or pads the source decimal to fit the size specified by field width.

Numeric

These properties apply to integer and float fields unless overridden at column level.

- **C_format**. Perform non-default conversion of data from integer or floating-point data to a string. This property specifies a C-language format string used for writing integer or floating point strings. This is passed to *sprintf*(). For example, specifying a C-format of %x and a field width of 8 ensures that integers are written as 8-byte hexadecimal strings.
- **In_format**. This property is not relevant for input links..
- **Out_format**. Format string used for conversion of data from integer or floating-point data to a string. This is passed to *sprintf*(). By default, InfoSphere 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 to pass formatting arguments to *sprintf*().

Date

These properties are applied to columns with a date data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Days since**. Dates are written as a signed integer containing the number of days since the specified date. Enter a date in the form *%yyyy-%mm-%dd* or in the default date format if you have defined a new one on an NLS system (see *IBM InfoSphere DataStage and QualityStage Globalization Guide*).
- **Format string**. The string format of a date. By default this is %yyyy-%mm-%dd. For details about the format, see "Date formats" on page 31.
- **Is Julian**. Select this to specify that dates are written as a numeric value containing the Julian day. A Julian day specifies the date as the number of days from 4713 BCE January 1, 12:00 hours (noon) GMT.

Time

These properties are applied to columns with a time data type unless overridden at column level. All of these are incompatible with a Data Format setting of Text.

- **Format string**. Specifies the format of columns representing time as a string. For details about the format, see "Time formats" on page 35
- **Is midnight seconds**. Select this to specify that times are written as a binary 32-bit integer containing the number of seconds elapsed from the previous midnight.

Timestamp

These properties are applied to columns with a timestamp data type unless overridden at column level.

- **Format string**. Specifies the format of a column representing a timestamp as a string. Defaults to *%yyyy-%mm-%dd %hh:%nn:%ss*. The format combines the format for date strings and time strings. See "Date formats" on page 31 and "Time formats" on page 35.

Column Export stage: Output page: The Output page allows you to specify details about data output from the Column Export stage. The Column Export stage can have only one output link, but can also have a reject link carrying records that have been rejected.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Column Export stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output links.

Details about Column Export stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Column Export stage: Mapping tab

For the Column Export stage the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them or how they are generated.

The left pane shows the input columns plus the composite column that the stage exports the specified input columns to. These are read only and cannot be modified on this tab.

The right pane shows the output columns for each link. This has a **Derivations** field where you can specify how the column is derived.You can fill it in by dragging input columns over, or by using the Auto-match facility.

The remaining columns are all being exported to comp_col, which is the specified Export Column. You could also pass the original columns through the stage, if required.

Column Export stage: Reject link

You cannot change the details of a Reject link. The link uses the column definitions for the link rejecting the data rows. Rows will be rejected if they do not match the expected schema.

Using RCP with Column Export stages

Runtime column propagation (RCP) allows InfoSphere DataStage to be flexible about the columns you define in a job. If RCP is enabled for a project, you can just define the columns you are interested in using in a job, but ask InfoSphere DataStage to propagate the other columns through the various stages. So such columns can be extracted from the data source and end up on your data target without explicitly being operated on in between.

You can only use RCP on Column Export stages if you have used the Schema File property (see "Schema File") to specify a schema which describes all the columns in the column. You need to specify the same schema file for any similar stages in the job where you want to propagate columns. Stages that will require a schema file are:

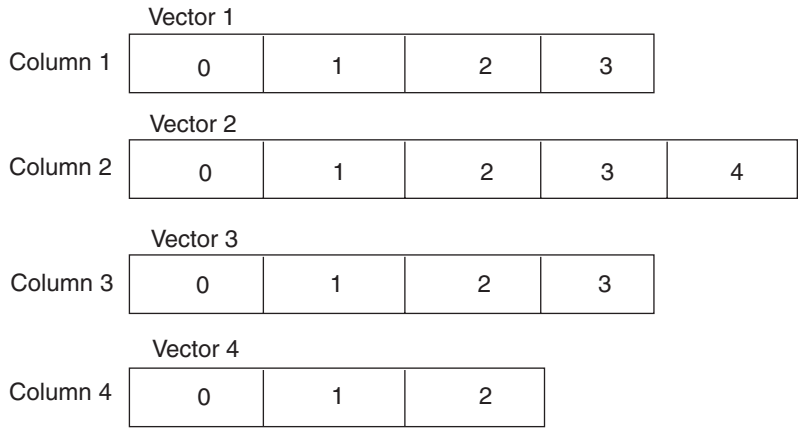
- Sequential File
- File Set
- External Source
- External Target
- Column Import
- Column Export

Make Subrecord stage

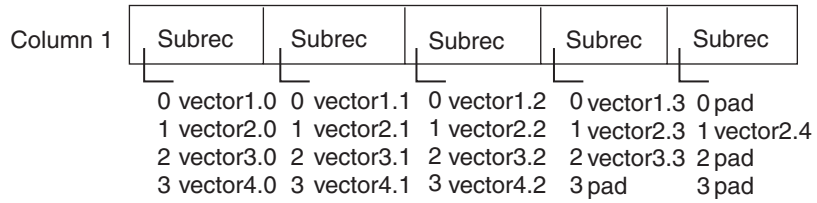
The Make Subrecord stage is a restructure stage. It can have a single input link and a single output link.

The Make Subrecord stage combines specified vectors in an input data set into a vector of subrecords whose columns have the names and data types of the original vectors. You specify the vector columns to be made into a vector of subrecords and the name of the new subrecord. See "Complex Data Types" for an explanation of vectors and subrecords.

input data



output data

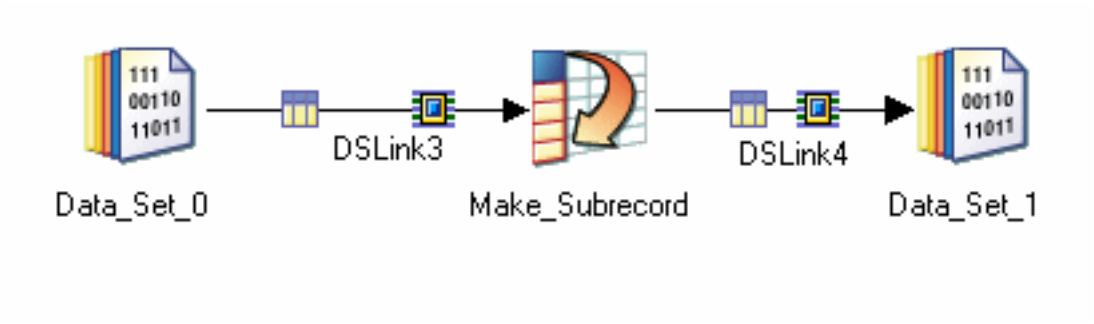


The Split Subrecord stage performs the inverse operation. See "Split Subrecord Stage."

The length of the subrecord vector created by this operator equals the length of the longest vector column from which it is created. If a variable-length vector column 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 columns and the corresponding type-dependent value for non-nullable columns. When the Make Subrecord stage encounters mismatched vector lengths, it warns you by writing to the job log.

You can also use the stage to make a simple subrecord rather than a vector of subrecords. If your input columns are simple data types rather than vectors, they will be used to build a vector of subrecords of length 1 - effectively a simple subrecord.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Examples

This section gives examples of input and output data from a Make Subrecord stage to give you a better idea of how the stage works.

In this example the Make Subrecord stage extracts data from four input columns, three of which carry vectors. The data is output in two columns, one carrying the vectors in a subrecord, and the non-vector

column being passed through the stage. The example assumes that the job is running sequentially. The column definitions for the input data set are as follows

Table 83. Column definitions

Column name	SQL type
key	Char
acol	Integer
bcol	Char
ccol	Char

.

The following are the rows from the input data set (superscripts represents the vector index):

Table 84. Input data set

	Key	acol	bcol	ccol
row	A	$12^0 13^1 4^2 64^3$	$Wills^0 wombat^1 bill^2 william$	$D^0 0^1$
row	B	$22^0 6^1 4^2 21^3$	$Robin^0 Dally^1 Rob^2 RD^3$	$G^0 A^1$
row	C	$76^0 0^1 52^2 2^3$	$Beth^0 Betany^1 Bethany^2 Bets^3$	$B^0 7^1$
row	D	$4^0 6^1 81^2 0^3$	$Heathcliff^0 HC^1 Hchop^2 Horror^3$	$A^0 1^1$
row	E	$2^0 4^1 6^2 8^3$	$Chaz^0 Swot^1 Chazlet^2 Twerp^3$	$C^0 H^1$
row	F	$18^0 8^1 5^2 8^3$	$kayser^0 Cuddles^1 KB^2 Ibn\ Kayeed^3$	$M^0 1^1$
row	G	$12^0 10^1 6^2 1^3$	$Jayne^0 Jane^1 J^2 JD^3$	$F^0 2^1$
row	H	$12^0 0^1 6^2 43^3$	$Ann^0 Anne^1 AK^2 AJK^3$	$H^0 E^1$
row	I	$3^0 0^1 7^2 82^3$	$Kath^0 Cath^1 Catherine^2 Katy^3$	$^0 H^1$
row	J	$5^0 0^1 7^2 02^3$	$upert^0 Rupe^1 Woopert^2 puss^3$	$B^0 C^1$

The stage outputs the subrecord it builds from the input data in a single column called parent. The column called key will be output separately. The output column definitions are as follows:

Table 85. Output column definitions

Column name	SQL type
key	Char
parent	Char

The Properties of the Make Subrecord stage are set as follows:

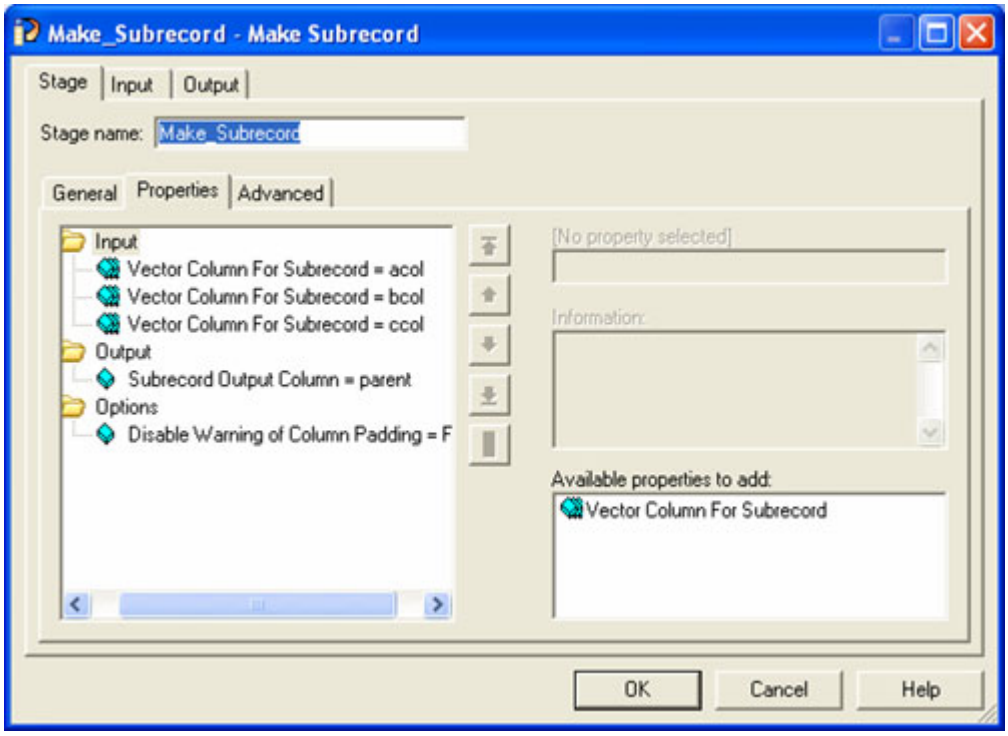


Figure 30. Properties tab

The output data set will be:

Table 86. Output data set

	Key	**Parent**			
		0	*1*	*2*	*3*
row	A	12	13	4	64
		Will	wombat	bill	William
		D	0	pad	pad
	B	22	6	4	21
		Robin	Dally	Rob	RD
		G	A	pad	pad
	C	76	0	52	2
		Beth	Betany	Bethany	Bets
		B	7	pad	pad
	D	4	6	81	0
		Heathcliff	HC	Hchop	Horror
		A	1	pad	pad
	E	2	4	6	8
		Chaz	Swot	Chazlet	Twerp
		C	H	pad	pad
	F	18	8	5	8
		Kayser	Cuddles	KB	Ibn Kayeed
		M	1	pad	pad

Table 86. Output data set (continued)

	Key	Parent			
	G	12	10	6	43
		Jayne	Jane	J	JD
		F	2	pad	pad
	H	12	0	6	43
		Ann	Anne	AK	AJK
		H	E	pad	pad
	I	3	0	7	82
		Kath	Cath	Catherine	Katy
		C	H	pad	pad
	J	5	0	7	02
		Rupert	Rupe	Woopert	puss
		B	C	pad	pad

Make Subrecord stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Make Subrecord stages in a job. This section specifies the minimum steps to take to get a Make Subrecord stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Make Subrecord stage:

- In the **Outputs** Stage **Properties Tab**, under the Input category:
  - Specify the vector column to combine into the subrecord, repeat the property to specify multiple vector columns.

    Under the Output category:
  - Specify the Subrecord output column.

Make Subrecord stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Make Subrecord stage: Properties tab

The Properties tab allows you to specify properties that determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 87. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Subrecord Output Column	Output Column	N/A	Y	N	N/A
Options/Vector Column for Subrecord	Input Column	N/A	N	Y	Key
Options/Disable Warning of Column Padding	True/False	False	N	N	N/A

Make Subrecord stage: Input category:
Vector column for subrecord

Specify the name of the column to include in the subrecord. You can specify multiple columns to be combined into a subrecord. For each column, specify the property followed by the name of the column to include. You can use the Column Selection dialog box to select multiple columns at once if required.

Make Subrecord stage: Output category:
Subrecord output column

Specify the name of the subrecord into which you want to combine the columns specified by the Vector Column for Subrecord property.

Make Subrecord stage: Options category:
Disable warning of column padding

When the stage combines vectors of unequal length, it pads columns and displays a message to this effect. Optionally specify this property to disable display of the message.

Make Subrecord stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Make Subrecord stage: Input page

The Input page allows you to specify details about the incoming data sets. The Make Subrecord stage expects one incoming data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being converted. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Make Subrecord stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Make Subrecord stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is converted. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode.

If the Make Subrecord stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Make Subrecord stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Make Subrecord stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Make Subrecord stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method of the Make Subrecord stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag columns.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place. This is the default partitioning method for the Make Subrecord stage.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Make Subrecord stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being converted. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default for the default Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Make Subrecord stage: Output page

The Output page allows you to specify details about data output from the Make Subrecord stage. The Make Subrecord stage can have only one output link.

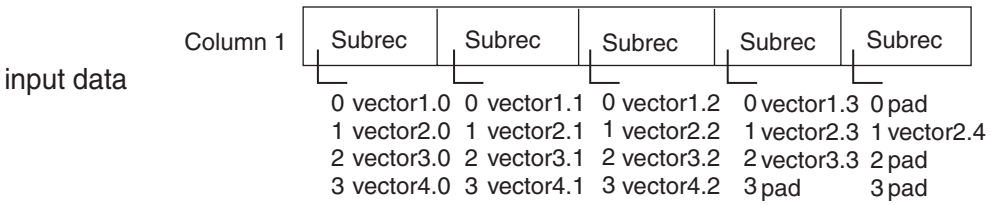
The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

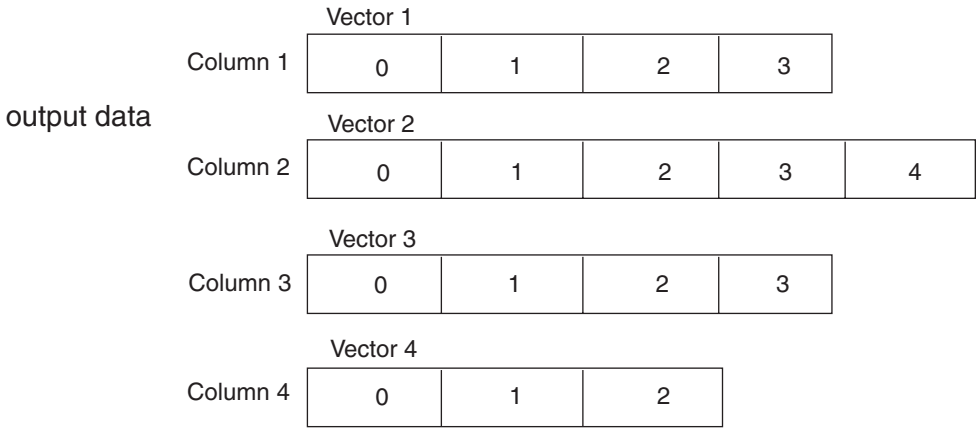
See "Stage Editors," for a general description of the tabs.

Split Subrecord Stage

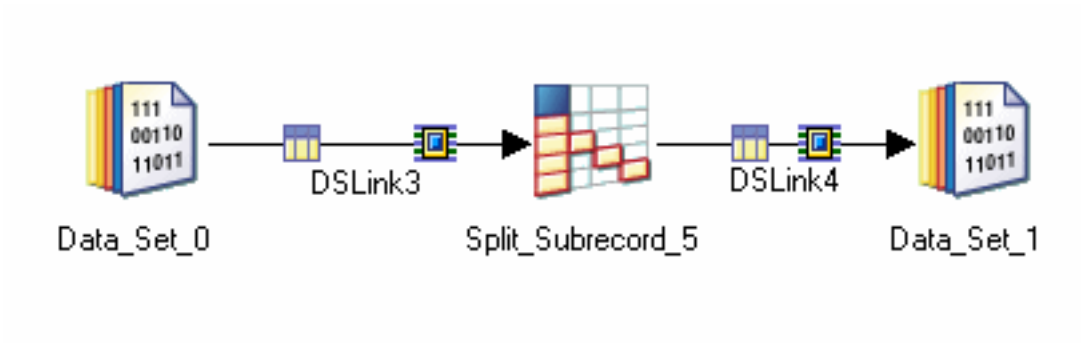
The Split Subrecord stage separates an input subrecord field into a set of top-level vector columns. It can have a single input link and a single output link.

The stage creates one new vector column for each element of the original subrecord. That is, each top-level vector column that is created has the same number of elements as the subrecord from which it was created. The stage outputs columns of the same name and data type as those of the columns that comprise the subrecord.





The Make Subrecord stage performs the inverse operation (see, "Make Subrecord Stage.")



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Examples

This section gives examples of input and output data from a Split Subrecord stage to give you a better idea of how the stage works.

In this example the Split Subrecord stage extracts data from a subrecord containing three vectors. The data is output in fours column, three carrying the vectors from the subrecord, plus another column which is passed through the stage. The example assumes that the job is running sequentially. The column definitions for the input data set are as follows:

Table 88. Column definitions

Column name	**SQL type**
key	Char
parent	Char

The following are the rows from the input data set (superscripts represents the vector index):

Table 89. Input data set

	Key	**Parent**			
		0	*1*	*2*	*3*
row	A	12	13	4	64
		Will	wombat	bill	William
		D	0	pad	pad
	B	22	6	4	21
		Robin	Dally	Rob	RD
		G	A	pad	pad
	C	76	0	52	2
		Beth	Betany	Bethany	Bets
		B	7	pad	pad
	D	4	6	81	0
		Heathcliff	HC	Hchop	Horror
		A	1	pad	pad
	E	2	4	6	8
		Chaz	Swot	Chazlet	Twerp
		C	H	pad	pad
	F	18	8	5	8
		Kayser	Cuddles	KB	Ibn Kayeed
		M	1	pad	pad
	G	12	10	6	43
		Jayne	Jane	J	JD
		F	2	pad	pad
	H	12	0	6	43
		Ann	Anne	AK	AJK
		H	E	pad	pad
	I	3	0	7	82
		Kath	Cath	Catherine	Katy
		C	H	pad	pad
	J	5	0	7	02
		Rupert	Rupe	Woopert	puss

Table 89. Input data set (continued)

	Key	Parent			
		B	C	pad	pad

The stage outputs the data it extracts from the subrecord in three separate columns each carrying a vector. The column called key will be output separately. The output column definitions are as follows:

Table 90. Output column definitions

Column name	SQL type
key	Char
acol	Integer
bcol	Char
ccol	Char

The Subrecord column property is set to 'parent' in the Properties tab.

The output data set will be (superscripts represents the vector index):

Table 91. Output data set

	Key	acol	bcol	ccol
row	A	$12^0 13^1 4^2 64^3$	$Wills^0 wombat^1 bill^2 william$	$D^0 0^1$
row	B	$22^0 6^1 4^2 21^3$	$Robin^0 Dally^1 Rob^2 RD^3$	$G^0 A^1$
row	C	$76^0 0^1 52^2 2^3$	$Beth^0 Betany^1 Bethany^2 Bets^3$	$B^0 7^1$
row	D	$4^0 6^1 81^2 0^3$	$Heathcliff^0 HC^1 Hchop^2 Horror^3$	$A^0 1^1$
row	E	$2^0 4^1 6^2 8^3$	$Chaz^0 Swot^1 Chazlet^2 Twerp^3$	$C^0 H^1$
row	F	$18^0 8^1 5^2 8^3$	$kayser^0 Cuddles^1 KB^2 Ibn\ Kayeed^3$	$M^0 1^1$
row	G	$12^0 10^1 6^2 1^3$	$Jayne^0 Jane^1 J^2 JD^3$	$F^0 2^1$
row	H	$12^0 0^1 6^2 43^3$	$Ann^0 Anne^1 AK^2 AJK^3$	$H^0 E^1$
row	I	$3^0 0^1 7^2 82^3$	$Kath^0 Cath^1 Catherine^2 Katy^3$	$^0 H^1$
row	J	$5^0 0^1 7^2 02^3$	$Rupert^0 Rupe^1 Woopert^2 puss^3$	$B^0 C^1$

Split Subrecord stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Split Subrecord stages in a job. This section specifies the minimum steps to take to get a Split Subrecord stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Split Subrecord stage:

- In the Outputs Stage **Properties Tab**, under the Input category:
  - Specify the subrecord column that the stage will extract vectors from.

Split Subrecord stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Split Subrecord stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. The Split Subrecord only has one property, and you must supply a value for this.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 92. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Subrecord Column	Input Column	N/A	Y	N	N/A

Split Subrecord stage: Options category: Subrecord column

Specifies the name of the vector whose elements you want to promote to a set of similarly named top-level columns.

Split Subrecord stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Split Subrecord stage: Input page

The Input page allows you to specify details about the incoming data sets. There can be only one input to the Split Subrecord stage.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being converted. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Split Subrecord stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Split Subrecord stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is converted. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Split Subrecord stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Split Subrecord stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Split Subrecord stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Split Subrecord stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Split Subrecord stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Split Subrecord stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.

- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being converted. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Split Subrecord stage: Output page

The Output page allows you to specify details about data output from the Split Subrecord stage. The Split Subrecord stage can have only one output link.

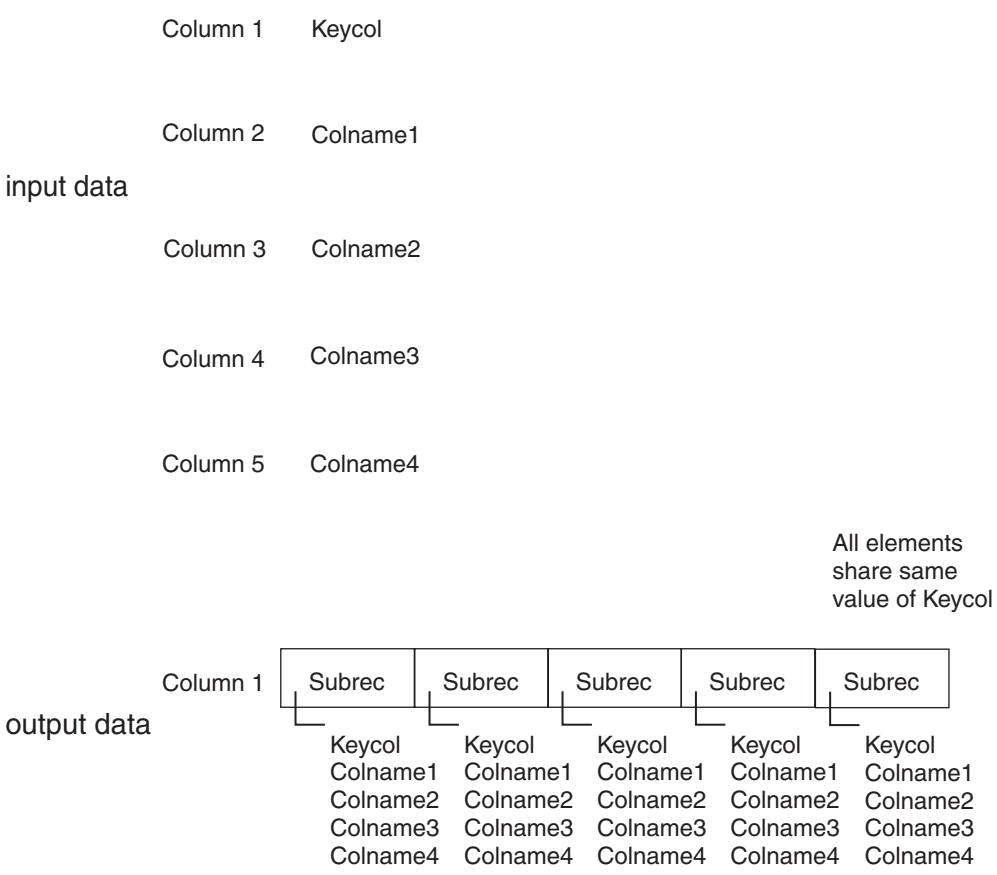
The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

See "Stage Editors," for a general description of these tabs.

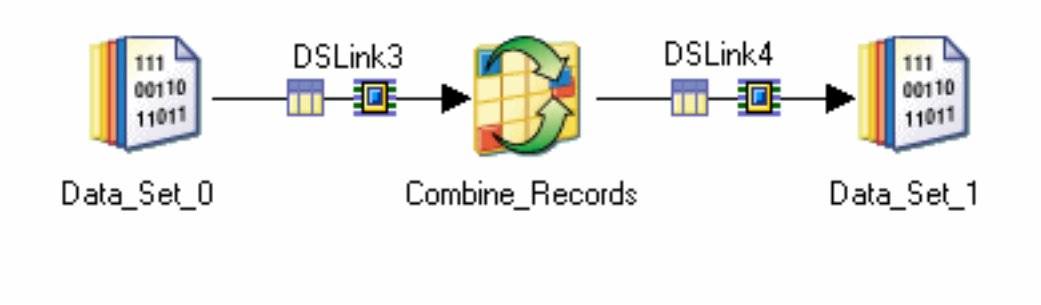
Combine Records stage

The Combine Records stage is restructure stage. It can have a single input link and a single output link.

The Combine Records stage combines records (that is, rows), in which particular key-column values are identical, into vectors of subrecords. As input, the stage takes a data set in which one or more columns are chosen as keys. All adjacent records whose key columns contain the same value are gathered into the same record as subrecords.



The data set input to the Combine Records stage must be key partitioned and sorted. This ensures that rows with the same key column values are located in the same partition and will be processed by the same node. Choosing the (auto) partitioning method will ensure that partitioning and sorting is done. If sorting and partitioning are carried out on separate stages before the Combine Records stage, InfoSphere DataStage in auto mode will detect this and not repartition (alternatively you could explicitly specify the Same partitioning method).



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Examples

This section gives examples of input and output data from a Combine Records stage to give you a better idea of how the stage works.

Example 1

This example assumes that the job is running sequentially. The column definitions for the input data set are as follows:

Table 93. Column definitions

Column name	Key	SQL type
keycol	Yes	Char
col1		TinyInt
col2		Time
col3		Dat

The following are some rows from the input data set:

Table 94. Input data set

	col1	col2	col3	col4
row	1	00:11:01	1960-01-02	A
row	3	08:45:54	1946-09-15	A
row	1	12:59:00	1955-12-22	B
row	2	07:33:04	1950-03-10	B
row	2	12:00:00	1967-02-06	B
row	2	07:37:04	1950-03-10	B
row	3	07:56:03	1977-04-14	B
row	3	09:58:02	1960-05-18	B
row	1	11:43:02	1980-06-03	C
row	2	01:30:01	1985-07-07	C
row	2	11:30:01	1985-07-07	C
row	3	10:28:02	1992-11-23	C
row	3	12:27:00	1929-08-11	C
row	3	06:33:03	1999-10-19	C
row	3	11:18:22	1992-11-23	C

Once combined by the stage, each group of rows will be output in a single column called suprecol. This contains the keycoll, col1, col2, and col3 columns. (If you do not take advantage of the runtime column propagation feature, you would have to set up the subrecord using the Edit Column Meta Data dialog box to set a level number for each of the columns the subrecord column contains.)

Table 95. Output metadata

Level number	Column name	SQL type	Key
	subreccol	Char	
02	keycol	Char	Yes
02	col1	TinyInt	

Table 95. Output metadata (continued)

Level number	Column name	SQL type	Key
02	col2	Time	
02	col3	Date	

The Properties of the stage are set as follows:

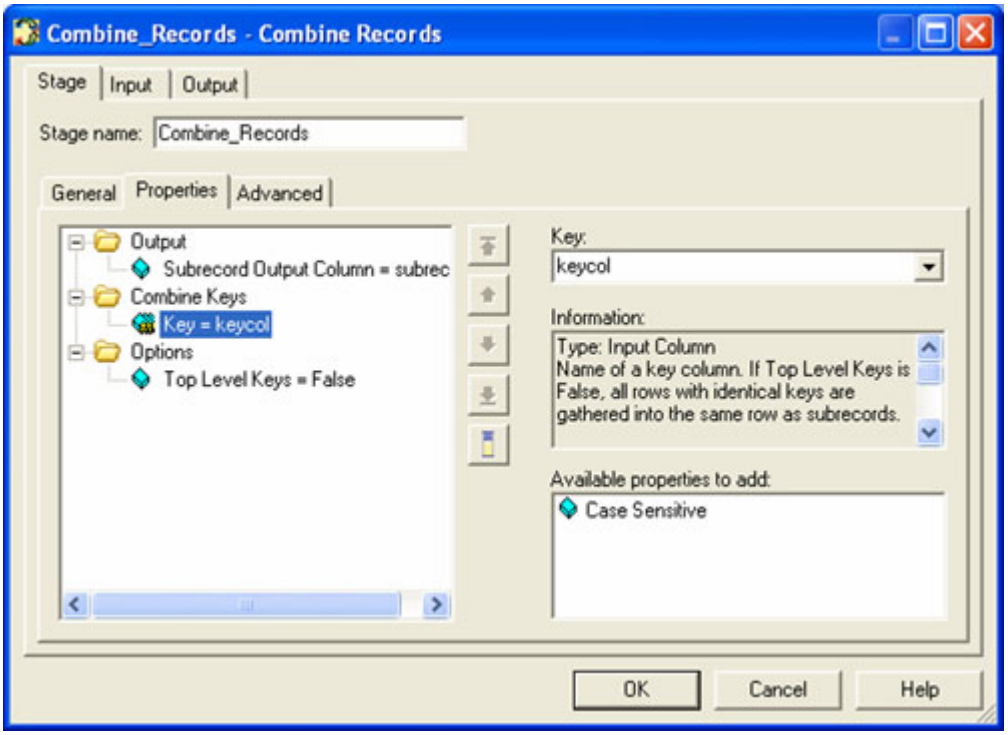


Figure 31. Properties tab

The Output data set will be:

Table 96. Output data set

		subreccol			
	vector index	**col1**	**col2**	**col3**	**Keycol**
row	0	1	00:11:01	1960-01-02	A
	1	3	08:45:54	1946-09-15	A
row	0	1	12:59:00	1955-12-22	B
	1	2	07:33:04	1950-03-10	B
	2	2	12:00:00	1967-02-06	B
	3	2	07:37:04	1950-03-10	B
	4	3	07:56:03	1977-04-14	B
	5	3	09:58:02	1960-05-18	B
row	0	1	11:43:02	1980-06-03	C
	1	2	01:30:01	1985-07-07	C
	2	2	11:30:01	1985-07-07	C

Table 96. Output data set (continued)

		subreccol			
	vector index	**col1**	**col2**	**col3**	**Keycol**
	3	3	10:28:02	1992-11-23	C
	4	3	12:27:00	1929-08-11	C
	5	3	06:33:03	1999-10-19	C
	6	3	11:18:22	1992-11-23	C

Example 2

This example shows a more complex structure that can be derived using the Top Level Keys Property. This can be set to True to indicate that key columns should be left as top level columns and not included in the subrecord. This example assumes that the job is running sequentially. The same column definition are used, except both col1 and keycol are defined as keys.

Table 97. Column definitions

Column name	**Key**	**SQL type**
keycol	Yes	Char
col1	Yes	TinyInt
col2		Time
col3		Dat

The same input data set is used:

Table 98. Input data set

	col1	**col2**	**col3**	**col4**
row	1	00:11:01	1960-01-02	A
row	3	08:45:54	1946-09-15	A
row	1	12:59:00	1955-12-22	B
row	2	07:33:04	1950-03-10	B
row	2	12:00:00	1967-02-06	B
row	2	07:37:04	1950-03-10	B
row	3	07:56:03	1977-04-14	B
row	3	09:58:02	1960-05-18	B
row	1	11:43:02	1980-06-03	C
row	2	01:30:01	1985-07-07	C
row	2	11:30:01	1985-07-07	C
row	3	10:28:02	1992-11-23	C
row	3	12:27:00	1929-08-11	C
row	3	06:33:03	1999-10-19	C
row	3	11:18:22	1992-11-23	C

The Output column definitions have two separate columns defined for the keys, as well as the column carrying the subrecords:

Table 99. Output column definitions

Level number	Column name	Key	SQL type
	subreccol	Char	
02	keycol	Char	Yes
02	col1	TinyInt	Yes
02	col2	Time	
02	col3	Date	

The properties of the stage are set as follows:

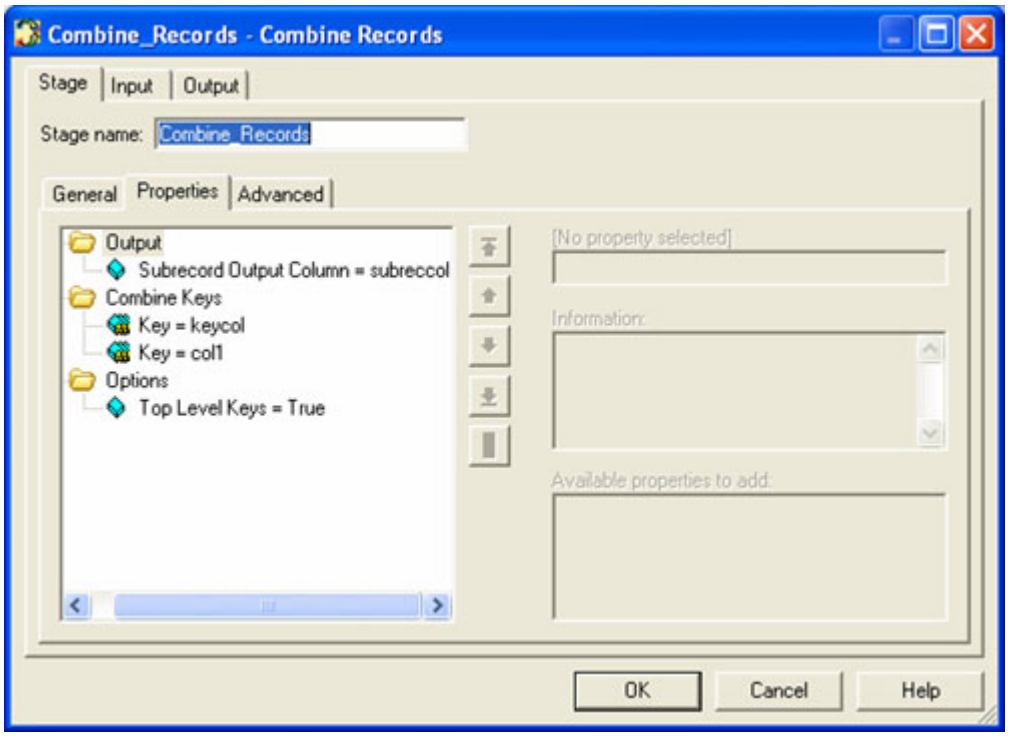


Figure 32. Properties tab

The Output data set will be:

Table 100. Output data set

	Keycol	col1	subreccol		
			vector index	col2	col3
row	A	1	0	00:11:01	1960-01-02
row	A	3	0	08:45:54	1946-09-15
row	B	1	0	12:59:00	1955-12-22
row	B	2	0	07:33:04	1950-03-10
			1	12:00:00	1967-02-06
			2	07:37:04	1950-03-10
row	B	3	0	07:56:03	1977-04-14
			1	09:58:02	1960-05-18

Table 100. Output data set (continued)

	Keycol	col1	subreccol		
			vector index	**col2**	**col3**
row	C	1	0	11:43:02	1980-06-03
row	C	2	0	01:30:01	1985-07-07
			1	11:30:01	1985-07-07
row	C	3	0	10:28:02	1992-11-23
			1	12:27:00	1929-08-11
			2	06:33:03	1999-10-19
			3	11:18:22	1992-11-23

Combine Records stage: fast path

This section specifies the minimum steps to take to get a Combine Records stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Combine Records stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

1. Go to the Stage page **Properties Tab**, under the Output category.
2. Specify the output column to carry the vector of subrecords in the Subrecord Output Column.
3. Under the Combine Keys category, specify the key column. Repeat the property to specify a composite key. All adjacent rows sharing the same value of the keys will be combined into a single row using the vector of subrecords structure.

Combine Records stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Combine Records stage: Properties tab

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 101. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Subrecord Output Column	Output Column	N/A	Y	N	N/A
Options/Key	Input Column	N/A	Y	Y	N/A

Table 101. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Case Sensitive	True/False	True	N	N	Key
Options/Top Level Keys	True/False	False	N	N	N/A

Combine Records stage: Outputs category: Subrecord output column

Specify the name of the subrecord that the Combine Records stage creates.

Combine Records stage: Combine keys category: Key

Specify one or more columns. You can use the Column Selection dialog box to select multiple columns at once if required. All records whose key columns contain identical values are gathered into the same record as subrecords. If the Top Level Keys property is set to False, each column becomes the element of a subrecord.

If the Top Level Keys property is set to True, the key column appears as a top-level column in the output record as opposed to in the subrecord. All non-key columns belonging to input records with that key column appear as elements of a subrecord in that key column's output record. Key has the following dependent property:

- **Case Sensitive**

  Use this to property to specify whether each key is case sensitive or not. It is set to True by default; for example, the values "CASE" and "case" would not be judged equivalent.

Combine Records stage: Options category: Top level keys

Specify whether to leave keys as top-level columns or have them put into the subrecord. False by default.

Combine Records stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pools or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse

button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Combine Records stage: NLS Locale tab

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Combine Records stage uses this when it is determining the sort order for key columns. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Combine Records stage: Input page

The Input page allows you to specify details about the incoming data sets. The Combine Records stage expects one incoming data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being converted. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Combine Records stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Combine Records stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is converted. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.Auto mode ensures that data being input to the Combine Records stage is hash partitioned and sorted.

If the Combine Records stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Combine Records stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Combine Records stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Combine Records stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Combine Records stage.

- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Combine Records stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available. In the case of a Combine Records stage, Auto will also ensure that the collected data is sorted.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being converted. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Combine Records stage: Output page

The Output page allows you to specify details about data output from the Combine Records stage. The Combine Records stage can have only one output link.

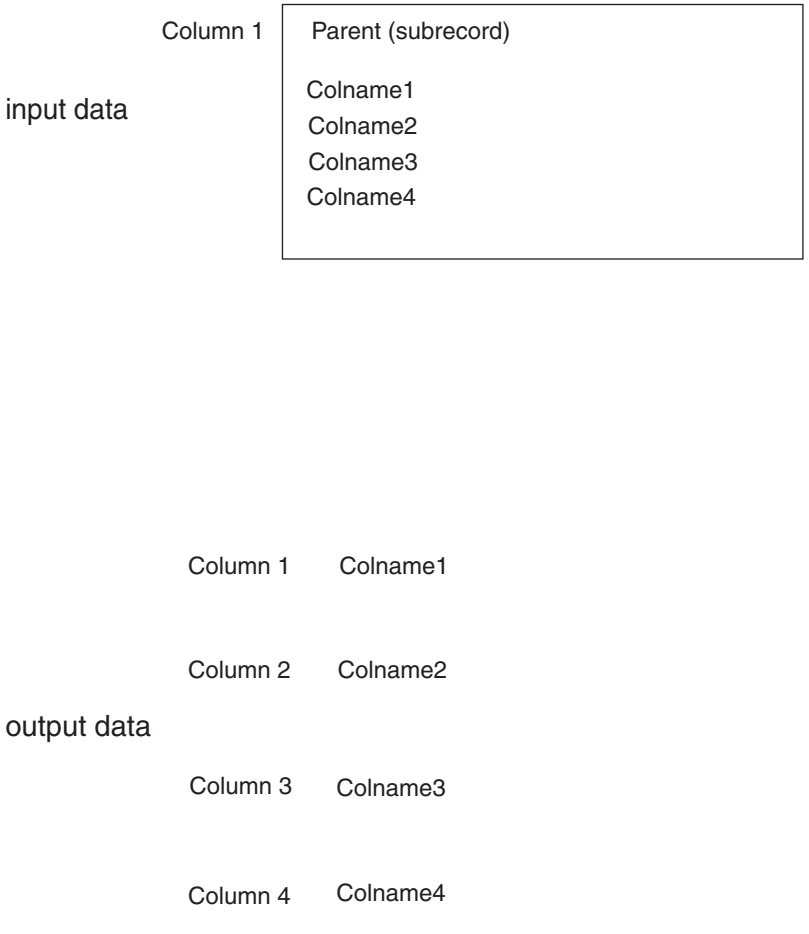
The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

See "Stage Editors," for a general description of the tabs.

Promote Subrecord stage

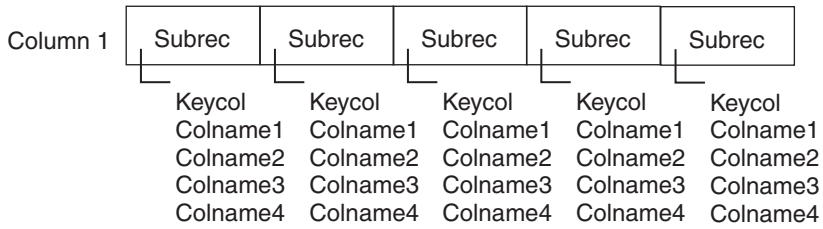
The Promote Subrecord stage is a restructure stage. It can have a single input link and a single output link.

The Promote Subrecord stage promotes the columns of an input subrecord to top-level columns. The number of output columns equals the number of subrecord elements. The data types of the input subrecord columns determine those of the corresponding top-level columns.



The stage can also promote the columns in vectors of subrecords, in which case it acts as the inverse of the Combine Records stage.

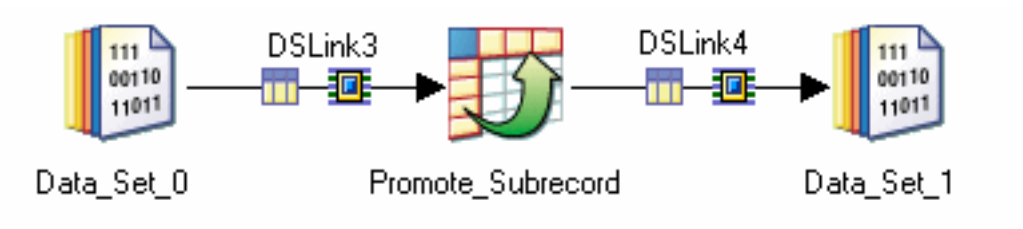
input data



output data

Column 1	Keycol
Column 2	Colname1
Column 3	Colname2
Column 4	Colname3
Column 5	Colname4

The Combine Records stage performs the inverse operation.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Examples

This section gives examples of input and output data from a Promote Subrecord stage to give you a better idea of how the stage works.

Example 1

In this example the Promote Subrecord stage promotes the records of a simple subrecord to top level columns. It extracts data from a single column containing a subrecord. The data is output in four columns, each carrying a column from the subrecord. The example assumes that the job is running sequentially. The column definition for the input data set contains a definition of a single column called subrec.

The following are the rows from the input data set:

Table 102. Input data set

		subrec			
	subrecord column name	**col1**	**col2**	**col3**	**col4**
row		1	AAD	Thurs	No
row		2	ABD	Thurs	No
row		3	CAD	Weds	Yes
row		4	CCC	Mon	Yes
		5	BDD	Mon	Yes
		6	DAK	Fri	No
		7	MDB	Tues	Yes

The stage outputs the data it extracts from the subrecord in four separate columns of appropriate type. The output column definitions are as follows:

Table 103. Output column definitions

Column name	**SQL type**
col1	TinyInt
col2	Char
col3	Char
col4	Char

The Subrecord Column property on the Properties tab of the Promote Subrecord stage is set to 'subrec'.

The output data set will be:

Table 104. Output data set

	Col1	**Col2**	**Col3**	**Col4**
row	1	AAD	Thurs	No
row	2	ABD	Thurs	No
row	3	CAD	Weds	Yes
row	4	CCC	Mon	Yes

Example 2

This example shows how the Promote Subrecord would operate on an aggregated vector of subrecords, as would be produced by the Combine Records stage. It assumes that the job is running sequentially. The column definition for the input data set contains a definition of a single column called subrec.

The following are some rows from the input data set:

Table 105. Input data set

		subreccol	**subreccol**	**subreccol**	**subreccol**
	vector index	**col1**	**col2**	**col3**	**Keycol**
row	0	1	00:11:01	1960-01-02	A

Table 105. Input data set (continued)

		subreccol	subreccol	subreccol	subreccol
	vector index	**col1**	**col2**	**col3**	**Keycol**
	1	3	08:45:54	1946-09-15	A
row	0	1	12:59:00	1955-12-22	B
	1	2	07:33:04	1950-03-10	B
	2	2	12:00:00	1967-02-06	B
	3	2	07:37:04	1950-03-10	B
	4	3	07:56:03	1977-04-14	B
	5	3	09:58:02	1960-05-18	B
row	0	1	11:43:02	1980-06-03	C
	1	2	01:30:01	1985-07-07	C
	2	2	11:30:01	1985-07-07	C
	3	3	10:28:02	1992-11-23	C
	4	3	12:27:00	1929-08-11	C
	5	3	06:33:03	1999-10-19	C
	6	3	11:18:22	1992-11-23	C

Once the columns in the subrecords have been promoted the data will be output in four columns as follows:

Table 106. Promoted columns

Column name	**Key**	**SQL type**
keycol	Yes	Char
col1		TinyInt
col2		Time
col3		Dat

The Subrecord Column property on the Properties tab of the Promote Subrecord stage is set to 'subrec'.

The Output data set will be:

Table 107. Output data set

	col1	**col2**	**col3**	**col4**
row	1	00:11:01	1960-01-02	A
row	3	08:45:54	1946-09-15	A
row	1	12:59:00	1955-12-22	B
row	2	07:33:04	1950-03-10	B
row	2	12:00:00	1967-02-06	B
row	2	07:37:04	1950-03-10	B
row	3	07:56:03	1977-04-14	B
row	3	09:58:02	1960-05-18	B
row	1	11:43:02	1980-06-03	C
row	2	01:30:01	1985-07-07	C

Table 107. Output data set (continued)

	col1	col2	col3	col4
row	2	11:30:01	1985-07-07	C
row	3	10:28:02	1992-11-23	C
row	3	12:27:00	1929-08-11	C
row	3	06:33:03	1999-10-19	C
row	3	11:18:22	1992-11-23	C

Promote Subrecord stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Promote Subrecord stages in a job. This section specifies the minimum steps to take to get a Promote Subrecord stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Promote Subrecord stage:

- In the Outputs Stage **Properties Tab**, under the Input category:
  - Specify the subrecord column that the stage will promote subrecords from.

Promote Subrecord stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Promote Subrecord stage: Properties tab

The Promote Subrecord Stage has one property:

Table 108. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Subrecord Column	Input Column	N/A	Y	N	N/A

Promote Subrecord stage: Options category: Subrecord column

Specifies the name of the subrecord whose elements will be promoted to top-level records.

Promote Subrecord stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the **Advanced** tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.

- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Promote Subrecord stage: Input page

The Input page allows you to specify details about the incoming data sets. The Promote Subrecord stage expects one incoming data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being converted. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Promote Subrecord stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Promote Subrecord stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is converted. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Promote Subrecord stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Promote Subrecord stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Promote Subrecord stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Promote Subrecord stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Promote Subrecord stage.
- **Entire**. Each file written to receives the entire data set.

- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place. This is the default partitioning method for the Promote Subrecord stage.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Promote Subrecord stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being converted. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning methods chosen (it is not available with the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Promote Subrecord stage: Output page

The Output page allows you to specify details about data output from the Promote Subrecord stage. The Promote Subrecord stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

See "Stage Editors," for a general description of the tabs.

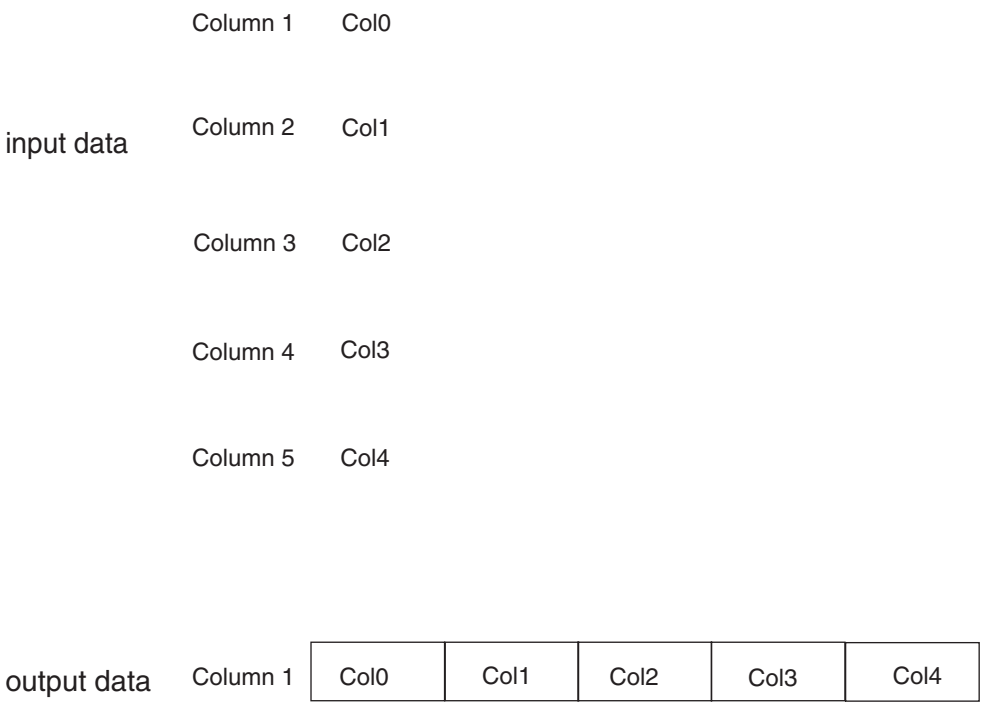
Make Vector stage

The Make Vector stage is an active stage. It can have a single input link and a single output link.

The Make Vector stage combines specified columns of an input data record into a vector of columns. The stage has the following requirements:

- The input columns must form a numeric sequence, and must all be of the same type.
- The numbers must increase by one.
- The columns must be named column_name0 to column_namen, where column_name starts the name of a column and 0 and *n* are the first and last of its consecutive numbers.
- The columns do not have to be in consecutive order.

All these columns are combined into a vector of the same length as the number of columns (n+1). The vector is called column_name. Any input columns that do not have a name of that form will not be included in the vector but will be output as top level columns.



The Split Vector stage performs the inverse operation. See "Split Vector Stage."

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Examples

This section gives examples of input and output data from a Make Vector stage to give you a better idea of how the stage works.

Example 1

In this example, all the input data will be included in the output vector. The example assumes that the job is running sequentially. The column definitions for the input data set are in the following table. Note the columns all have the same type and names in the form *column_nameN.*:

Table 109. Column definitions

Column name	SQL type
col0	TinyInt
col1	TinyInt
col2	TinyInt
col3	TinyInt
col4	TinyInt

The following are some rows from the input data set:

Table 110. Input data set

	col0	col1	col2	col3	col4
row	3	6	2	9	9
row	3	2	7	2	4
row	7	8	8	5	3
row	4	8	7	1	6
row	1	6	2	5	1
row	0	1	6	7	8
row	9	9	6	4	2
row	0	8	4	4	3
row	1	7	2	5	3
row	7	9	4	7	8

The stage outputs the vectors it builds from the input data in a single column called col. You do not have to explicitly define the output column name, InfoSphere DataStage will do this for you as the job runs, but you might wish to do so to make the job more understandable.

The Column's Common Partial Name property is set to 'col' in the Properties tab.

The output data set will be:

Table 111. Output data set

	col				
Vector index	**0**	**1**	**2**	**3**	**4**
row	3	6	2	9	9
row	3	2	7	2	4
row	7	8	8	5	3
row	4	8	7	1	6
row	1	6	2	5	1
row	0	1	6	7	8
row	9	9	6	4	2
row	0	8	4	4	3
row	1	7	2	5	3
row	7	9	4	7	8

Example 2

In this example, there are additional columns as well as the ones that will be included in the vector. The example assumes that the job is running sequentially. The column definitions for the input data set are show below, note the additional columns called *name* and *code*:

Table 112. Column definitions

Column name	**SQL type**
name	Char
code	Char
col0	TinyInt
col1	TinyInt
col2	TinyInt
col3	TinyInt
col4	TinyInt

The following are some rows from the input data set:

Table 113. Input data set

	Name	**Code**	**col0**	**col1**	**col2**	**col3**	**col4**
row	Will	D070	3	6	2	9	9
row	Robin	GA36	3	2	7	2	4
row	Beth	B777	7	8	8	5	3
row	Heathcliff	A100	4	8	7	1	6
row	Chaz	CH01	1	6	2	5	1

Table 113. Input data set (continued)

	Name	**Code**	**col0**	**col1**	**col2**	**col3**	**col4**
row	Kayser	CH02	0	1	6	7	8
row	Jayne	M122	9	9	6	4	2
row	Ann	F234	0	8	4	4	3
row	Kath	HE45	1	7	2	5	3
row	Rupert	BC11	7	9	4	7	8

The stage outputs the vectors it builds from the input data in a single column called *column_name*. The two other columns are output separately. You do not have to explicitly define the output column names, InfoSphere DataStage will do this for you as the job runs, but you might wish to do so to make the job more understandable.

Table 114. Output column definitions

Column name	**SQL type**
name	Char
code	Char
col	Char

The Column's Common Partial Name property is set to 'col' in the Properties tab

The output data set will be:

Table 115. Output data set

Vector index			**0**	**1**	**2**	**3**	**4**
row	Will	D070	3	6	2	9	9
row	Robin	GA36	3	2	7	2	4
row	Beth	B777	7	8	8	5	3
row	Heathcliff	A100	4	8	7	1	6
row	Chaz	CH01	1	6	2	5	1
row	Kayser	CH02	0	1	6	7	8
row	Jayne	M122	9	9	6	4	2
row	Ann	F234	0	8	4	4	3
row	Kath	HE45	1	7	2	5	3
row	Rupert	BC11	7	9	4	7	8

Make Vector stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Make Vector stages in a job. This section specifies the minimum steps to take to get a Make Vector stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Make Vector stage:

- In the Stage Page **Properties Tab**:
  - Specify the Column's Common Partial Name, this is the *column_name* part that is shared by all the columns in the input data set, which will be the name of the output column containing the vector.

Make Vector stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Make Vector stage: Properties tab

The Make Vector stage has one property:

Table 116. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/ Column's Common Partial Name	Name	N/A	Y	N	N/A

Make Vector stage: Options category:
Column's common partial name

Specifies the beginning *column_name* of the series of consecutively numbered columns *column_name0* to *column_namen* to be combined into a vector called *column_name*.

Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Make Vector stage: Input page

The Input page allows you to specify details about the incoming data sets. The Make Vector stage expects one incoming data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being converted. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Make Vector stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Make Vector stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is converted. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode.

If the Make Vector stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Make Vector stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Make Vector stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Make Vector stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place. This is the default partitioning method for the Make Vector stage.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Make Vector stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being converted. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Make Vector stage: Output page

The Output page allows you to specify details about data output from the Make Vector stage. The Make Vector stage can have only one output link.

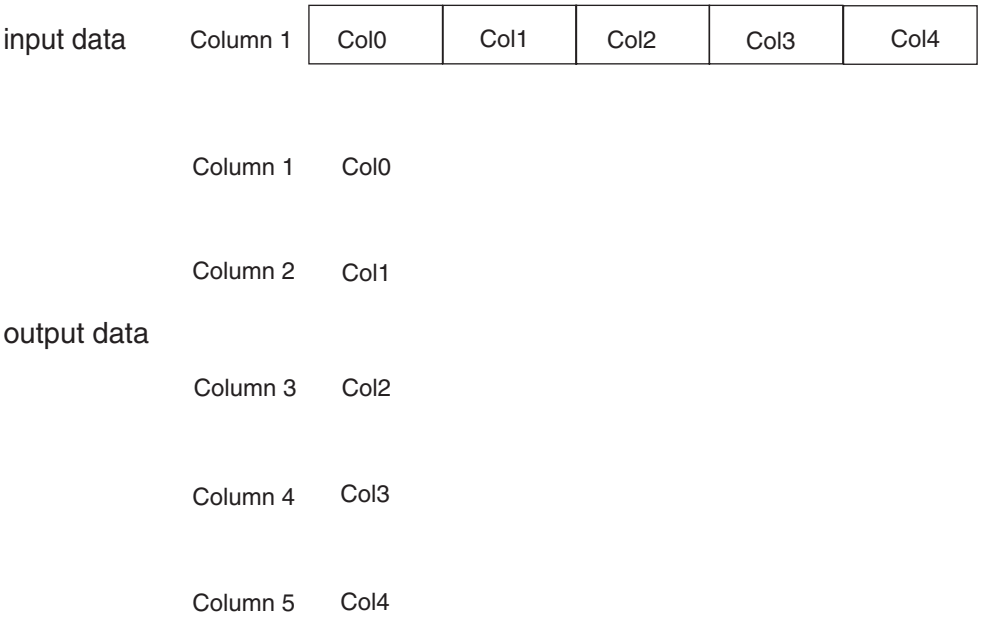
The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

See "Stage Editors," for a general description of the tabs.

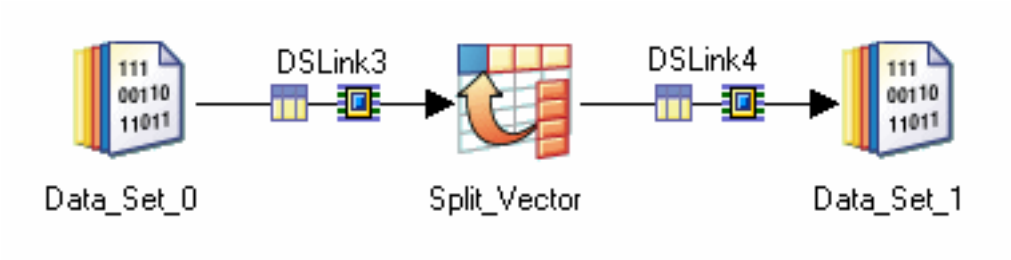
Split Vector Stage

The Split Vector stage is a restructure stage. It can have a single input link and a single output link.

The Split Vector stage promotes the elements of a fixed-length vector to a set of similarly named top-level columns. The stage creates columns of the format name0 to namen, where name is the original vector's name and 0 and *n* are the first and last elements of the vector.



The Make Vector stage performs the inverse operation (see "Make Vector stage" on page 486).



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Examples

This section gives examples of input and output data from a Split Vector stage to give you a better idea of how the stage works.

Example 1

In this example the input data comprises a single column carrying a vector. The example assumes that the job is running sequentially. The input data set has a single column called 'col'..

The following are some rows from the input data set:

Table 117. Input data set

Vector index	**0**	**1**	**2**	**3**	**4**
row	3	6	2	9	9
row	3	2	7	2	4

Table 117. Input data set (continued)

Vector index	**0**	**1**	**2**	**3**	**4**
row	7	8	8	5	3
row	4	8	7	1	6
row	1	6	2	5	1
row	0	1	6	7	8
row	9	9	6	4	2
row	0	8	4	4	3
row	1	7	2	5	3
row	7	9	4	7	8

The stage splits the columns it extracts from the vector into separate columns called *column_nameN*. You do not have to explicitly define the output column names, InfoSphere DataStage will do this for you as the job runs, but you might wish to do so to make the job more understandable.

Table 118. Output column names

Column name	**SQL type**
col0	TinyInt
col1	TinyInt
col2	TinyInt
col3	TinyInt
col4	TinyInt

The Vector Column property in the Properties tab is set to 'col'.

The output data set will be:

Table 119. Output data set

	col0	**col1**	**col2**	**col3**	**col4**
row	3	6	2	9	9
row	3	2	7	2	4
row	7	8	8	5	3
row	4	8	7	1	6
row	1	6	2	5	1
row	0	1	6	7	8
row	9	9	6	4	2
row	0	8	4	4	3
row	1	7	2	5	3
row	7	9	4	7	8

Example 2

In this example, there are additional columns as well as the ones containing the vector. The example assumes that the job is running sequentially. The column definitions for the input data set are as follows, note the additional columns called *name* and *code*:

Table 120. Column definitions

Column name	SQL type
name	Char
code	Char
col	Char

The following are some rows from the input data set:

Table 121. Input data set

Vector index			**0**	**1**	**2**	**3**	**4**
row	Will	D070	3	6	2	9	9
row	Robin	GA36	3	2	7	2	4
row	Beth	B777	7	8	8	5	3
row	Heathcliff	A100	4	8	7	1	6
row	Chaz	CH01	1	6	2	5	1
row	Kayser	CH02	0	1	6	7	8
row	Jayne	M122	9	9	6	4	2
row	Ann	F234	0	8	4	4	3
row	Kath	HE45	1	7	2	5	3
row	Rupert	BC11	7	9	4	7	8

The stage splits the columns it extracts from the vector into separate columns called *column_nameN*. You do not have to explicitly define the output column names, InfoSphere DataStage will do this for you as the job runs, but you might wish to do so to make the job more understandable.

Table 122. Output column definitions

Column name	SQL type
name	Char
code	Char
col0	TinyInt
col1	TinyInt
col2	TinyInt
col3	TinyInt
col4	TinyInt

The Vector Column property in the Properties tab is set to 'col'.

The output data set will be:

Table 123. Output data set

	Name	**Code**	**col0**	**col1**	**col2**	**col3**	**col4**
row	Will	D070	3	6	2	9	9
row	Robin	GA36	3	2	7	2	4
row	Beth	B777	7	8	8	5	3
row	Heathcliff	A100	4	8	7	1	6

Table 123. Output data set (continued)

	Name	Code	col0	col1	col2	col3	col4
row	Chaz	CH01	1	6	2	5	1
row	Kayser	CH02	0	1	6	7	8
row	Jayne	M122	9	9	6	4	2
row	Ann	F234	0	8	4	4	3
row	Kath	HE45	1	7	2	5	3
row	Rupert	BC11	7	9	4	7	8

Split Vector stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Split Vector stages in a job. This section specifies the minimum steps to take to get a Split Vector stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Split Vector stage:

- In the Stage Page **Properties Tab**:
  - Specify the name of the input column carrying the vector to be split.

Split Vector stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Split Vector stage: Properties tab

The Split Vector stage has one property:

Table 124. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Vector Column	Name	N/A	Y	N	N/A

Split Vector stage: Options category: Vector column

Specifies the name of the vector whose elements you want to promote to a set of similarly named top-level columns.

Split Vector stage: Advanced tab

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.

- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Split Vector stage: Input page

The Input page allows you to specify details about the incoming data sets. There can be only one input to the Split Vector stage.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being converted. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Split Vector stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Split Vector stage: Partitioning tab

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is converted. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file.

If the Split Vector stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Split Vector stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Split Vector stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Split Vector stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Split Vector stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Split Vector stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being converted. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default Auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Split Vector stage: Output page

The Output page allows you to specify details about data output from the Split Vector stage. The Split Vector stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the output link.

See "Stage Editors," for a general description of the tabs.

Chapter 9. Debugging parallel jobs

Run your parallel jobs in debug mode or use the debugging stages to help you debug your parallel job designs.

Running parallel jobs in debug mode

You debug a parallel job by setting one or more breakpoints on the links in the job, running the job in debug mode, and examining the column data when the job stops at the breakpoints.

About this task

From within your job design, you can use the Debug toolbar options or the **Debug** menu to debug your job designs. Set breakpoints on the links where you want the job to stop when you run the job in debug mode, so you can understand the job flow. When the job stops at a breakpoint, you can examine the column data that is being processed.

When you run a parallel job in debug mode, each processing node stops when it reaches a breakpoint. The Debug Window shows how many breakpoints have been reached, and how many processing nodes have stopped at each of the breakpoints. Column data is shown for each processing node that has stopped at a breakpoint. You can also add columns to the Watch List to see their values in all the processing nodes in the Debug Window.

Note: If you close and reopen a parallel job, breakpoints are retained. When you compile a job, breakpoints are validated. If you rename, move, or move either end of a link that contains a breakpoint, the breakpoint is retained. If you delete a link and create another link with the same name, the new link does not inherit the breakpoint. Also, if you upgrade, export, or save a job with a different name, breakpoints are not inherited in those new jobs. Lastly, you cannot set breakpoints on links in shared containers.

Adding a breakpoint to a parallel job

To debug a parallel job, you must add one or more breakpoints on links in that parallel job.

Procedure

1. Select the required link on the parallel job canvas.
2. In the Debug Window, on the toolbar, click **Toggle Breakpoint**. A circle is drawn on the link to indicate that a breakpoint has been added
3. Define the breakpoint.
   a. In the Debug Window, on the toolbar, click **Edit Breakpoints**.
   b. In the **Breakpoints** list, select the breakpoint you want to define.
   c. Set the required **Break On** criteria.

What to do next

Run your parallel job in debug mode.

Running a parallel job in debug mode

To stop the processing of your parallel job at a breakpoint so you can examine its state, run the parallel job in debug mode.

Before you begin

1. Your parallel job must have been compiled successfully before you can debug it.
2. You must have defined the required job parameters before you run your parallel job in debug mode. Click **Debug** > **Job Parameters** to open the **Job Run Options** window to specify the job parameters. Parameters values that are set on the **General** tab are not persisted between runs.
3. You must have set up your required breakpoints before you run your parallel job in debug mode.
4. You must have set the APT_PXDEBUGGER_FORCE_SEQUENTIAL environment variable if you want any stages to be run in sequential mode.

Procedure

1. Click **Debug** > **Debug Window**.
2. Optional: To show the job entries for your parallel job, click **View** > **Job Log**.
3. In the Debug Window, click **Start/Continue Debugging**. Your job is now running in debug mode. The status pane in the Debug Window provides information about the progress of your job. The **Active Breakpoints** list shows the breakpoints where one or more processing nodes have stopped.
4. Select the required breakpoint in the **Active Breakpoints** list. The **Column data per node** pane changes to show information for your selected breakpoint. A separate tab is displayed for each processing node that has stopped at that breakpoint, showing the values for each column in the row that is being processed by the node.
5. Select a column on any of tabs in the **Column data per node** pane. The same column is highlighted in all the tabs.
6. Right-click a column on one tab in the **Column data per node** pane, and click **Add to Watch list**. The column is added to the Watch list pane, which shows the column values for all of the processing nodes. You can use the Watch list to monitor the value of a column as your parallel job stops at each breakpoint.

   To remove a column from the Watch list, right-click the column in the Watch list pane, and click **Remove**.
7. Optional: To restart all the processing nodes that have stopped at a breakpoint, and to run the job to completion, click **Run to End**. The job runs to completion, ignoring all further breakpoints.
8. Optional: To stop your job running, click **Stop Job**.
9. To restart all the processing nodes that have stopped at a breakpoint, click **Start/Continue Debugging**.

Debugging parallel jobs with the debugging stages

Use the stages in the Debugging section of the palette to help you debug your parallel job designs.

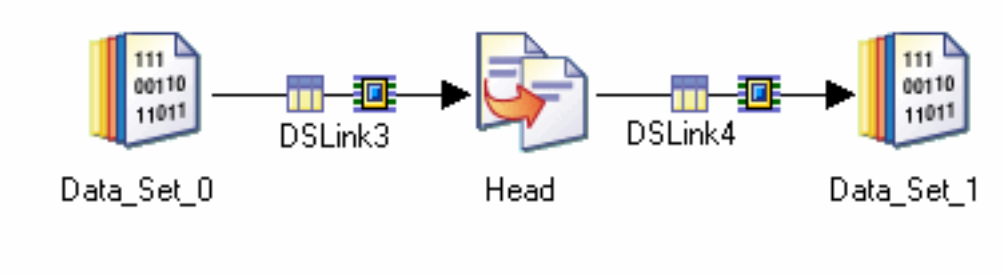
Head stage

The Head Stage is a Development/Debug stage. It can have a single input link and a single output link. It is one of a number of stages that InfoSphere DataStage provides to help you sample data, see also:

- Tail stage, “Tail stage” on page 508.
- Sample stage, “Sample stage” on page 513.
- Peek stage, “Peek stage” on page 522.

The Head Stage selects the first *N* rows from each partition of an input data set and copies the selected rows to an output data set. You determine which rows are copied by setting properties which allow you to specify:

- The number of rows to copy
- The partition from which the rows are copied
- The location of the rows to copy
- The number of rows to skip before the copying operation begins



This stage is helpful in testing and debugging applications with large data sets. For example, the Partition property lets you see data from a single partition to determine if the data is being partitioned as you want it to be. The Skip property lets you access a certain portion of a data set.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Examples

Head stage default behavior:

The input data set comprises data from the Global customer billing information, which has previously been hash-partititioned into four partitions. You accept the default setting to sample ten rows from the start of each partition.

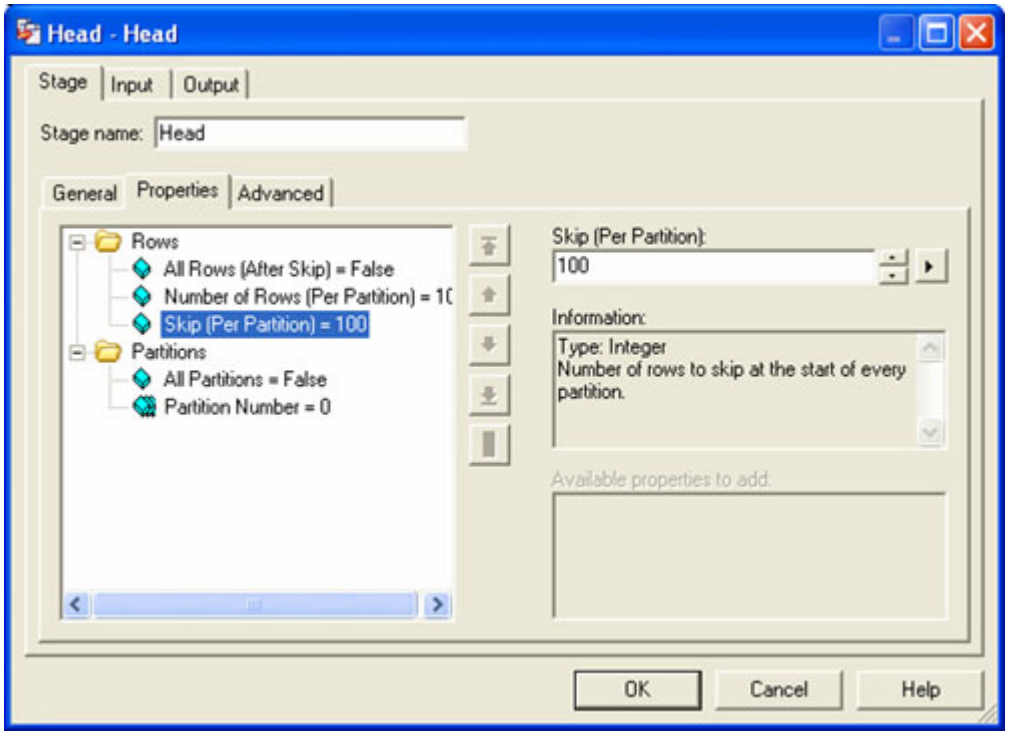
After the job is run you get a data set comprising four partitions each containing ten rows. Here is a sample of partition 0 as input to the Head stage, and partition 0 in its entirety as output by the stage:

```
"GC22145","ADELE BECKER","AM HUNGERSBERG 123","5/26/2004"
"GC20002","JEAN DUPONT","576 RUE DE PARIS","7/14/2003"
"GC13933","MARY GARDENER","127 BORDER ST","8/28/2009"
"GC20040","NICOLE GIRARD","1234 QUAI DE LA TOURNELLE","10/31/2007"
"GC21745","ELIZABETH PARKER","35 YORK ROAD","1/31/2006"
"GC14127","HUW WILLIAMS","579 DIGBETH AVENUE","6/29/2011"
"GC20049","DANIELLE BLANC","987 BOULEVARD AUXERRE","3/3/2003"
"GC13849","JON SMITH","789 LEDBURY ROAD","2/17/2007"
"GC14036","CHRIS TRAIN","1400 NEW ST","9/7/1998"
"GC14263","SARA PEARS","45 ALCESTER WAY","4/12/2008"
"GC21874","WINSTON HILL","87 MULBERRY CLOSE","GC21874","5/12/2011"
"GC14346","LUC TEACHER","3 BIRMINGHAM ROAD","11/7/2010"
"GC20021","PIERRE FOURNIER","321 RUE VOLTAIRE","11/14/2007"
"GC22478","KLAUS SCHMIDT","WOLBURGSWEG 7645","5/11/1999"
"GC20012","MARIE TISON","14 AVENUE DE CALAIS","12/30/2008"
```

```
"GC22145","ADELE BECKER","AM HUNGERSBERG 123","5/26/2004"
"GC20002","JEAN DUPONT","576 RUE DE PARIS","7/14/2003"
"GC13933","MARY GARDENER","127 BORDER ST","8/28/2009"
"GC20040","NICOLE GIRARD","1234 QUAI DE LA TOURNELLE","10/31/2007"
"GC21745","ELIZABETH PARKER","35 YORK ROAD","1/31/2006"
"GC14127","HUW WILLIAMS","579 DIGBETH AVENUE","6/29/2011"
"GC20049","DANIELLE BLANC","987 BOULEVARD AUXERRE","3/3/2003"
"GC13849","JON SMITH","789 LEDBURY ROAD","2/17/2007"
"GC14036","CHRIS TRAIN","1400 NEW ST","9/7/1998"
"GC14263","SARA PEARS","45 ALCESTER WAY","4/12/2008"
```

Skipping Data:

In this example we are using the same data, but this time we are only interested in partition 0, and are skipping the first 100 rows before we take the ten rows. The Head stage properties are set as follows:



Here is the data set output by the stage:

```
"GC21745","ELIZABETH PARKER","35 YORK ROAD","1/31/2006"
"GC14127","HUW WILLIAMS","579 DIGBETH AVENUE","6/29/2011"
"GC20049","DANIELLE BLANC","987 BOULEVARD AUXERRE","3/3/2003"
"GC13849","JON SMITH","789 LEDBURY ROAD","2/17/2007"
"GC14036","CHRIS TRAIN","1400 NEW ST","9/7/1998"
"GC14263","SARA PEARS","45 ALCESTER WAY","4/12/2008"
"GC21874","WINSTON HILL","87 MULBERRY CLOSE","GC21874","5/12/2011"
"GC14346","LUC TEACHER","3 BIRMINGHAM ROAD","11/7/2010"
"GC20021","PIERRE FOURNIER","321 RUE VOLTAIRE","11/14/2007"
"GC22478","KLAUS SCHMIDT","WOLBURGSWEG 7645","5/11/1999"
```

Head stage: fast path

This section specifies the minimum steps to take to get a Head stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Head stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

In the Stage page **Properties Tab**, under the Rows category:

1. Specify the number of rows per partition that you want to copy from the source data set to the target data set. This defaults to ten. You can also do the following:
   - Specify that the stage should skip the first *N* rows per partition.
   - Specify that the stage will output all rows in a partition after the skip.

- Specify that the stage should output every *N*th row.
- Under the Partitions category, specify that the stage will only output rows from the selected partitions.

2. In the Output Page **Mapping Tab**, specify how the headed data maps onto your output columns.

Head stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Head stage: Properties tab:

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 125. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Rows/All Rows	True/False	False	N	N	N/A
Rows/Number of Rows (per Partition)	Count	10	N	N	N/A
Rows/Period (per Partition)	Number	N/A	N	N	N/A
Rows/Skip (per Partition)	Number	N/A	N	N	N/A
Partitions/All Partitions	Partition Number	N/A	N	Y	N/A
Partitions/ Partition Number	Number	N/A	Y (if All Partitions = False)	Y	N/A

Head stage: Rows category:
All rows

Copy all input rows to the output data set. You can skip rows before Head performs its copy operation by using the Skip property. The Number of Rows property is not needed if All Rows is true.

Number of rows (per partition)

Specify the number of rows to copy from each partition of the input data set to the output data set. The default value is 10. The Number of Rows property is not needed if All Rows is true.

Period (per partition)

Copy every *P*th record in a partition, where *P* is the period. You can start the copy operation after records have been skipped by using the Skip property. *P* must equal or be greater than 1.

Skip (per Partition)

Ignore the first *number* of rows of each partition of the input data set, where *number* is the number of rows to skip. The default skip count is 0.

Head stage: Partitions category:
All partitions

If False, copy records only from the indicated partition, specified by number. By default, the operator copies rows from all partitions.

Partition number

Specifies particular partitions to perform the Head operation on. You can specify the Partition Number property multiple times to specify multiple partition numbers.

Head stage: Advanced tab:

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Head stage: Input page

The Input page allows you to specify details about the incoming data sets. The Head stage expects one input.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being headed. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Head stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Head stage: Partitioning tab:

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is headed. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages, and how many nodes are specified in the Configuration file.

If the Head stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Head stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Head stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Head stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages, and how many nodes are specified in the Configuration file. This is the default partitioning method for the Head stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Head stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being headed. The sort is always carried out within data partitions. If the stage is partitioning

incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Head stage: Output page

The Output page allows you to specify details about data output from the Head stage. The Head stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Head stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Head stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Head stage: Mapping tab:

For the Head stage the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them or how they are generated.

The left pane shows the input columns or the generated columns. These are read only and cannot be modified on this tab.

The right pane shows the output columns for each link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility.

Tail stage

The Tail Stage is a Development/Debug stage. It can have a single input link and a single output link. It is one of a number of stages that InfoSphere DataStage provides to help you sample data, see also:

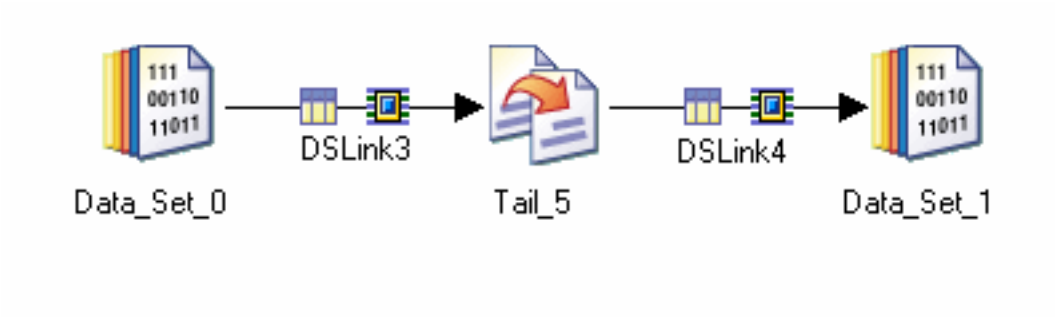
- Head stage, “Head stage” on page 502.
- Sample stage, “Sample stage” on page 513.
- Peek stage, “Peek stage” on page 522.

The Tail Stage selects the last *N* records from each partition of an input data set and copies the selected records to an output data set. You determine which records are copied by setting properties which allow you to specify:

- The number of records to copy

- The partition from which the records are copied

This stage is helpful in testing and debugging applications with large data sets. For example, the Partition property lets you see data from a single partition to determine if the data is being partitioned as you want it to be. The Skip property lets you access a certain portion of a data set.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Examples

The input data set comprises data from the Global customer billing information, which has previously been hash-partititioned into four partitions. You accept the default setting to sample ten rows from the end of each partition.

After the job is run you get a data set comprising four partitions each containing ten rows. Here is a sample of partition 0 as input to the Tail stage, and partition 0 in its entirety as output by the stage:

```
"GC22145","ADELE BECKER","AM HUNGERSBERG 123","5/26/2004"
"GC20002","JEAN DUPONT","576 RUE DE PARIS","7/14/2003"
"GC13933","MARY GARDENER","127 BORDER ST","8/28/2009"
"GC20040","NICOLE GIRARD","1234 QUAI DE LA TOURNELLE","10/31/2007"
"GC21745","ELIZABETH PARKER","35 YORK ROAD","1/31/2006"
"GC14127","HUW WILLIAMS","579 DIGBETH AVENUE","6/29/2011"
"GC20049","DANIELLE BLANC","987 BOULEVARD AUXERRE","3/3/2003"
"GC13849","JON SMITH","789 LEDBURY ROAD","2/17/2007"
"GC14036","CHRIS TRAIN","1400 NEW ST","9/7/1998"
"GC14263","SARA PEARS","45 ALCESTER WAY","4/12/2008"
"GC21874","WINSTON HILL","87 MULBERRY CLOSE","GC21874","5/12/2011"
"GC14346","LUC TEACHER","3 BIRMINGHAM ROAD","11/7/2010"
"GC20021","PIERRE FOURNIER","321 RUE VOLTAIRE","11/14/2007"
"GC22478","KLAUS SCHMIDT","WOLBURGSWEG 7645","5/11/1999"
"GC20012","MARIE TISON","14 AVENUE DE CALAIS","12/30/2008"
```

```
"GC14127","HUW WILLIAMS","579 DIGBETH AVENUE","6/29/2011"
"GC20049","DANIELLE BLANC","987 BOULEVARD AUXERRE","3/3/2003"
"GC13849","JON SMITH","789 LEDBURY ROAD","2/17/2007"
"GC14036","CHRIS TRAIN","1400 NEW ST","9/7/1998"
"GC14263","SARA PEARS","45 ALCESTER WAY","4/12/2008"
"GC21874","WINSTON HILL","87 MULBERRY CLOSE","GC21874","5/12/2011"
"GC14346","LUC TEACHER","3 BIRMINGHAM ROAD","11/7/2010"
"GC20021","PIERRE FOURNIER","321 RUE VOLTAIRE","11/14/2007"
"GC22478","KLAUS SCHMIDT","WOLBURGSWEG 7645","5/11/1999"
"GC20012","MARIE TISON","14 AVENUE DE CALAIS","12/30/2008"
```

Tail stage: fast path
About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Tail stages in a job. This section specifies the minimum steps to take to get a Tail stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Tail stage:

- In the Stage page **Properties Tab**, under the Rows category:
  - Specify the number of rows per partition that you want to copy from the source data set to the target data set. This defaults to ten.

    Under the Partitions category:
  - Specify that the stage will only output rows from the selected partitions.
- In the Output Page **Mapping Tab**, specify how the tailed data maps onto your output columns.

Tail stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Tail stage: Properties tab:

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 126. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Rows/Number of Rows (per Partition)	Count	10	N	N	Key
Partitions/All Partitions	Partition Number	N/A	N	Y	N/A
Partitions/ Partition Number	Number	N/A	Y (if All Partitions = False)	Y	N/A

Tail stage: Rows category:
Number of rows (per partition)

Specify the number of rows to copy from each partition of the input data set to the output data set. The default value is 10.

Tail stage: Partitions category:
All partitions

If False, copy records only from the indicated partition, specified by number. By default, the operator copies records from all partitions.

Partition number

Specifies particular partitions to perform the Tail operation on. You can specify the Partition Number property multiple times to specify multiple partition numbers.

Tail stage: Advanced tab:

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Tail stage: Input page

The Input page allows you to specify details about the incoming data sets. The Tail stage expects one input.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being tailed. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Tail stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Tail stage: Partitioning tab:

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is tailed. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. If the Preserve Partitioning option has been set on the previous stage in the job, this stage will warn if it cannot preserve the partitioning of the incoming data.

If the Tail stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Tail stage is set to execute in parallel or sequential mode.

- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Tail stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Tail stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default partitioning method for the Tail stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for Tail stages. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being tailed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning method chosen.

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Tail stage: Output page

The Output page allows you to specify details about data output from the Tail stage. The Tail stage can have only one output link.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Tail stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output link.

Details about Tail stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Tail stage: Mapping tab:

For the Tail stage the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them or how they are generated.

The left pane shows the input columns or the generated columns. These are read only and cannot be modified on this tab.

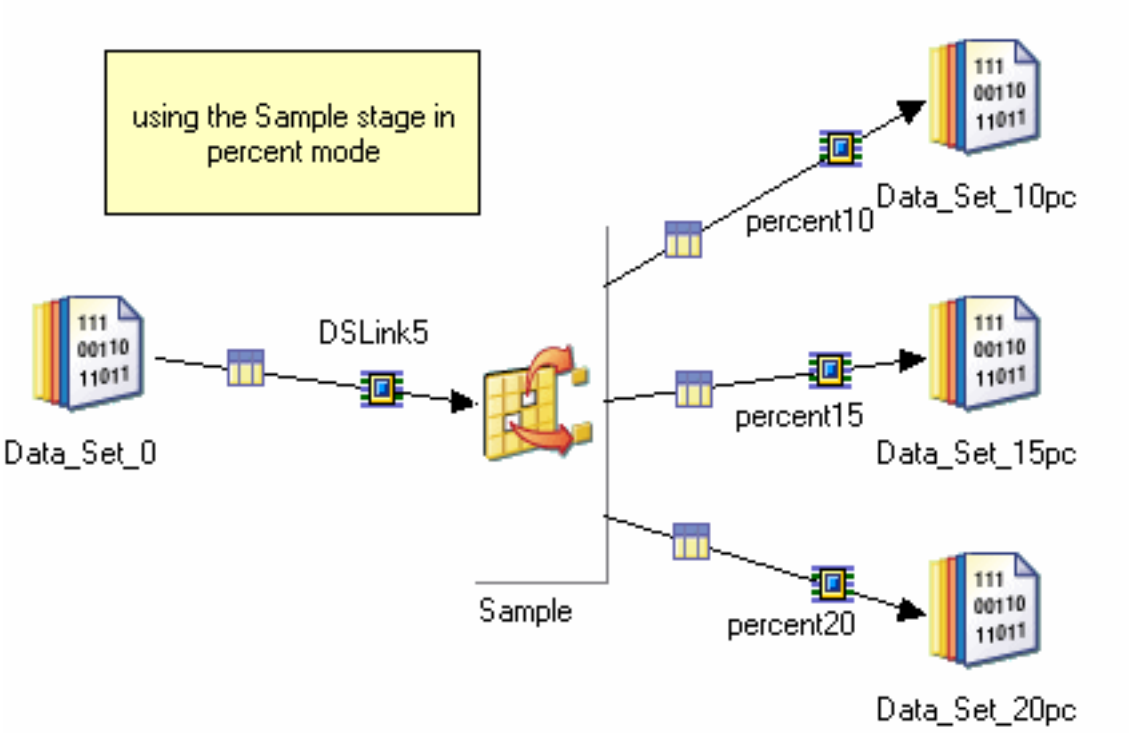
The right pane shows the output columns for each link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility.

Sample stage

The Sample stage is a Development/Debug stage. It can have a single input link and any number of output links when operationg in percent mode, or a single input and single output link when operating in period mode. It is one of a number of stages that InfoSphere DataStage provides to help you sample data, see also:

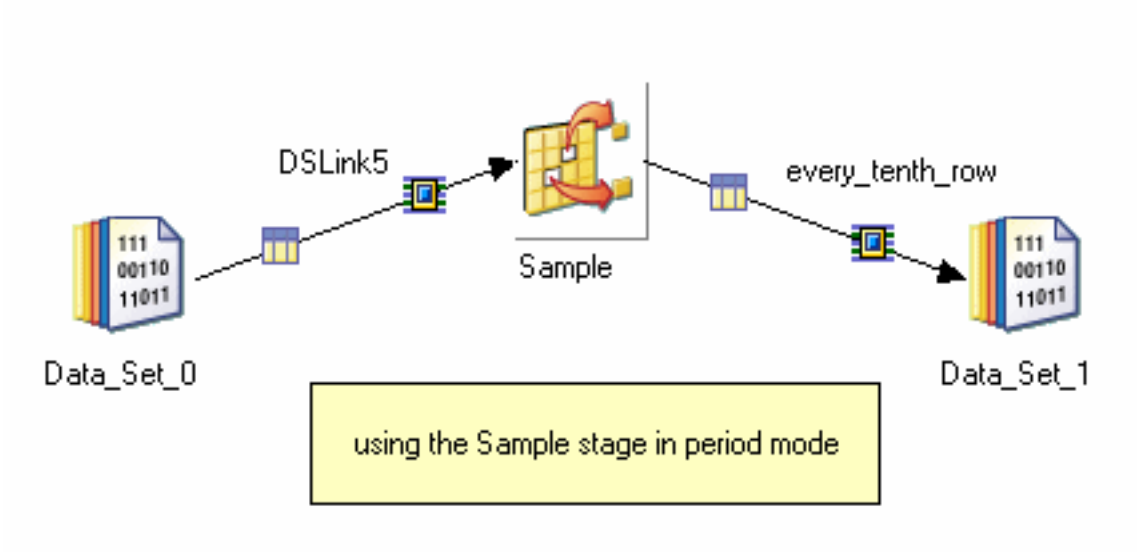
- Head stage, “Head stage” on page 502.
- Tail stage, “Tail stage” on page 508.
- Peek stage, “Peek stage” on page 522.

The Sample stage samples an input data set. It operates in two modes. In Percent mode, it extracts rows, selecting them by means of a random number generator, and writes a given percentage of these to each output data set. You specify the number of output data sets, the percentage written to each, and a seed value to start the random number generator. You can reproduce a given distribution by repeating the same number of outputs, the percentage, and the seed value



.

In Period mode, it extracts every *N*th row from each partition, where *N* is the period, which you supply. In this case all rows will be output to a single data set, so the stage used in this mode can only have a single output link



For both modes you can specify the maximum number of rows that you want to sample from each partition.

The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify details about the data set being Sampled.

- **Output Page**. This is where you specify details about the Sampled data being output from the stage.

Examples

Sampling in percent mode:

The input data set comprises billing information about GlobalCo's customers, which has previously been hash-partititioned into four partitions. You are going to take three samples, one of 10%, one of 15%, and one of 20%, and write these to three different files.

In the Stage page Properties tab you specify which percentages are written to which outputs as follows:

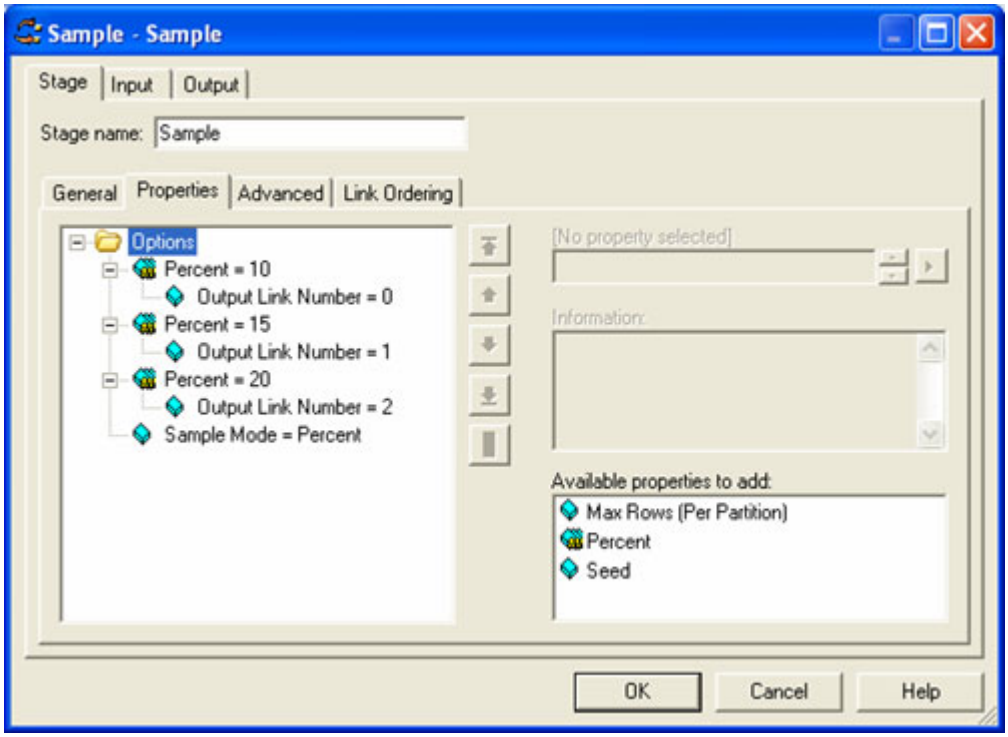


Figure 33. Properties tab

You use the **Link Ordering** tab to specify which outputs relate to which output links:

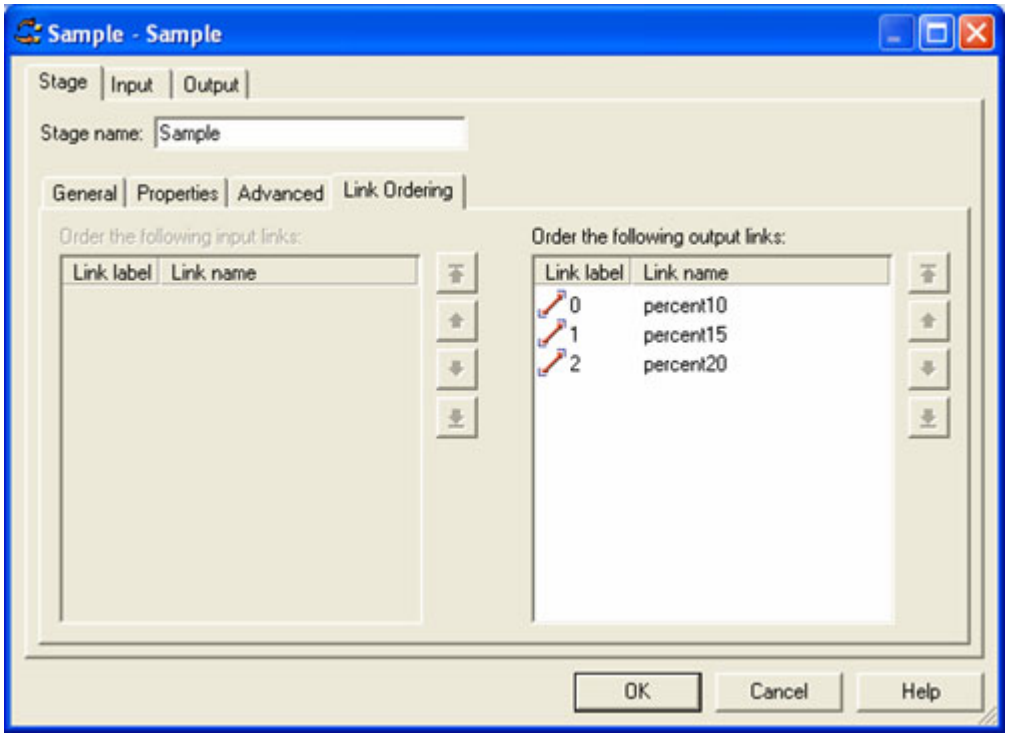


Figure 34. Link Ordering tab

When you run the job you get three data sets of different sizes as shown in the following tables (you would see this information if you looked at the data sets using the Data Set Manager):

Table 127. 10 Percent Sample

#	Node	Records	Blocks	Bytes
0	node1	27	2	34194
1	node2	21	1	26586
2	node3	18	1	22788
3	node4	21	1	26598

Table 128. 15 Percent Sample

#	Node	Records	Blocks	Bytes
0	node1	12	1	15192
1	node2	10	1	12660
2	node3	13	1	16458
3	node4	16	1	20256

Table 129. 20Percent Sample

#	Node	Records	Blocks	Bytes
0	node1	26	2	32928
1	node2	29	2	36738
2	node3	32	2	40527
3	node4	29	2	36714

Sampling in Period Mode:

In this example we are going to extract every twentieth row from each partition, up to a maximum of forty rows (in paractice the example data set is not large enough to reach this maximum). In period mode, you are limited to sampling into a single data set.

In the Stage page **Properties** tab we specify the period sample as follows:

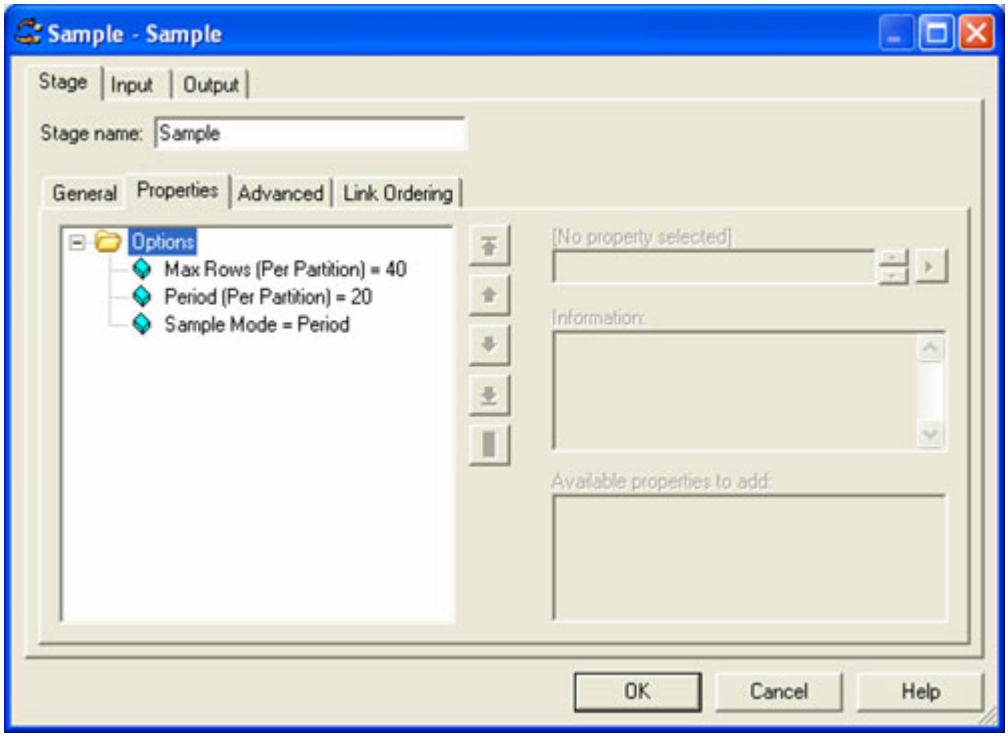


Figure 35. Properties tab

Because there can only be one output from the stage, we do not need to bother with the **Link Ordering** tab.

When you run the job it produces a sample from each partion. Here is the data sampled from partition 0:

Table 130. Partition 1 data

DocumentID	Year	Fname	Sname	Office
WBR 96 folio 34	1627	Guye	Beckley	none
(court leet attendees)	1687	Thomas	Cisill	none
WBR 96 folio 34	1627	John	Dizell	freeman
WBR 96 folio 1-46	1662	Gyles	Franklyn	councillor
WBR 96 folio 1-46	1662	John	Hawkins	none
WBR 96 folio 34	1627	Edward	Johnson	freeman
WBR 96 folio 21	1622	William	Neele	none
WBR 96 folio 11	1617	Thomas	Paynter	freeman
WBR 96 folio 1-46	1662	James	Ryman	freeman
WBR 96 folio 21	1622	Mathew	Tomes	freeman

Table 130. Partition 1 data (continued)

DocumentID	Year	Fname	Sname	Office
WBR 96 folio 21	1622	John	Woodroffe	none

Sample stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Sample stages in a job. This section specifies the minimum steps to take to get a Sample stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Sample stage:

- In the Stage page **Properties Tab**, choose the sample mode. This is Percent by default, but you can also choose Period.

  If you have chosen Percent, Specify the sampling percentage for an output link, and specify the output link number it will be output on (links are numbered from 0). Repeat these properties to specify the percentage for each of your output links.

  If you have chosen the Period mode, specify the Period. This will sample every *N*th row in each partition.
- If you have chosen Percent mode, in the Stage page **Link Ordering Tab**, specify which of your actual output links corresponds to link 0, link 1 and so on.
- In the Output Page **Mapping Tab**, specify how output columns on each link are derived from the columns of the input data.

Sample stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes. TheLink Ordering tab allows you to specify which output links are which.

Sample stage: Properties tab:

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 131. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Sample Mode	percent/period	percent	Y	N	N/A
Options/Percent	number	N/A	Y (if Sample Mode = Percent)	Y	N/A

Table 131. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Output Link Number	number	N/A	Y	N	Percent
Options/Seed	number	N/A	N	N	N/A
Options/Period (Per Partition)	number	N/A	Y (if Sample Mode = Period)	N	N/A
Options/Max Rows Per Partition	number	N/A	N	N	N/A

Sample stage: Options category:
Sample mode

Specifies the type of sample operation. You can sample on a percentage of input rows (percent), or you can sample the *N*th row of every partition (period).

Percent

Specifies the sampling percentage for each output data set when use a Sample Mode of Percent. You can repeat this property to specify different percentages for each output data set. The sum of the percentages specified for all output data sets cannot exceed 100%. You can specify a job parameter if required.

Percent has a dependent property:

- Output Link Number

  This specifies the output link to which the percentage corresponds. You can specify a job parameter if required.

Seed

This is the number used to initialize the random number generator. You can specify a job parameter if required. This property is only available if Sample Mode is set to percent.

Period (per partition)

Specifies the period when using a Sample Mode of Period.

Max rows per partition

This specifies the maximum number of rows that will be sampled from each partition.

Sample stage: Advanced tab:

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.

- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request the next stage should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Sample stage: Link Ordering tab:

In Percent mode, this tab allows you to specify the order in which the output links are processed. This is how they correspond to the Output Link Number properties on the **Properties Tab**.

By default the output links will be processed in the order they were added. To rearrange them, choose an output link and click the up arrow button or the down arrow button.

Sample stage: Input page

The Input page allows you to specify details about the data set being sampled. There is only one input link.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data.

Details about Sample stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Sample stage: Partitioning on input links:

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the sample is performed.

By default the stage uses the auto partitioning method. If the Preserve Partitioning option has been set on the previous stage in the job, the stage will warn if it cannot preserve the partitioning of the incoming data.

If the Sample stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Sample stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Sample stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Sample stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Sample stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Sample stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the sample is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Sample stage: Output page

The Output page allows you to specify details about data output from the Sample stage. In Percent mode, the stage can have any number of output links, in Period mode it can only have one output. Choose the link you want to work on from the Output Link drop down list.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of outgoing data. The Mapping tab allows you to specify the relationship between the columns being input to the Sample stage and the output columns. The Advanced tab allows you to change the default buffering settings for the output links. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Sample stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Sample stage: Mapping tab:

For Sample stages the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them.

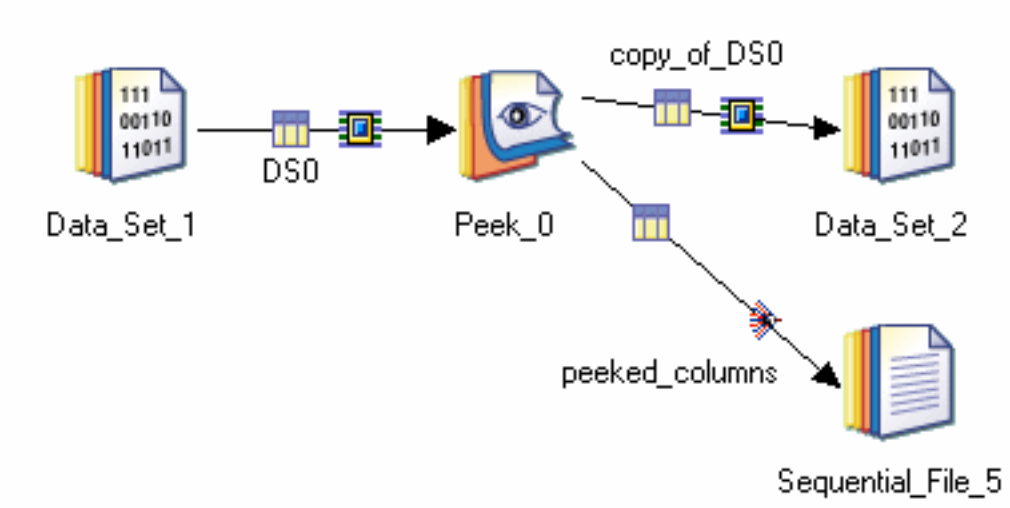
The left pane shows the columns of the sampled data. These are read only and cannot be modified on this tab. This shows the meta data from the incoming link

The right pane shows the output columns for the output link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility.

Peek stage

The Peek stage is a Development/Debug stage. It can have a single input link and any number of output links.

The Peek stage lets you print record column values either to the job log or to a separate output link as the stage copies records from its input data set to one or more output data sets. Like the Head stage (“Head stage” on page 502) and the Tail stage (“Sample stage” on page 513), the Peek stage can be helpful for monitoring the progress of your application or to diagnose a bug in your application.



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is where you specify the details about the single input set from which you are selecting records.
- **Output Page**. This is where you specify details about the processed data being output from the stage.

Peek stage: fast path
About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Peek stages in a job. This section specifies the minimum steps to take to get a Peek stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Peek stage:

- In the Stage page **Properties Tab**, check that the default settings are suitable for your requirements.
- In the Stage page **Link Ordering Tab**, if you have chosen to output peeked records to a link rather than the job log, choose which output link will carry the peeked records.

Peek stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Peek stage: Properties tab:

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 132. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Rows/All Records (After Skip)	True/False	False	N	N	N/A
Rows/Number of Records (Per Partition)	number	10	Y	N	N/A
Rows/Period (per Partition)	Number	N/A	N	N	N/A
Rows/Skip (per Partition)	Number	N/A	N	N	N/A
Columns/Peek All Input Columns	True/False	True	Y	N	N/A
Columns/Input Column to Peek	Input Column	N/A	Y (if Peek All Input Columns = False)	Y	N/A
Partitions/All Partitions	True/False	True	Y	N	N/A

Table 132. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Partitions/ Partition Number	number	N/A	Y (if All Partitions = False)	Y	N/A
Options/Peek Records Output Mode	Job Log/Output	Job Log	N	N	N/A
Options/Show Column Names	True/False	True	N	N	N/A
Options/ Delimiter String	space/nl/tab	space	N	N	N/A

Peek stage: Rows category:
All records (after skip)

True to print all records from each partition. Set to False by default.

Number of records (per partition)

Specifies the number of records to print from each partition. The default is 10.

Period (per partition)

Print every *P*th record in a partition, where *P* is the period. You can start the copy operation after records have been skipped by using the Skip property. *P* must equal or be greater than 1.

Skip (per partition)

Ignore the first *number* of rows of each partition of the input data set, where *number* is the number of rows to skip. The default skip count is 0.

Peek stage: Columns category:
Peek all input columns

True by default and prints all the input columns. Set to False to specify that only selected columns will be printed and specify these columns using the Input Column to Peek property.

Input column to peek

If you have set Peek All Input Columns to False, use this property to specify a column to be printed. Repeat the property to specify multiple columns.

Peek stage: Partitions category:
All partitions

Set to True by default. Set to False to specify that only certain partitions should have columns printed, and specify which partitions using the Partition Number property.

Partition number

If you have set All Partitions to False, use this property to specify which partition you want to print columns from. Repeat the property to specify multiple columns.

Peek stage: Options category:
Peek records output mode

Specifies whether the output should go to an output column (the Peek Records column) or to the job log.

Show column names

If True, causes the stage to print the column name, followed by a colon, followed by the column value. If False, the stage prints only the column value, followed by a space. It is True by default.

Delimiter string

The string to use as a delimiter on columns. Can be space, tab or newline. The default is space.

Peek stage: Advanced tab:

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. It adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request that next stage in the job should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Peek stage: Link Ordering tab:

This tab allows you to specify which output link carries the peek records data set if you have chosen to output the records to a link rather than the job log.

By default the last link added will represent the peek data set. To rearrange the links, choose an output link and click the up arrow button or the down arrow button.

Peek stage: Input page

The Input page allows you to specify details about the incoming data sets. The Peek stage expects one incoming data set.

The General tab allows you to specify an optional description of the input link. The Partitioning tab allows you to specify how incoming data is partitioned before being peeked. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Peek stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Peek stage: Partitioning tab:

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before it is peeked. It also allows you to specify that the data should be sorted before being operated on.

By default the stage partitions in Auto mode. This attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. If the Preserve Partitioning option has been set on the previous stage in the job, this stage will warn if it cannot preserve the partitioning of the incoming data.

If the Peek stage is operating in sequential mode, it will first collect the data using the default Auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Peek stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Peek stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Peek stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method of the Peek stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best collection method depending on execution modes of current and preceding stages, and how many nodes are specified in the Configuration file. This is the default collection method for Peek stages.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operator starts over.

- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the Available list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before being peeked. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available with the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the Available list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Peek stage: Output page

The Output page allows you to specify details about data output from the Peek stage. The Peek stage can have any number of output links. Select the link whose details you are looking at from the **Output name** drop-down list.

The General tab allows you to specify an optional description of the output link. The Columns tab specifies the column definitions of the data. The Mapping tab allows you to specify the relationship between the columns being input to the Peek stage and the Output columns. The Advanced tab allows you to change the default buffering settings for the output links.

Details about Peek stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Peek stage: Mapping tab:

For the Peek stage the Mapping tab allows you to specify how the output columns are derived, that is, what input columns map onto them or how they are generated.

The left pane shows the columns being peeked. These are read only and cannot be modified on this tab.

The right pane shows the output columns for each link. This has a **Derivations** field where you can specify how the column is derived. You can fill it in by dragging input columns over, or by using the Auto-match facility.

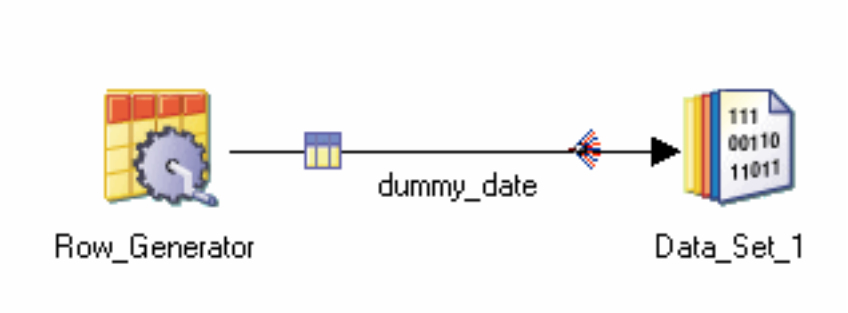
Row Generator stage

The Row Generator stage is a Development/Debug stage. It has no input links, and a single output link.

The Row Generator stage produces a set of mock data fitting the specified meta data. This is useful where you want to test your job but have no real data available to process. (See also the Column Generator stage which allows you to add extra columns to existing data sets, “Column Generator stage” on page 533.)

The meta data you specify on the output link determines the columns you are generating.

For decimal values the Row Generator stage uses dfloat. As a result, the generated values are subject to the approximate nature of floating point numbers. Not all of the values in the valid range of a floating point number are representable. The further a value is from zero, the greater the number of significant digits, the wider the gaps between representable values.



The stage editor has two pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Output Page**. This is where you specify details about the generated data being output from the stage.

Examples

Using a Row Generator Stage in Default Mode:

In this example you are going to allow the Row Generator stage to generate a data set using default settings for the data types. The only change you make is to ask for 100 rows to be generated, rather than the default ten. You do this by setting the Number of records property to 100 in the Properties tab.

You need to tell the stage how many columns in the generated data set and what type each column has. You do this in the Output page Columns tab by specifying the following column definitions:

Table 133. Column definitions

Column name	SQL type
string	Char
date	Date
time	Time
timestamp	Char
integer	integer
decimal	Decimal

When you run the job, InfoSphere DataStage generates a data set containing the following rows (sample shown):

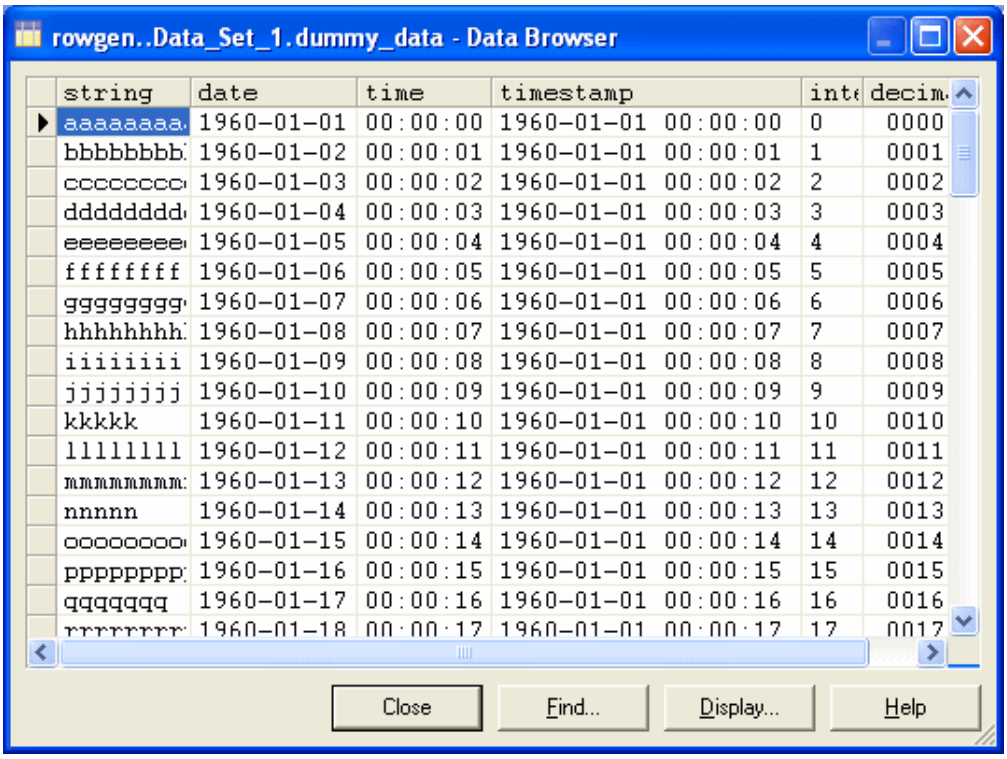


Figure 36. Sample data set

You can see from this the type of that is generated by default. For example, for date fields, the first row has January 1st 1960, and this is incremented by one day for each subsequent row.

You can specify more details about each data type if required to shape the data being generated.

Example of specifying data to be generated:

You can specify more details about the type of data being generated from the Edit Column Meta Data dialog box. This is accessed from the **Edit row...** shortcut menu for individual column definitions on the Output page Columns tab.

The Edit Column Meta Data dialog box contains different options for each data type. The possible options are described in "Generator" . You can use the **Next** and **<Previous** buttons to go through all the columns.

Using this dialog box you specify the following for the generated data:

- **string**
  - Algorithm = cycle
  - seven separate Values (assorted animals).
- **date**
  - Epoch = 1958-08-18
  - Type = cycle
  - Increment = 10
- **time**
  - Scale factor = 60
  - Type = cycle
  - Increment = 1

- **timestamp**
  - Epoch = 1958-08-18
  - Scale factor = 60
  - Type = cycle
  - Increment = 1
- **integer**
  - Type = cycle
  - Initial value = 300
  - Increment = 10
  - Limit = 3000
- **decimal**
  - Percent invalid = 20
  - Percent zeros = 20
  - Type = random
  - Seed=200

Here is the data generated by these settings, compare this with the data generated by the default settings.

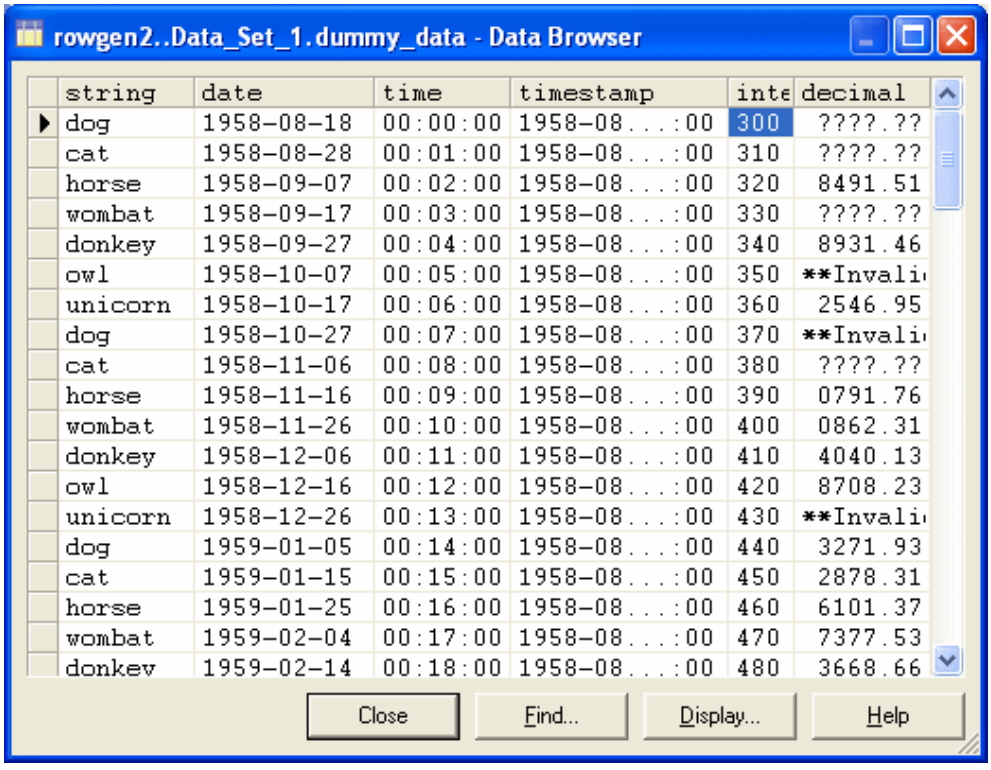


Figure 37. Generated data

Example of generating data in parallel:

By default the Row Generator stage runs sequentially, generating data in a single partition. You can, however, configure it to run in parallel, and you can use the partition number when you are generating data to, for example, increment a value by the number of partitions. You will also get the Number of Records you specify in each partition (so in the example where you have asked for 100 records, you will get 100 records in each partition rather than 100 records divided between the number of partitions).

In this example you are generating a data set comprising two integers. One is generated by cycling, one by random number generation.

The cycling integer's initial value is set to the partition number (using the special value `part') and its increment is set to the number of partitions (using the special value `partcount'). This is set in the Edit Column Meta Data dialog box as follows (select column in Columns tab and choose **Edit Row...** from shortcut menu):

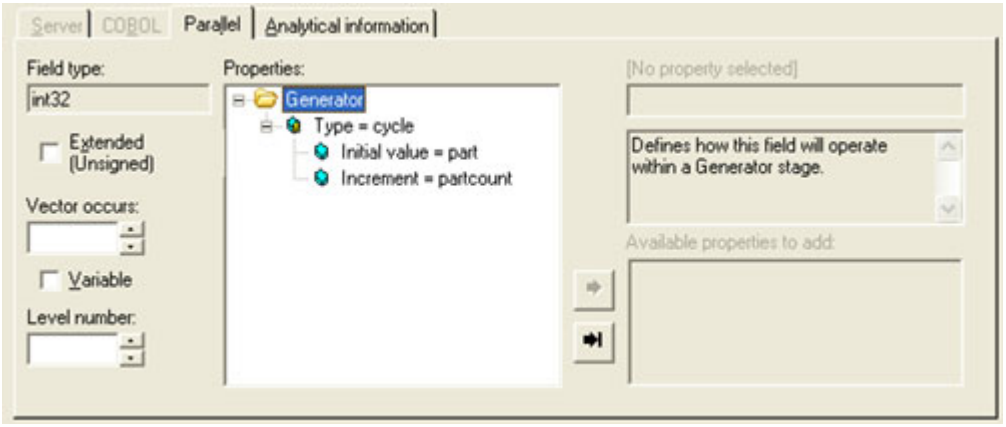


Figure 38. Generator settings

The random integer's seed value is set to the partition number, and the limit to the total number of partitions.

When you run this job in parallel, on a system with four nodes, the data generated in partition 0 is as follows:

Table 134. Partition 0 data

integer1	integer2
0	1
4	2
8	3
12	2
16	3
20	1
24	2
28	3
32	3
36	1
40	0
44	3
48	3
52	2
56	0
60	0
64	1

Table 134. Partition 0 data (continued)

integer1	integer2
68	3
72	2
76	2
80	0
84	1
...	...

Row Generator stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Row Generator stages in a job. This section specifies the minimum steps to take to get a Row Generator stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Row Generator stage:

- In the Stage page **Properties Tab**, specify the Number of Records you want to generate.
- Specify the meta data for the rows you want to generate. You can do this either in the Output page Columns Tab, or by specifying a schema file using the Schema File property on the Stage Page **Properties Tab**.

Row Generator stage: Stage page

The General tab allows you to specify an optional description of the stage. The Advanced tab allows you to specify how the stage executes.

Row Generator stage: Advanced tab:

This tab allows you to specify the following:

- **Execution Mode**. The Generate stage executes in Sequential mode by default. You can select Parallel mode to generate data sets in separate partitions.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. If you have an input data set, it adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request the next stage should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Row Generator stage: Output page

The Output page allows you to specify details about data output from the Row Generator stage.

The General tab allows you to specify an optional description of the output link. The Properties tab lets you specify what the stage does. The Columns tab specifies the column definitions of outgoing data. The Advanced tab allows you to change the default buffering settings for the output link.

Row Generator stage: Properties tab:

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them. The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 135. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Number of Records	number	10	Y	N	N/A
Options/Schema File	pathname	N/A	N	N	N/A

Row Generator stage: Options category:
Number of records

The number of records you want your generated data set to contain.

The default number is 10.

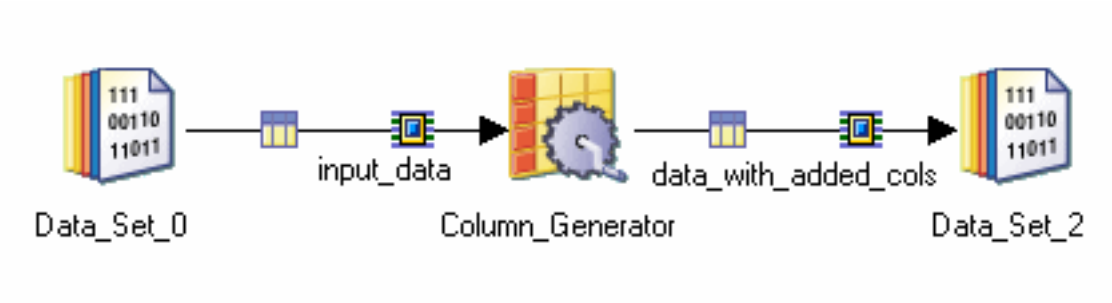
Schema file

By default the stage will take the meta data defined on the output link to base the mock data set on. But you can specify the column definitions in a schema file, if required. You can browse for the schema file or specify a job parameter.

Column Generator stage

The Column Generator stage is a Development/Debug stage. It can have a single input link and a single output link.

The Column Generator stage adds columns to incoming data and generates mock data for these columns for each data row processed. The new data set is then output. (See also the Row Generator stage which allows you to generate complete sets of mock data, “Row Generator stage” on page 527.)



The stage editor has three pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.

- **Input Page**. This is where you specify details about the input link.
- **Output Page**. This is where you specify details about the generated data being output from the stage.

Column generator stage example

For the example you are going to generate an extra column for a data set containing a list of seventeenth-century inhabitants of Woodstock, Oxfordshire. The extra column will contain a unique id for each row.

The columns for the data input to the Column Generator stage is as follows:

Table 136. Input data

Column name	SQL type	Length
DocumentID	VarChar	50
Year	VarChar	50
Orderno	Integer	10
Fname	VarChar	50
Sname	VarChar	50
Office	VarChar	50

You set the Column Generator properties to add an extra column called uniqueid to the data set as follows:

- Column Method = Explicit
- Column To Generate = uniqueid

The new column now appears on the Output page Mapping tab and can be mapped across to the output link (so it appears on the Output page Columns tab):

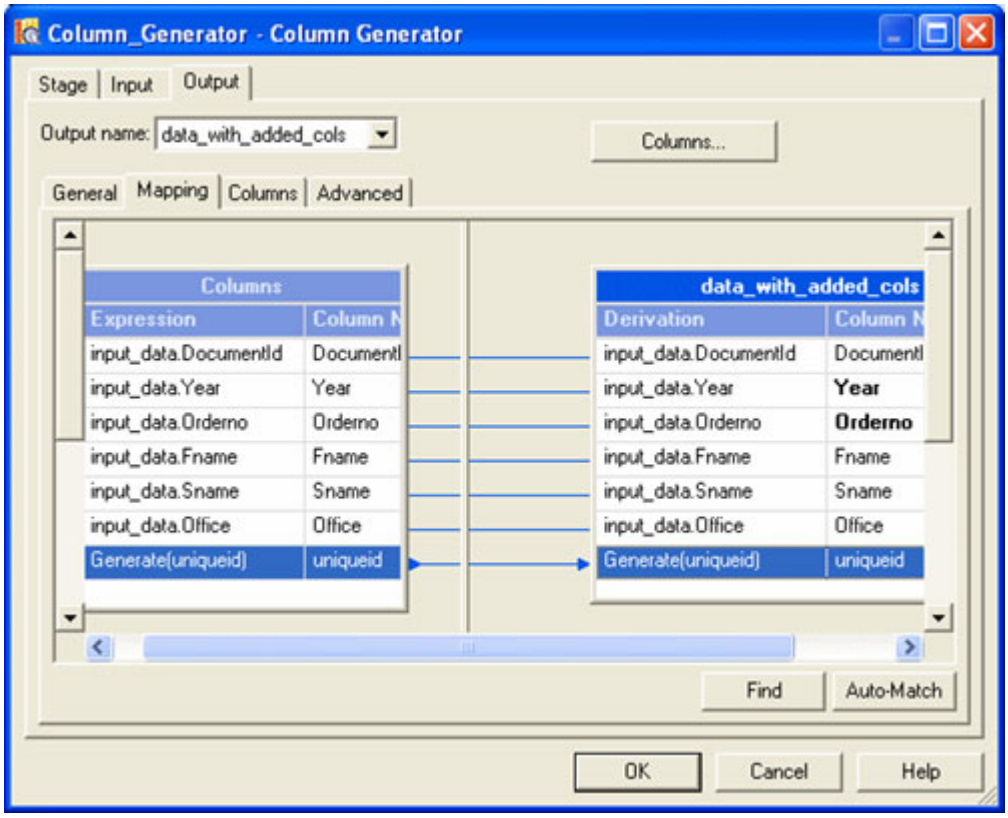


Figure 39. Mapping tab

In this example you select the uniqueid column on the Output page Columns tab, then choose **Edit Row...** from the shortcut menu. The Edit Column Meta Data dialog box appears and lets you specify more details about the data that will be generated for the new column. First you change the type from the default of char to integer. Because you are running the job in parallel, you want to ensure that the id you are generating will be unique across all partitions, to do this you set the initial value to the partition number (using the special value `part') and the increment to the number of partitions (using the special `partcount'):

When you run the job in parallel on a four-node system the stage will generate the uniqueid column for each row. Here are samples of partition 0 and partition 1 to show how the unique number is generated:

Table 137. Unique number generation for partition 0

uniqueid	DocumentID	Year	Orderno	Fname	Sname
0	WBR 96 folio 11	1617	110	Henry	Archer
4	WBR 96 folio 15	1619	115	Henry	Archer
8	WBR 96 folio 21	1622	117	Henry	Archer
12	WBR 96 folio 15	1619	95	John	Archer
16	WBR 96 folio 21	1622	82	John	Archer
20	WBR 96 folio 34	1627	95	John	Archer
24	WBR 96 folio 1-46	1662	125	Thomas	Archer
28	(court leet attendees)	1687	67	Thomas	Archer
32	Portmouth Book	1616	36	William	Archer
36	WBR 96 folio 11	1617	37	William	Archer

Table 137. Unique number generation for partition 0 (continued)

uniqueid	DocumentID	Year	Orderno	Fname	Sname
40	WBR 96 folio 1-46	1662	102	William	Ball
44	WBR 96 folio 11	1617	59	George	Barnsley
48	Portmouth Book	1616	35	John	Batt
52	Portmouth Book	1616	65	John	Batt
56	WBR 96 folio 11	1617	35	John	Batt
60	WBR 96 folio 15	1619	37	John	Batt
64	WBR 96 folio 15	1619	38	John	Batt
68	WBR 96 folio 15	1619	114	Gey	Becksley
72	WBR 96 folio 21	1622	116	Gey	Becksley
76	WBR 96 folio 34	1627	114	Guye	Becksley

Table 138. Unique number generation for partition 1

uniqueid	DocumentID	Year	Orderno	Fname	Sname
1	WBR 96 folio 21	1622	113	Marmad	?
5	WBR 96 folio 21	1622	68	Thomas	Asplene
9	(court leet attendees)	1687	83	Charles	Aston
13	WBR 96 folio 34	1627	107	Richard	Barnshooe
17	WBR 96 folio 34	1627	103	Walter	Bayless
21	(court leet attendees)	1687	3	Nicholas	Baynton
25	WBR 96 folio 21	1622	111	Robert	Belcher
29	WBR 96 folio 11	1617	102	Thomas	Belcher
33	WBR 96 folio 1-46	1662	24	Perrigrine	Bertie
37	WBR 96 folio 1-46	1662	16	John	Bignill
41	Portmouth Book	1616	16	Robert	Bignill
45	WBR 96 folio 1-46	1662	48	James	Blunden
49	(court leet attendees)	1687	66	William	Blunden
53	WBR 96 folio 1-46	1662	119	John	Borman
57	WBR 96 folio 11	1617	68	Henry	Bradshaw
61	WBR 96 folio 34	1627	88	William	Bradshaw
65	WBR 96 folio 21	1622	23	Robert	Brewce
69	WBR 96 folio 21	1622	21	Thomas	Brewce
73	WBR 96 folio 1-46	1662	84	Richard	Broffett
77	(court leet attendees)	1687	34	John	Brotherton

Column Generator stage: fast path

This section specifies the minimum steps to take to get a Column Generator stage functioning.

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Column Generator stages in a job. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

Procedure

1. In the Stage page **Properties Tab**, specify the Column Method. This is explicit by default, which means that you should specify the meta data for the columns you want to generate on the Output Page Columns tab. If you use the Explicit method, you also need to specify which of the output link columns you are generating in the Column to Generate property. You can repeat this property to specify multiple columns. If you use the Schema File method, you should specify the schema file.
2. Ensure you have specified the meta data for the columns you want to add. If you have specified a Column Method of explicit, you should do this on the Output Page Columns tab. If you have specified a Column Method of Schema File, you should specify a schema file.
3. In the Output Page **Mapping Tab**, specify how the incoming columns and generated columns map onto the output columns.

Column Generator stage: Stage page

The General tab allows you to specify an optional description of the stage. The Properties tab lets you specify what the stage does. The Advanced tab allows you to specify how the stage executes.

Column Generator stage: Properties tab:

The Properties tab allows you to specify properties which determine what the stage actually does. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 139. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/Column Method	Explicit/ Column Method	Explicit	Y	N	N/A
Options/Column to Generate	output column	N/A	Y	Y (if Column Method = Explicit)	N/A
Options/Schema File	pathname	N/A	N	Y (if Column Method = Schema File)	N/A

Column Generator stage: Options category:
Column method

Select Explicit if you are going to specify the column or columns you want the stage to generate data for. Select Schema File if you are supplying a schema file containing the column definitions.

Column to generate

When you have chosen a column method of Explicit, this property allows you to specify which output columns the stage is generating data for. Repeat the property to specify multiple columns. You can

specify the properties for each column using the Parallel tab of the Edit Column Meta Dialog box (accessible from the shortcut menu on the columns grid of the output Columns tab). You can use the Column Selection dialog box to specify several columns at once if required.

Schema file

When you have chosen a column method of schema file, this property allows you to specify the column definitions in a schema file. You can browse for the schema file or specify a job parameter.

Column Generator stage: Advanced tab:

This tab allows you to specify the following:

- **Execution Mode**. The stage can execute in parallel mode or sequential mode. In parallel mode the input data is processed by the available nodes as specified in the Configuration file, and by any node constraints specified on the Advanced tab. In Sequential mode the entire data set is processed by the conductor node.
- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Propagate** by default. If you have an input data set, it adopts **Set** or **Clear** from the previous stage. You can explicitly select **Set** or **Clear**. Select **Set** to request the next stage should attempt to maintain the partitioning.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Column Generator stage: Input page

The Input page allows you to specify details about the incoming data set you are adding generated columns to. There is only one input link and this is optional.

The General tab allows you to specify an optional description of the link. The Partitioning tab allows you to specify how incoming data on the source data set link is partitioned. The Columns tab specifies the column definitions of incoming data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Generate stage partitioning are given in the following section. See "Stage Editors," for a general description of the other tabs.

Column Generator stage: Partitioning on input links:

The Partitioning tab allows you to specify details about how the incoming data is partitioned or collected before the generate is performed.

By default the stage uses the auto partitioning method.

If the Column Generator stage is operating in sequential mode, it will first collect the data before writing it to the file using the default auto collection method.

The Partitioning tab allows you to override this default behavior. The exact operation of this tab depends on:

- Whether the Column Generator stage is set to execute in parallel or sequential mode.
- Whether the preceding stage in the job is set to execute in parallel or sequential mode.

If the Column Generator stage is set to execute in parallel, then you can set a partitioning method by selecting from the **Partition type** drop-down list. This will override any current partitioning.

If the Column Generator stage is set to execute in sequential mode, but the preceding stage is executing in parallel, then you can set a collection method from the **Collector type** drop-down list. This will override the default auto collection method.

The following partitioning methods are available:

- **(Auto)**. InfoSphere DataStage attempts to work out the best partitioning method depending on execution modes of current and preceding stages and how many nodes are specified in the Configuration file. This is the default method for the Column Generator stage.
- **Entire**. Each file written to receives the entire data set.
- **Hash**. The records are hashed into partitions based on the value of a key column or columns selected from the **Available** list.
- **Modulus**. The records are partitioned using a modulus function on the key column selected from the **Available** list. This is commonly used to partition on tag fields.
- **Random**. The records are partitioned randomly, based on the output of a random number generator.
- **Round Robin**. The records are partitioned on a round robin basis as they enter the stage.
- **Same**. Preserves the partitioning already in place.
- **DB2**. Replicates the DB2 partitioning method of a specific DB2 table. Requires extra properties to be set. Access these properties by clicking the properties button.
- **Range**. 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. Requires extra properties to be set. Access these properties by clicking the properties button.

The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Column Generator stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operation starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the column generate operation is performed. The sort is always carried out within data partitions. If the stage is partitioning incoming data the sort occurs after the partitioning. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the partitioning or collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu.

Column Generator stage: Output page

Details about Column Generator stage mapping is given in the following section. See "Stage Editors," for a general description of the other tabs.

Column Generator stage: Mapping tab:

For Column Generator stages the Mapping tab allows you to specify how the output columns are derived, that is, how the generated data maps onto them.

The left pane shows the generated columns. These are read only and cannot be modified on this tab. These columns are automatically mapped onto the equivalent output columns.

The right pane shows the output columns for the output link. This has a **Derivations** field where you can specify how the column is derived.You can fill it in by dragging input columns over, or by using the Auto-match facility.

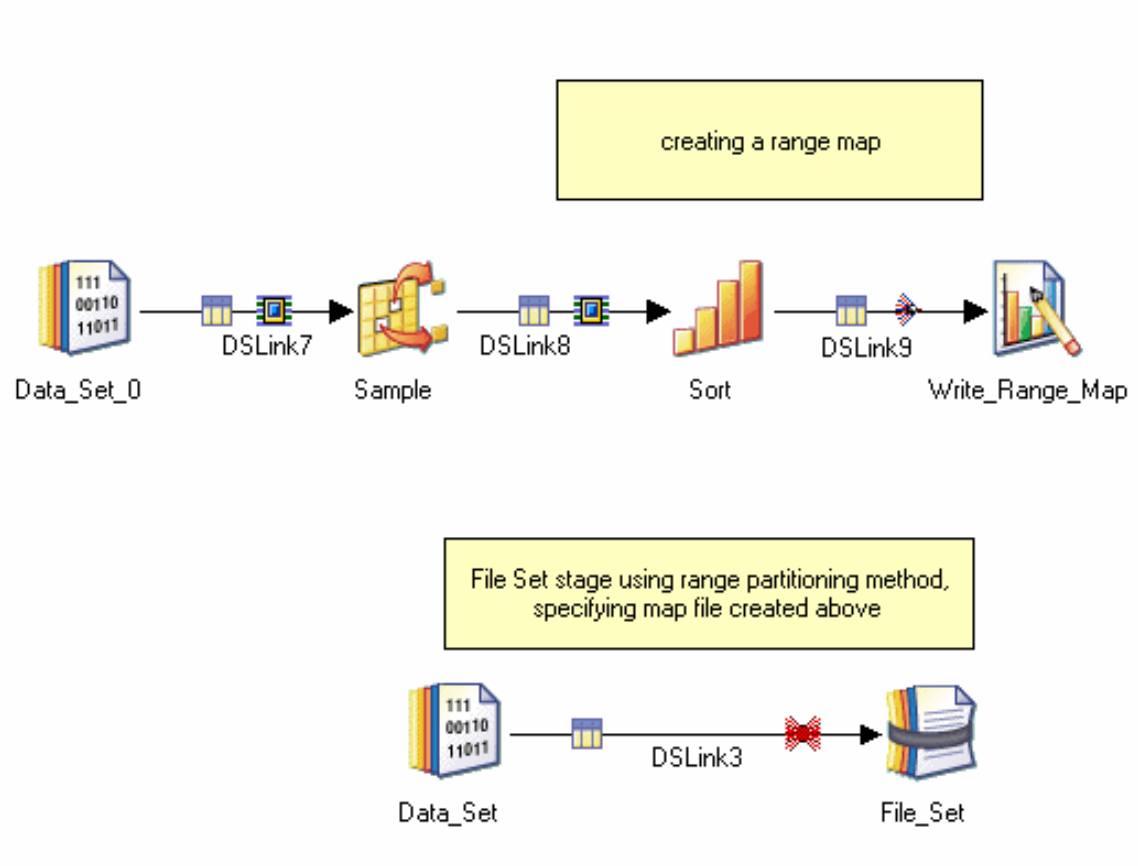
The right pane represents the data being output by the stage after the generate operation.

Write Range Map stage

The Write Range Map stage is a Development/Debug stage. It allows you to write data to a range map. The stage can have a single input link. It can only run in sequential mode.

The Write Range Map stage takes an input data set produced by sampling and sorting a data set and writes it to a file in a form usable by the range partitioning method. The range partitioning method uses the sampled and sorted data set to determine partition boundaries. .

A typical use for the Write Range Map stage would be in a job which used the Sample stage to sample a data set, the Sort stage to sort it and the Write Range Map stage to write the range map which can then be used with the range partitioning method to write the original data set to a file set.



The Write Range Map stage editor has two pages:

- **Stage Page**. This is always present and is used to specify general information about the stage.
- **Input Page**. This is present when you are writing a range map. This is where you specify details about the file being written to.

Example of the Write Range Map stage

In this example, you sample the data in a flat file then pass it to the Write Range Map stage. The stage sorts the data itself before constructing a range map and writing it to a file.

The stage sorts the data on the same key that it uses to create the range map. The diagram shows the properties that determine how the stage will produce the range map:

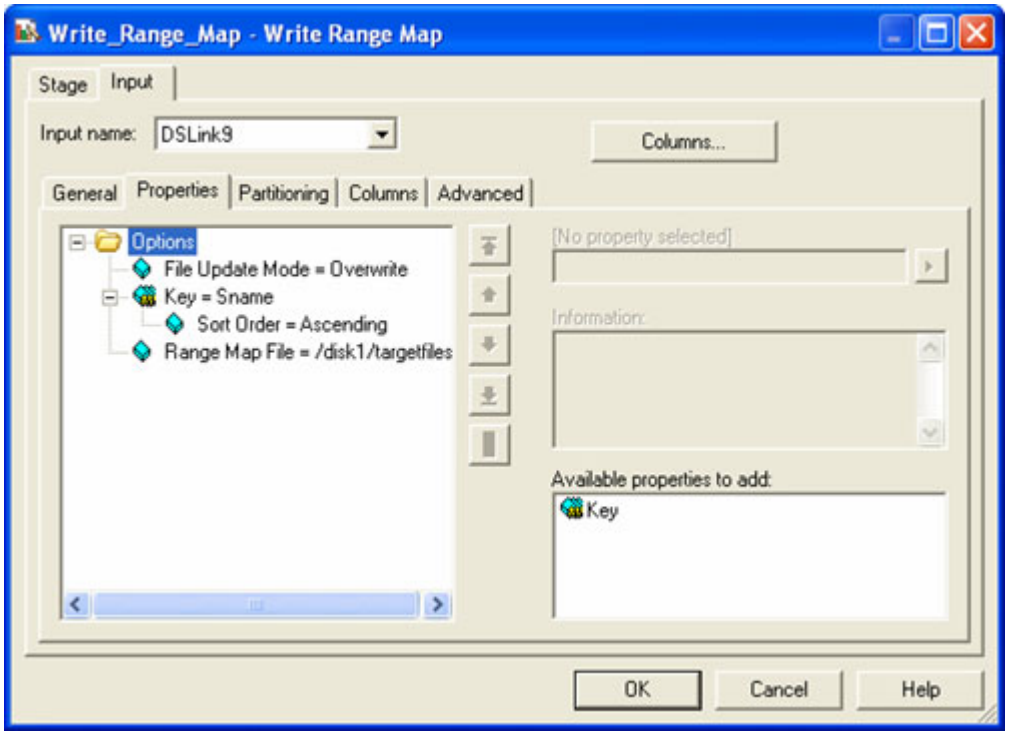


Figure 40. Property settings

Write Range Map stage: fast path

About this task

InfoSphere DataStage has many defaults which means that it can be very easy to include Write Range Map stages in a job. This section specifies the minimum steps to take to get a Write Range Map stage functioning. InfoSphere DataStage provides a versatile user interface, and there are many shortcuts to achieving a particular end, this section describes the basic method, you will learn where the shortcuts are when you get familiar with the product.

To use a Write Range Map stage:

- In the **Input Link Properties Tab**:
  - Specify the key column(s) for the range map you are creating.
  - Specify the name of the range map you are creating.
  - Specify whether it is OK to overwrite an existing range map of that name (be default an error occurs if a range map with that name already exists).
- Ensure that column definitions have been specified for the range map (this can be done in an earlier stage).

Write Range Map stage: Stage page

The General tab allows you to specify an optional description of the stage. The Advanced tab allows you to specify how the stage executes. The NLS Locale tab appears if your have NLS enabled on your system. It allows you to select a locale other than the project default to determine collating rules.

Write Range Map stage: Advanced tab:

This tab allows you to specify the following:

- **Execution Mode**. The stage always executes in sequential mode.

- **Combinability mode**. This is Auto by default, which allows InfoSphere DataStage to combine the operators that underlie parallel stages so that they run in the same process if it is sensible for this type of stage.
- **Preserve partitioning**. This is **Set** by default. The Partition type is range and cannot be overridden.
- **Node pool and resource constraints**. Select this option to constrain parallel execution to the node pool or pools or resource pool or pools specified in the grid. The grid allows you to make choices from drop down lists populated from the Configuration file.
- **Node map constraint**. Select this option to constrain parallel execution to the nodes in a defined node map. You can define a node map by typing node numbers into the text box or by clicking the browse button to open the Available Nodes dialog box and selecting nodes from there. You are effectively defining a new node pool for this stage (in addition to any node pools defined in the Configuration file).

Write Range Map stage: NLS Locale tab:

This appears if you have NLS enabled on your system. It lets you view the current default collate convention, and select a different one for this stage if required. You can also use a job parameter to specify the locale, or browse for a file that defines custom collate rules. The collate convention defines the order in which characters are collated. The Write Range Map stage uses this when it is determining the sort order for key columns. Select a locale from the list, or click the arrow button next to the list to use a job parameter or browse for a collate file.

Write Range Map stage: Input page

The Input page allows you to specify details about how the Write Range Map stage writes the range map to a file. The Write Range Map stage can have only one input link.

The General tab allows you to specify an optional description of the input link. The Properties tab allows you to specify details of exactly what the link does. The Partitioning tab allows you to view collecting details. The Columns tab specifies the column definitions of the data. The Advanced tab allows you to change the default buffering settings for the input link.

Details about Write Range Map stage properties and collecting are given in the following sections. See "Stage Editors," for a general description of the other tabs.

Write Range Map stage: Input Link Properties tab:

The Properties tab allows you to specify properties for the input link. These dictate how incoming data is written to the range map file. Some of the properties are mandatory, although many have default settings. Properties without default settings appear in the warning color (red by default) and turn black when you supply a value for them.

The following table gives a quick reference list of the properties and their attributes. A more detailed description of each property follows.

Table 140. Properties

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/File Update Mode	Create/Overwrite	Create	Y	N	N/A
Options/Key	input column	N/A	Y	Y	N/A
Options/	Ascending/ Descending	Ascending	N	N	Key
Options/	True/False	True	N	N	Key
Options/	First/Last	First	N	N	Key

Table 140. Properties (continued)

Category/ Property	Values	Default	Mandatory?	Repeats?	Dependent of
Options/	True/False	False	N	N	Key
Options/Range Map File	pathname	N/A	Y	N	N/A

Write Range Map stage: Options category:
File update mode

This is set to Create by default. If the file you specify already exists this will cause an error. Choose Overwrite to overwrite existing files.

Key

This allows you to specify the key for the range map. Choose an input column from the drop-down list. You can specify a composite key by specifying multiple key properties. You can use the Column Selection dialog box to select several keys at once if required). Key has the following dependent properties:

Sort Order
Choose Ascending or Descending. The default is Ascending.

Case Sensitive
This property is optional. Use this to specify whether each group key is case sensitive or not, this is set to True by default, that is, the values "CASE" and "case" would not be judged equivalent.

Nulls Position
This property is optional. By default columns containing null values appear first in the sorted data set. To override this default so that columns containing null values appear last in the sorted data set, select Last.

Sort as EBCDIC
To sort as in the EBCDIC character set, choose True.

Range map file

Specify the file that is to hold the range map. You can browse for a file or specify a job parameter.

Write Range Map stage: Partitioning tab:

The Partitioning tab normally allows you to specify details about how the incoming data is partitioned or collected before it is written to the file or files. In the case of the Write Range Map stage execution is always sequential, so there is never a need to set a partitioning method.

You can set a collection method if collection is required. The following Collection methods are available:

- **(Auto)**. This is the default collection method for the Write Range Map stage. Normally, when you are using Auto mode, InfoSphere DataStage will eagerly read any row from any input partition as it becomes available.
- **Ordered**. Reads all records from the first partition, then all records from the second partition, and so on.
- **Round Robin**. Reads a record from the first input partition, then from the second partition, and so on. After reaching the last partition, the operation starts over.
- **Sort Merge**. Reads records in an order based on one or more columns of the record. This requires you to select a collecting key column from the **Available** list.

The Partitioning tab also allows you to specify that data arriving on the input link should be sorted before the write range map operation is performed. If the stage is collecting data, the sort occurs before the collection. The availability of sorting depends on the collecting method chosen (it is not available for the default auto methods).

Select the check boxes as follows:

- **Perform Sort**. Select this to specify that data coming in on the link should be sorted. Select the column or columns to sort on from the **Available** list.
- **Stable**. Select this if you want to preserve previously sorted data sets. This is the default.
- **Unique**. Select this to specify that, if multiple records have identical sorting key values, only one record is retained. If stable sort is also set, the first record is retained.

If NLS is enabled an additional button opens a dialog box allowing you to select a locale specifying the collate convention for the sort.

You can also specify sort direction, case sensitivity, whether sorted as ASCII or EBCDIC, and whether null columns will appear first or last for each column. Where you are using a keyed partitioning method, you can also specify whether the column is used as a key for sorting, for partitioning, or for both. Select the column in the **Selected** list and right-click to invoke the shortcut menu. Because the partition mode is set and cannot be overridden, you cannot use the stage sort facilities, so these are disabled.

Chapter 10. Viewing the job log

View the entries in the job log when you run your current job within the IBM InfoSphere DataStage and QualityStage Designer client.

About this task

You can view the entries in the job log from within the Designer client, when you either run a job or run a job in debug mode.

Procedure

1. Click **View** > **Job Log**.
2. To choose how the job log entries are displayed, click **View Options** on the toolbar on the Job Log pane.
3. To see the latest job log entries, click **Refresh** on the toolbar.
4. To choose whether to show all job log entries or only warning and fatal job log entries, click **Filter Events** on the toolbar.
5. Double-click a job entry to see full details of the event.
6. Click **Reset to most recent log** on the toolbar to remove job log entries that do not belong to the most recent job.

Chapter 11. Introduction to InfoSphere DataStage Balanced Optimization

You can use Balanced Optimization to improve the performance of some IBM InfoSphere DataStage jobs.

You can optimize parallel jobs that use Teradata, IBM DB2, or Oracle connectors to connect to Teradata, IBM DB2, or Oracle databases.

Balanced Optimization is a licensed add-on to InfoSphere DataStage.

InfoSphere DataStage jobs provide connectivity, data manipulation functionality, and highly scalable performance. The InfoSphere DataStage visual flow-design paradigm is easy to use when designing simple-to-complex data integration jobs. Better performance might be achieved, however, if the processing load can be shared or redistributed among InfoSphere DataStage and the source or target databases. You can control where the intensive work is done: in source databases, in InfoSphere DataStage, or in target databases.

For job designs that use connectors to read or write data from data sources, you can use Balanced Optimization to give you greater control over the job. You design your job as normal, then use Balanced Optimization to redesign the job automatically to your stated preferences. This redesign process can maximize performance by minimizing the amount of input and output performed, and by balancing the processing against source, intermediate, and target environments. You can then examine the new optimized job design, and save it as a new job. Your root job design remains unchanged. The Balanced Optimization enables you to take advantage of the power of the databases without becoming an expert in native SQL.

The following principles can lead to the better performance of parallel jobs:

Minimize I/O and data movement
: Reduce the amount of source data read by the job by performing computations within the source database. Where possible, move processing of data to the database and avoid extracting data just to process it and write it back to the same database.

Maximize optimization within source or target databases
: Make use of the highly developed optimizations that databases achieve by using local indexes, statistics, and other specialized features.

Maximize parallelism
: Take advantage of default InfoSphere DataStage behavior when reading and writing databases: use parallel interfaces and pipe the data through the job, so that data flows from source to target without being written to intermediate destinations.

Balanced Optimization uses these principles to improve the potential performance of a job. You influence the job redesign by setting options within the tool to specify which of the principles are followed.

Balanced Optimization does not change or optimize machine configurations, InfoSphere DataStage configurations, or database configurations

Job optimization

Optimization pushes processing functionality and related data I/O into database sources or targets, depending on the optimization options that you choose.

When you optimize a job, Balanced Optimization searches the job for patterns of stages, links, and property settings. The patterns typically include one or more of the supported database connector stages (Teradata, DB2, or Oracle). When a candidate pattern is found, Balanced Optimization combines the processing into the corresponding source or target database SQL and removes or replaces any stages and links that are no longer needed. The tool then adjusts the remaining stages and links. After the tool has found a pattern and modified the job design, it repeats the process. Optimization terminates when none of the patterns match anything further in the optimized job, which indicates that there is no more work to be done.

Generally, optimizations are performed in a priority order. When there is ambiguity (for example, some processing could be performed either in a source or target database) processing is pushed into database targets.

Optimization scenarios

There are some common scenarios for which Balanced Optimization is suitable.

Using Balanced Optimization in the following scenarios can improve the performance of your job.

Convert a job to use bulk staging tables
: This optimization can improve performance where you have a large volume of data. Any target connector whose mode properties include bulk staging are converted to bulk INSERT data into a temporary staging table in the target database. Post-processing SQL is added to move the data from the staging table into the real target table. If you want the staging table to be created in a different target database space from the real target table, then you can specify the space in a property.

Do processing, joining, and lookups in data targets
: This optimization pushes as much of the work as possible into the target database, including when a lookup or join source table is already in the target database. You can optionally also implement bulk INSERT into a staging table. If the work pushed into the target database involves data reduction (for example, the job contains an Aggregator stage, or a Transformer stage with a constraint expression), you can also choose to do data reduction in data sources.

Do data reduction in data sources
: This optimization can improve performance where your job has an Aggregator stage, a Transformer stage with a constraint that would discard many rows, a Remove Duplicates stage, or a Sort stage. This optimization pushes as much processing as possible into database source connectors.

Do joining and lookups in data sources.
: This optimization pushes Join and Lookup stages involving data from the same database server (although possibly different database instances) into the source database. This optimization can also save extra sorting that is often performed implicitly by the Join stage.

Balance the work between source databases, the InfoSphere Information Server engine, and target databases
: This optimization pushes as much work as possible to the target database or databases, then as much of the remaining work into the source database or databases, leaving the rest of the work in the job.

Push all the work into the database
: If all the data used by your job is on the same database server, and all processing in your job can be performed inside the target database, you can save all the database I/O and have all the processing run as SQL inside the target database.

If you want to specifically limit optimization to include only a section of your job design, you can set the **Name of a stage where optimization should stop** property. No processing is pushed to source or target databases beyond the stage named in this property.

Balanced Optimization process overview

The process overview sets out the way to get the best results when using Balanced Optimization.

The process comprises these steps:

1. Design an InfoSphere DataStage job to perform the required processing. Follow these rules in your design:
   - Express your logic in the native InfoSphere DataStage forms of stages, links, and expressions that are supported by the visual design canvas.
   - Use the default values for stage properties wherever possible.
   - Use simple target SQL queries that are generated by the stage where possible. Express logic in multiple stages instead of specifying more complex SQL in a single stage (Balanced Optimization processes the job and produces the complex SQL for you).
2. Compile and run the job with a representative set of source and target data, and verify the results. Be sure to run the job in a similar environment, including multi-node parallelism, that you will run the optimized job in. You must compile the job before trying to optimize it; compilation ensures that only valid jobs are submitted for optimization. Run the job to obtain performance and resource utilization baseline data.
3. Examine the job log of the root job to locate any warnings or errors. Some warnings about items like type or precision or length mismatches, duplicate updates, and nulls and non-null columns can lead to problems or errors in an optimized job, and these issues must be resolved before you optimize the job.
4. Select your job design in the Designer client, and select **File** > **Optimize**
5. In the Balanced Optimization window, select your optimization options, and supply any optimization properties that are required.
6. Generate an optimized job, and then save it.
7. In the Designer client, examine the optimized job to see how the job design has changed.
8. Compile the optimized job and then run it with the same data that you used when you verified the root job. Ensure that both jobs are run under the same conditions. For example, if your root job created a table, be sure to delete that table before you run the optimized job.
9. Compare the results of the optimized job to the root job and see if there are improvements in performance and resource utilization. Use the performance analysis tools in the Designer client to help you.

You can follow these steps in an iterative process. Start with default optimization options, optimize your job and obtain performance statistics. Then change the optimization options and optimize the root job again. Repeat this process until you are satisfied with your job optimization.

Relationship between root job and optimized job

IBM InfoSphere DataStage maintains a relationship between a job, and any optimized versions of that job.

There is a two-way relationship between a job and any optimized versions of that job. When an optimized job is created, it is related to its root job, and the relationships can be queried in both directions. Starting from the root job, you can show the names of all its optimized jobs, and from an optimized job you can show the name of its root job.

You can compile and run the root job without the optimized versions being present, but if you perform a where used query on the root job, the query returns a list of optimized versions of that job. An optimized job requires the root job to be present only if you want to rerun the optimization, but a dependent of query on the optimized job includes the root job in its results.

You can edit an optimized job without destroying its relationship to the root job. However, any changes made to the optimized job will not be applied back to the root job, and are lost if the job is reoptimized. You can use the timestamp of the optimized job to detect if it has been modified since it was created from the root job.

The root job can also be edited while it has optimized jobs related to it. Changes to the root job do not automatically update the optimized jobs. Optimized jobs need to be regenerated to incorporate changes to the root job.

Even when related to a root job, an optimized job can be compiled, run, and deployed separately from the root job. The root job can also be compiled, run, and deployed separately if required.

You cannot open an optimized job and optimize it while it is still related to the root job.

You receive an impact analysis warning if you attempt to delete a root job or rename a root job while it is still related to optimized jobs. You do not receive a warning if you delete optimized jobs that are still related to their root job.

Balanced Optimization options and properties

You can influence how your jobs are optimized by setting options and properties in the Balanced Optimization window.

You use optimization options to control the optimization process. Only options meaningful to the particular job being optimized are displayed. Optimization attempts to perform optimizations according to the options that you select. If the database does not support some functionality in the job, then that functionality cannot be pushed to the database.

Properties supply values that the optimization procedure can use. For example, if you want to create a staging table in a separate Teradata database instance, or DB2 or Oracle table space, then you can specify the name of the database instance or table space in a property.

The options and properties apply to the job as a whole, not to specific stages or links. For example, if a sequence of lookups, joins, and transformations feed a database target, and you select the option to **Push processing to database targets**, then the optimizer successively attempts to push each processing stage (lookup, join, transformation, and so on) in the sequence into the target database.

You can set the following options and properties:

Push processing to database sources
: Select this option to push processing that is implemented by Transformer, Sort, Aggregator, Remove Duplicates, Join, and Lookup stages back into database sources where possible.

Push processing to database targets
: Select this option to push processing that is implemented by Transformer, Sort, Aggregator, Join, and Lookup stages back into database targets where possible.

Push data reductions to database targets
: Data reduction processing involves reducing the volume of data before it is written to the target database. Aggregations and transforms that have constraint expressions set are both examples of data reduction processing. In normal circumstances, you do not want to push this processing to a target, because it means that a greater volume of data is written to the target. However, pushing data reduction processing to the target database can be useful; for example, when the aggregation or filtering is performed on data that is already in the target database.

Use bulk loading
: Select this option to perform target database modification by using high-performance bulk loading of a temporary staging table in the target database.

Data used to INSERT, UPDATE, or DELETE rows in the real target table is first bulk-loaded into a temporary staging table. After the staging table is bulk-loaded, SQL statements are run within the database engine (using the target connector after-SQL capability) to both manage the actual destination table as originally specified, and to post-process the data into its final destination table. By default, any staging table is created in the same target database space as the real target table, but you can specify a different database space for the staging table if required. By default, the staging table is always dropped if it exists, and then is recreated on each run of the job. You can manually change these settings in the optimized job before you run it.

Push all processing into the database
If you have a job where all the data sources and targets are located in the same database, and the database is capable of carrying out the processing tasks, then setting this option moves all the job logic to the database. The optimized job comprises two stages: a dummy Row Generator stage feeding a target database stage (the dummy stage is required because InfoSphere DataStage does not support single-stage jobs). When the optimized job is run, all the data remains within the database; there is no database I/O and no actual data flows through the job. Complex SQL is generated to perform all the processing within the target database.

You can specify values for the following properties:

Name of a stage where optimization should stop
Select the Name of a processing stage in the job where optimization should stop. Leave blank to allow optimization of any processing stage.

Staging table database instance name (Teradata) Staging tablespace (IBM DB2, or Oracle)
You can use this property to supply a name for an alternative Teradata database instance, DB2 table space, or Oracle table space, where bulk staging and parallel synchronization tables are created when bulk loading is used. By default, the current instance or table space is used. The specified database instance or table space must be accessible using the same authentication details as the default instance or table space.

Staging table name
You can use this property to supply a name for the staging table that is used when bulk loading is used.

Teradata database version
Select the Teradata database version from the list to control the SQL dialect generated by Balanced Optimization for this optimization run. Note this Teradata version might be different from the Teradata version that you normally use. The default value reflects the most general and most compatible SQL dialect.

Maximum number of SQL nested joins
Database SQL processors can have difficulty handling large numbers of nested joins. Select **no limit** to have Balanced Optimization use as many joins as needed. Select a positive integer value, *N*, to limit optimization in any single database connector to no more than *N* nested joins.

Effects of optimization on parallelism, partitioning, and sorting

Optimizing a job can affect its parallelism, its partitioning, and how data is sorted by the job.

When optimization pushes processing into a source or target database, it inserts complex native SQL into the connector that communicates with that database. The optimization process sets various partitioning and sorting properties in the connector, depending upon the type of optimization that is performed.

When processing is pushed into a target connector:

- The execution mode (sequential or parallel) set on the connector is left as it is, assuming that it reflects the intentions of the job designer (for example, to minimize target database resource usage by setting the connector to run sequentially).
- The partitioning or collection method on the input link of the connector is set to **auto**.

When processing is pushed into a source connector:

- Any sorting specified in the stage whose logic is pushed into the connector becomes an ORDER BY clause in the connector. The sort operation replaces any ORDER BY clause already in the connector. There is currently no distinction between a stable and an unstable sort.
- The execution mode specified in the source connector is left unchanged, except where the logic of a Sort stage is pushed into a source connector that uses **Bulk** mode. In this case, the execution mode is changed from parallel to sequential.
- If the **same** partitioning or collection method is used by the stage whose logic is pushed into the source connector, the method is reset to **(auto)**. The **(auto)** mode inserts appropriate partitioning based on key columns on the link. If any other partitioning or collection method is used, the method is retained as previously set.
- Where the logic of a Sort stage is pushed into the source connector, if the following stage in the job specifies the **(auto)** or **same** partitioning method, this method is replaced in the optimized job by the partition mode of the Sort stage. The partitioning method is replaced to maintain the desired partitioning in the job. For example, If a Sort stage specified **hash** partitioning on a particular key, and was followed by a Modify stage with **same** partitioning, then, in the optimized job, the Modify stage would specify the hash partitioning method on that key.

InfoSphere DataStage often processes or creates sorted data. Many of the sorting semantics cannot be directly reflected in SQL.

In an UPDATE statement, if two or more data rows can affect a single row in a database table in a non-commutative way (for example, setting a person's salary to two different values), the order in which the data rows are applied to the target table is important.

In an INSERT statement, if two or more data rows with the same primary key attempt to INSERT rows into the target table, only the first succeeds. However, SQL does not guarantee any particular order of processing of data. When InfoSphere DataStage job processing is pushed into database targets, the actual order in which data rows are applied cannot be guaranteed. If your data might contain multiple rows that affect the same row in a target table inside the database in an order-dependent way (for example, UPDATE with a fixed value or a non-commutative arithmetic expression), you can turn off the **Push Processing To Database Targets** option to guarantee correct operation.

Optimizing InfoSphere DataStage jobs

You optimize a job by opening the job in the IBM InfoSphere DataStage Designer client and selecting the **optimize** option.

To optimize an InfoSphere DataStage job, do the following steps:

1. Start the Designer client and attach to the project that contains the job.
2. Open the job that you want to optimize.
3. Set the options and properties that control optimization.
4. Optimize the job.
5. View the optimization log.
6. Save the optimized job as a new job.

Selecting the job to optimize

You can optimize parallel jobs that contain Teradata connectors, IBM DB2 connectors, or Oracle connectors.

Procedure

1. Open the job to be optimized in the Designer client.
2. Compile the job and fix any compilation issues.
3. Select **File** > **Optimize** to open the optimization window. If the job is not suitable for optimization, an error message is displayed.

Setting optimization options

You set options to control the optimization of each job in the Optimization window.

Before you begin

Open the job that you want to optimize in the Designer client, and select **File** > **Optimize** to open the Optimize window.

About this task

You set general options on the **Optimization options** tab of the Optimize window. You set advanced options on the **Advanced options** tab of the Optimize window.

Procedure

1. On the **Optimization options** tab, select or clear the following options:

Option	Setting
Push processing to database sources	Select this option to push the processing implemented by Transformers, Aggregators, Sorts, Funnels, Remove Duplicates, Joins, and Lookup to database sources. This option is selected by default.
Push processing to database targets	Select this option to push the processing implemented by Transformers, Aggregators, Sorts, Funnels, Joins, and Lookup to database targets. This option is selected by default.
Use bulk loading of target staging tables	Select this option to update a target database by using high-performance bulk loading of a temporary staging table in the target database. This option is selected by default.
Push all processing into the database	Select this option to move all the job logic to the database if you have a job where all the data sources and targets are located in the same database, and the database is capable of carrying out the processing tasks. This option is selected by default.
Push data reduction processing to database targets	Select this option to have aggregations and other data reduction processing pushed to database targets. This option is not selected by default.

2. Specify values for the following advanced options:

Property	Value
Name of stage where optimization should stop	Select the name of a processing stage in the job where optimization should stop. Leave blank to allow optimization of any processing stage.

Property	Value
Teradata database version	Select the Teradata version from the list to control the SQL dialect generated by Balanced Optimizer for this optimization run. Note that this Teradata version might be different from the Teradata version you usually use. The default value reflects the most general and most compatible SQL dialect.
Staging table database instance name (Teradata) Staging tablespace (IBM DB2) Staging tablespace (Oracle)	If you select the **Use bulk loading of target staging tables** option, you can specify the name of the database instance or table space that holds the staging table in this property. By default, the staging table is located in the same database instance or table space as the target table. The specified database instance or table space must be accessible with the same authentication as the default instance or table space.
Maximum number of SQL nested joins	Select **no limit** to have the Balanced Optimizer use as many joins as needed. Select an integer *N* to limit the number of nested joins to *N*. The default value is 2.

Optimizing a job and saving the new optimized job

You optimize a job and save the optimized version of the job from the Optimize window.

Before you begin

In the Designer client, open the job that you want to optimize set any required options and properties for that job.

About this task

After you optimize the job and save it, you can view the optimized job in the Designer client and examine the changes. If you want different results, you can alter some of the options and properties and try optimizing the job again.

Procedure

1. In the Designer client, open the job that you want to optimize.
2. Click **Optimize**.
3. When the optimization is finished, view the optimized job design in the Optimize window. Use the controls in the title bar to zoom in and out as required.
4. Optional: Click the **Compare** tab to view the root job alongside the optimized job.
5. Optional: Click the **Logs** tab to view the log that was generated when the job was optimized.
6. To save the new optimized job, click **Save As**.
7. Browse the repository tree in the Save Optimized Job As window and select the location for the optimized job.
8. In the **Item name** field, specify a name for the new job, or accept the default name.
9. Click **Save** to save the new optimized job to the repository.

Example

The following figure shows the view of the optimized job compared to the root job before optimization.

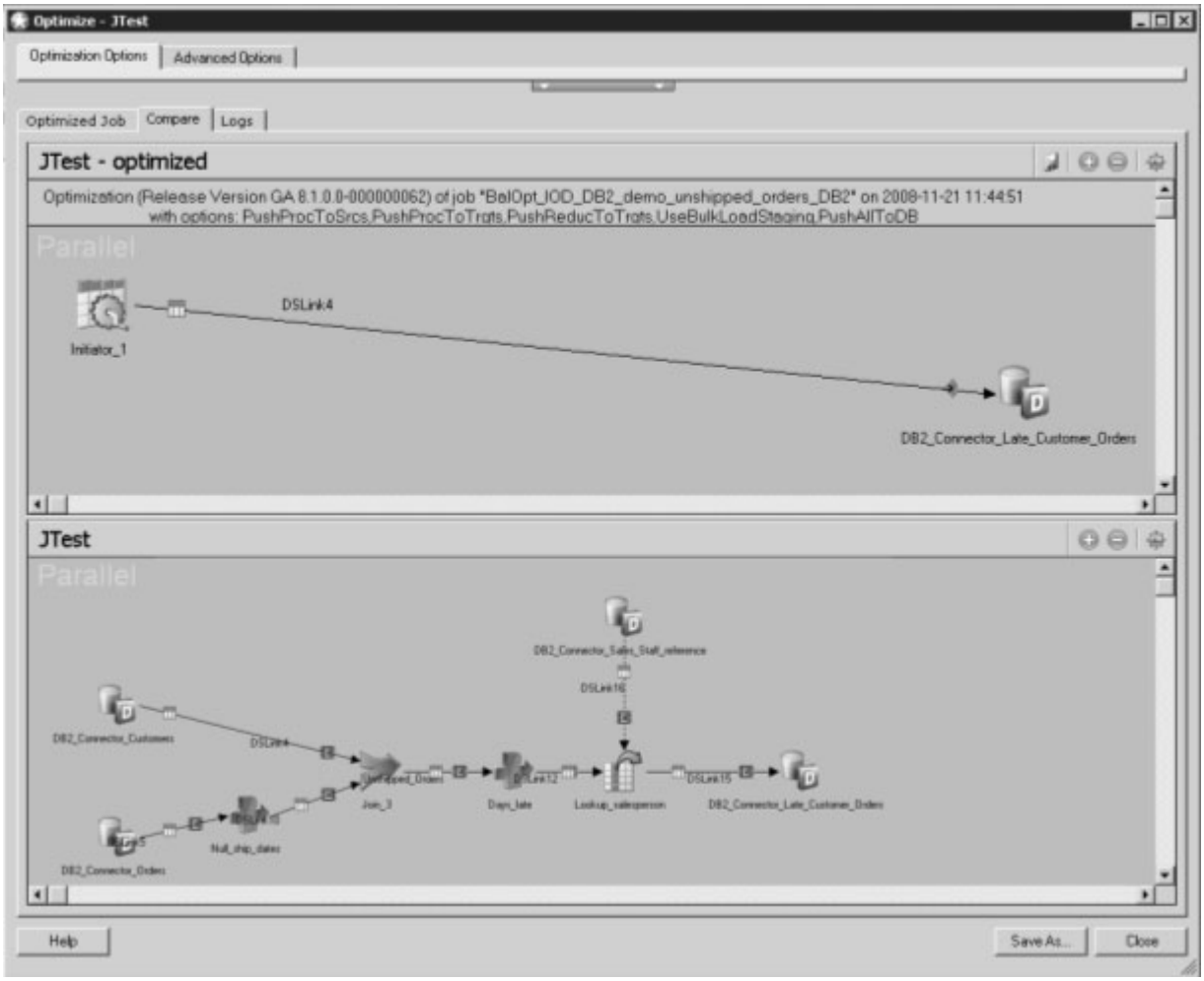


Figure 41. Balanced Optimization optimized job and root job

Viewing the optimization log

The optimization log shows details about which optimizations were attempted on your job and in what order.

About this task

The final optimization is the result of trying many patterns, and retrying patterns as the job is changed by previous patterns. The details of what optimizations were tried and in what order, what they did when they succeeded, and details of why they did not succeed, are logged.

You can view the optimization log in the Log window. You can copy text from the log and paste it into another application. The optimization log can be found in the directory C:\Documents and Settings*user*\Application Data\IBM\InformationServer\DataStage Client*clientTagID*\BalOp\logs\optimization.log, for example, C:\Documents and Settings\ds_user1\Application Data\IBM\InformationServer\DataStage Client\\BalOp\logs\optimization.log.

Procedure

1. Optimize your job.
2. In the Optimize window, click the **Log** tab.

3. View the log details. The log messages are color-coded.

Example

The following picture shows an optimization log.

Figure 42. Balanced Optimization log

Searching for optimized jobs

A relationship is maintained between a job and optimized versions of that job. You can search for all the optimized versions of a particular job.

About this task

You can compile and run the root job without the optimized versions being present. If you use a 'where used' query on the root job, however, the query returns a list of optimized versions of that job. If you change the root job, you must reoptimize the job to pick up the changes.

Procedure

1. Select the job in the repository tree in the Designer client.
2. Either right-click to open the menu or open the **Repository** menu from the main menu bar.
3. Select **Find where used** > **Jobs** to search for the optimized versions of the selected job and show the results in the Advanced Find window.

Searching for the root job

A relationship is maintained between an optimized job and the root job it was optimized from. You can discover the root job by searching from an optimized job.

About this task

There are two methods of discovering the root job of an optimized job. You can use the impact analysis facilities in the Designer client to perform a dependencies query, or you can view the dependencies of the optimized job in the Job Properties window.

Procedure

1. To discover the root job by using an dependencies query:
   a. Select the job in the repository tree in the Designer client.
   b. Either right-click to open the menu or open the **Repository** menu from the main menu bar.
   c. Select **Find dependencies** > **Jobs** to search for the root version of the selected job and show the results in the Advanced Find window.
2. To view the dependencies in a Job Properties window:
   a. Open the optimized job in the Designer client.
   b. Select **Edit** > **Job Properties** to open the Job Properties window.
   c. Click the **Dependencies** tab to open the Dependencies page.
   d. View the details of the root job in the Dependencies grid. The grid displays the name of the root job, and the date and time that it was optimized to produce the current job.

Breaking the relationship between optimized job and root job

You can break the relationship that is maintained between an optimized job and the root job that it was optimized from.

About this task

Although you can edit, compile, and run a root job and any optimized versions of that job independently of each other, a relationship is maintained. The relationship causes the root job to appear as a dependency of an optimized job. The relationship prevents an optimized job itself from being optimized. You can, if required, break the relationship so the root job and the optimized job are no longer linked, and behave like ordinary jobs.

You break the relationship by using the Job Properties window of the optimized job.

Procedure

1. Open the optimized job in the Designer client.
2. Select **Edit** > **Job Properties** to open the Job Properties window.
3. Click the **Dependencies** tab to open the Dependencies page.
4. In the Dependencies grid, select the row that displays the root job.
5. Right-click and select **Delete row** from the menu to break the relationship.

Exporting optimized jobs

You can export an optimized job from the IBM InfoSphere DataStage Designer client and maintain its relationship to the root job.

About this task

Export optimized jobs together with their root job because if you export the jobs in isolation, the relationship is exported too. If a job exists on the target computer with the same name as the root job, then a link is established with that job. This linking might be the required behavior, but it can also lead to unexpected relationships.

You can export optimized jobs from the Designer client. You can also export optimized jobs by using any of the other available export methods including the export service, the asset interchange interface, or the InfoSphere Information Server Manager.

Procedure

1. Select the optimized job in the repository tree.
2. Right-click and select **Export** from the menu.
3. In the Repository Export window, select the Include dependent items option.
4. Specify the name of the export file.
5. Click **OK** to export the optimized job with its root job.

Exporting root jobs

You can export a root job from the IBM InfoSphere DataStage Designer client, and maintain its relationship to any optimized versions of the job.

About this task

You must explicitly add the optimized versions of the job to the exported jobs.

Procedure

1. Select the root job in the repository tree.
2. Right-click and select **Export**.
3. Click **Add**, select the optimized jobs from the Select Items window, and click **OK** to add the jobs to the export list.
4. Specify the name of the export file.
5. Click **OK** to export the root job with the selected optimized jobs.

Job design considerations

There are some limitations to the optimizations that can be performed on InfoSphere DataStage jobs.

The SQL that is native to the source or target databases does not support some of the capabilities of InfoSphere DataStage and so that functionality cannot be pushed to a source or target database.

Supported functions

Database native SQL does not provide equivalent functionality for all the functions that you can use in job designs.

The following table lists functions that can be used in a Transformer stage and states whether they can be reproduced in SQL.

If you use any of these unsupported functions in any expression in a Transformer stage, then the processing contained in that Transformer stage cannot be pushed into target or source SQL. If you want to use these items in your job, segregate them into a separate Transformer stage which will not be pushed into a database source or target.

Table 141. Functions

Function	Supported in Teradata SQL	Supported in IBM DB2 SQL	Supported in Oracle SQL	Comment
Abs(Number)	Yes	Yes	Yes	
Acos(Number)	Yes	Yes	Yes	
Alnum(String)	No	No	No	no SQL equivalent
Alpha(String)	No	No	No	no SQL equivalent
AsDouble(Number)	Yes	Yes	Yes	
AsFloat(Number)	Yes	Yes	Yes	
ASin(Number)	Yes	Yes	Yes	
AsInteger(Number)	Yes	Yes	Yes	
Atan(Number)	Yes	Yes	Yes	
BitAnd(Num1,Num2)	No	No	Yes	
BitCompress(String)	No	No	No	no SQL equivalent
BitExpand(Number)	No	No	No	no SQL equivalent
BitOr(Num1,Num2)	No	No	No	no SQL equivalent
Ceil(Number)	Yes	Yes	Yes	
Char(Number)	Yes	No	Yes	
CompactWhiteSpace(String)	No	No	No	no SQL equivalent
Compare(String1,String2)	Yes	Yes	Yes	
Compare(String1,String2,"L")	Yes	Yes	Yes	
Compare(String1,String2,"R")	No	No	Yes	
CompareNoCase(String1,String2)	Yes	Yes	Yes	
CompareNum(String1,String2, Length)	Yes	Yes	Yes	
CompareNumNoCase(String1,String2, Length)	Yes	Yes	Yes	
Convert(FromList,ToList,String)	No	Yes	Yes	no Teradata SQL equivalent
Cos(Number)	Yes	Yes	Yes	
Cosh(Number)	Yes	Yes	Yes	
Count(String1,String2)	No	No	No	no SQL equivalent
CurrentDate()	Yes	Yes	Yes	
CurrentTime()	Yes	Yes	Yes	
CurrentTimeMS()	Yes	No	Yes	no DB2 SQL equivalent
CurrentTimestamp()	Yes	Yes	Yes	
CurrentTimestampMS()	Yes	Yes	Yes	
DateFromDaysSince(Days,Date)	Yes	Yes	Yes	
DateFromDaysSince(Days)	No	No	Yes	
DateFromJulianDay(JulianDayNumber)	Yes	Yes	Yes	
DateToString(Date)	Yes	Yes	Yes	

Table 141. Functions (continued)

Function	Supported in Teradata SQL	Supported in IBM DB2 SQL	Supported in Oracle SQL	Comment
DateToString(Date,Format)	Yes	Yes	Yes	DB2 formats limited to "%DD.%MM.%YYYY", "%MM-%DD-%YYYY", "%MM/%DD/%YYYY", "%YYYY-%MM-%DD"
DaysSinceFromDate(SourceDate, GivenDate)	Yes	Yes	Yes	
DCount(String,Delimiter)	No	No	No	no SQL equivalent
DecimalToDecimal(Number)	Yes	No	Yes	DB2 default rounding does not match DataStage default rounding
DecimalToDecimal(Number,Rounding)	Partial	Partial	Partial	for DB2, Rounding="trunc_zero" only. For Teradata, and Oracle, Rounding="round_inf" only
DecimalToDFloat(Number)	Yes	Yes	Yes	
DecimalToDFloat(Number,"fix_zero")	Yes	Yes	Yes	
DecimalToString(Number)	Yes	Yes	Yes	
DecimalToString(Number,"fix_zero")	Yes	Yes	Yes	
DFloatToDecimal(Number)	Yes	No	Yes	DB2 default rounding does not match DataStage default rounding
DFloatToDecimal(Number,Rounding)	Partial	Partial	Partial	for DB2, Rounding="trunc_zero" only. For Teradata, and Oracle, Rounding="round_inf" only
DFloatToStringNoExp(Number,scale)	No	No	No	no SQL equivalent
Div(Number,Divisor)	Yes	Yes	Yes	
DownCase(String)	Yes	Yes	Yes	
DQuote(String)	Yes	Yes	Yes	
DSJobStartDate()	Yes	Yes	Yes	
DSJobStartTime()	Yes	Yes	Yes	
DSJobStartTimestamp()	Yes	Yes	Yes	
ElementAt(Arg)	No	No	No	no SQL equivalent
Exp(Number)	Yes	Yes	Yes	
Fabs(Number)	Yes	Yes	Yes	
Field(String,Delimiter,Occurrence)	Partial	Partial	Yes	supported only for Occurrence=1
Field(String,Delimiter,Occurrence, Number)	Partial	Partial	Partial	supported only for Occurrence=1 and Number=1
Floor(Number)	Yes	Yes	Yes	
GetEnvironment(EnvVar)	No	No	No	no SQL equivalent

Table 141. Functions (continued)

Function	Supported in Teradata SQL	Supported in IBM DB2 SQL	Supported in Oracle SQL	Comment
HoursFromTime(Time)	Yes	Yes	Yes	
Index(String1,String2,Number)	Partial	Partial	Yes	For Teradata and DB2 supported only for Number=1
IsNotNull(Arg)	Yes	Yes	Yes	
IsNull(Arg)	Yes	Yes	Yes	
isValid(Type,Arg)	Partial	No	No	for Teradata, the following Type arguments are supported: "date", "dfloat", "int16", "int32", "int64", "int8", "raw", "sfloat", "string", "time", "uint16", "uint32", "uint64", "unit8", "timestamp"
JulianDayFromDate(Date)	Yes	Yes	Yes	
Ldexp(Mantissa,Exponent)	No	No	No	no SQL equivalent
Left(String,Length)	Yes	Yes	Yes	
Len(String)	Yes	Yes	Yes	
Llabs(Number)	No	Yes	Yes	no Teradata SQL equivalent
Ln(Number)	Yes	Yes	Yes	
Log10(Number)	Yes	Yes	Yes	
MakeNull(Arg,NullString)	No	No	No	no SQL equivalent
MantissaFromDecimal(Number)	No	No	No	no SQL equivalent
MantissaFromDFloat(Number)	No	No	No	no SQL equivalent
Max(Num)	Yes	Yes	Yes	
Max(Num1,Num2)	Yes	Yes	Yes	
Mean(Num)	Yes	Yes	Yes	
Mean(Num1,Num2)	Yes	Yes	Yes	
MicrosecondsFromTime(Time)	Yes	No	Yes	DB2 TIME type does not support microseconds
Min(Num)	Yes	Yes	Yes	
Min(Num1,Num2)	Yes	Yes	Yes	
MinutesFromTime(Time)	Yes	Yes	Yes	
Mod(Num1,Num2)	Yes	Yes	No	
MonthDayFromDate(Date)	Yes	Yes	Yes	
MonthFromDate(Date)	Yes	Yes	Yes	
Neg(Number)	Yes	Yes	Yes	
NextSKChain(Vector)	No	No	No	no SQL equivalent
NextSurrogateKey()	No	No	No	no SQL equivalent
NextWeekdayFromDate(Date,Weekday)	No	No	Yes	
NullToEmpty(Arg)	Yes	Yes	Yes	

Table 141. Functions (continued)

Function	Supported in Teradata SQL	Supported in IBM DB2 SQL	Supported in Oracle SQL	Comment
NullToValue(Arg1,Arg2)	Yes	Yes	No	
NullToZero(Arg)	Yes	Yes	No	
Num(Arg)	No	No	No	no SQL equivalent
PadString(String,PadString)	No	No	No	no SQL equivalent
PadString(String,PadString,Length)	No	Yes	Yes	no Teradata SQL equivalent
PreviousWeekdayFromDate(Date, Weekday)	No	No	Yes	
PrevSKChain(Vector)	No	No	No	no SQL equivalent
Pwr(Num1,Num2)	Yes	Yes	Yes	
Rand()	No	No	Yes	produces unsigned 32-bit values; SQL INTEGER is signed 32-bit
Random()	No	No	Yes	produces unsigned 32-bit values; SQL INTEGER is signed 32-bit
RawLength(Raw)	Yes	Yes	Yes	
Right(String,Length)	Yes	Yes	Yes	
SecondsFromTime(Time)	Yes	Yes	Yes	
SecondsSinceFromTimestamp (Timestamp1,Timestamp2)	No	Yes	No	No Teradata or Oracle SQL equivalent
SetBit(Number)	No	No	No	no SQL equivalent
SetNull()	Yes	Yes	Yes	
Seq(Character)	No	Yes	Yes	
Sin(Number)	Yes	Yes	Yes	
Sinh(Number)	Yes	Yes	Yes	
Soundex(String)	Yes	Yes	Yes	
Space(Number)	Partial	Yes	Yes	For Teradata, limited to N <= 254
Sqrt(Number)	Yes	Yes	Yes	
SQuote(String)	Yes	Yes	Yes	
Str(String,RepeatCount)	Partial	Yes	Yes	For Teradata, RepeatCount <= 5
StringToDate(String)	Yes	Yes	Yes	
StringToDate(String,Format)	Yes	Partial	Yes	For DB2, Format limited to "%MM/%DD/%YYYY"
StringToDecimal(String)	Yes	No	Yes	
StringToDecimal(String,Format)	Partial	Partial	Partial	For DB2, Format can only be “trunc_zero”; for Teradata, and Oracle, Format can only be “round_inf”
StringToRaw(String)	Yes	No	Yes	

Table 141. Functions (continued)

Function	Supported in Teradata SQL	Supported in IBM DB2 SQL	Supported in Oracle SQL	Comment
StringToTime(String)	Yes	Yes	Yes	
StringToTime(String,Format)	Yes	Partial	No	For DB2, Format limited to "%HH:%NN:%SS"
StringToTimestamp(String)	Yes	Yes	Yes	
StringToTimestamp(String,Format)	Yes	Partial	Yes	For DB2, Format limited to "%YYYY-%MM-%DD %HH-%NN-%SS"
StringToUstring(String)	No	No	No	no SQL equivalent
StringToUstring(String,map_name)	No	No	No	no SQL equivalent
StripWhiteSpace(String)	No	No	Yes	
Substring(String,SubString,Length)	Yes	Yes	Yes	
Tan(Number)	Yes	Yes	Yes	
Tanh(Number)	Yes	Yes	Yes	
TimeDate()	Yes	Yes	Yes	
TimeFromMidnightSeconds(Seconds)	Yes	Yes	Yes	
TimestampFromDateTime(Date,Time)	Yes	Yes	Yes	
TimestampFromSecondsSince(Seconds)	Yes	Yes	Yes	
TimestampFromSecondsSince(Seconds, Timestamp)	Yes	Yes	Yes	
TimestampFromTimet(Time_t)	No	No	No	no SQL equivalent
TimestampToDate(Timestamp)	Yes	Yes	Yes	
TimestampToString(Timestamp)	Yes	Yes	Yes	
TimestampToString(Timestamp,Format)	Yes	Partial	Yes	For DB2, Format limited to "%YYYY-%MM-%DD %HH:%MI:%SS". For Format does not include microseconds
TimestampToTime(Timestamp)	Yes	Yes	Yes	
TimetFromTimestamp(Timestamp)	No	No	No	no SQL equivalent
TimeToString(Time)	Yes	Yes	Yes	
TimeToString(Time,Format)	Yes	Partial	Yes	For DB2, Format limited to "%HH:%NN:%SS"
Trim(String)	No	No	No	No SQL equivalent for compressing internal white space
Trim(String,Option)	Partial	Partial	Partial	Option = 'E' or 'F' only.
Trim(String,StripChar,Option)	Partial	Partial	Partial	Option = 'B', 'L' or 'T' only
TrimB(String)	Yes	Yes	Yes	
TrimF(String)	Yes	Yes	Yes	
TrimLeadingTrailing(String)	Yes	Yes	Yes	
Upcase(String)	Yes	Yes	Yes	
UstringToString(UnicodeString)	No	No	No	no SQL equivalent

Table 141. Functions (continued)

Function	Supported in Teradata SQL	Supported in IBM DB2 SQL	Supported in Oracle SQL	Comment
UstringToString(UnicodeString, MapName)	No	No	No	no SQL equivalent
WeekdayFromDate(Date)	No	Yes	Yes	No Teradata SQL equivalent
YeardayFromDate(Date)	Yes	Yes	Yes	
YearFromDate(Date)	Yes	Yes	Yes	
YearweekFromDate(Date)	No	Yes	Yes	No Teradata SQL equivalent

Supported operators

Database native SQL does not provide equivalent functionality for all the operators that you can use in job designs.

The following table lists operators that can be used in a Transformer stage and states whether they can be reproduced in SQL.

If you use any of these unsupported operators in any expression in a Transformer stage, then the processing contained in that Transformer stage cannot be pushed into target or source SQL. If you want to use these items in your job, segregate them into a separate Transformer stage which will not be pushed into a database source or target.

Table 142. Operators

Operator	Supported in Teradata SQL	Supported in IBM DB2 SQL	Supported in Oracle	Comment
-A	Yes	Yes	Yes	
A - B	Yes	Yes	Yes	
A ! B	Yes	Yes	Yes	
A # B	Yes	Yes	Yes	
A #< B	Yes	Yes	Yes	
A #> B	Yes	Yes	Yes	
A & B	Yes	Yes	Yes	
A * B	Yes	Yes	Yes	
A / B	Yes	Yes	Yes	
A : B	Yes	Yes	Yes	
A ^ B	Yes	Yes	Yes	
A + B	Yes	Yes	Yes	
A < B	Yes	Yes	Yes	
A <= B	Yes	Yes	Yes	
A <> B	Yes	Yes	Yes	
A = B	Yes	Yes	Yes	
A =< B	Yes	Yes	Yes	
A => B	Yes	Yes	Yes	

Table 142. Operators (continued)

Operator	Supported in Teradata SQL	Supported in IBM DB2 SQL	Supported in Oracle	Comment
A > B	Yes	Yes	Yes	
A >< B	Yes	Yes	Yes	
A >= B	Yes	Yes	Yes	
A and B	Yes	Yes	Yes	
A eq B	Yes	Yes	Yes	
A ge B	Yes	Yes	Yes	
A gt B	Yes	Yes	Yes	
A le B	Yes	Yes	Yes	
A lt B	Yes	Yes	Yes	
A ne B	Yes	Yes	Yes	
A or B	Yes	Yes	Yes	
A**B	Yes	Yes	Yes	
IF A THEN B ELSE C	Yes	Yes	Yes	
not A	Yes	Yes	Yes	
not(A)	Yes	Yes	Yes	
S[A]	Yes	Yes	Yes	
S[A,B]	Yes	Yes	Yes	
S[A,B,C]	No	No	No	no SQL equivalent for N^{th} substring

Supported macros and system variables

Database native SQL does not provide equivalent functionality for all the macros and system variables that you can use in job designs.

The following table lists macros and system variables that can be used in a Transformer stage and states whether they can be reproduced in SQL.

If you use any of these unsupported macros and system variables in any expression in a Transformer stage, then the processing contained in that Transformer stage cannot be pushed into target or source SQL. If you want to use these items in your job, segregate them into a separate Transformer stage which will not be pushed into a database source or target.

Table 143. Macros and system variables

Element	Supported in Teradata SQL	Supported in IBM DB2 SQL	Supported in Oracle	Comment
Macros: DSHostName DSJobController DSJobInvocationId DSJobName DSJobWaveNo DSProjectName	No	No	No	These macros concern various aspects of the running a job. There are no equivalents in SQL.

Table 143. Macros and system variables (continued)

Element	Supported in Teradata SQL	Supported in IBM DB2 SQL	Supported in Oracle	Comment
Macros: DSJobStartDate DSJobStartTime DSJobStartTimestamp	Yes	Yes	Yes	These transformer macros are evaluated once at the start of job execution. Translated into Teradata SQL CURRENT_DATE, CURRENT_TIME and CURRENT_TIMESTAMP functions, which are evaluated once at the start of the SQL statement. For Oracle, the CURRENT_TIME is translated into CAST('00'||' '||TO_CHAR(CURRENT_TIMESTAMP, 'HH24:MI:SS') AS INTERVAL DAY(2) TO SECOND(0)).
System variables: @TRUE @FALSE @INROWNUM @NUMPARTITIONS @OUTROWNUM @PARTITIONNUM	No	No	No	There are no equivalents in SQL.

Database stage types

The Balanced Optimization tool can only optimize jobs that contain certain types of database connector stages.

Teradata connector

Only the Teradata connector is supported for optimization. Other Teradata access stages are not supported for optimization.

The Teradata connector requires the Teradata Parallel Transporter interface to be installed on each InfoSphere DataStage node that will run jobs containing the Teradata connector.

Join, lookup, and funnel stage processing in a job can be pushed into a Teradata connector source or target only if all the Connection property values of the involved connector stages are compatible. All such stages must have the same values for **Server**, **Username**, and **Password** properties. These values can be literal values or job parameters. The database properties of all such stages must be compatible, and conform to one of the following schemes:

- Database properties of all involved connectors are set to blank (the empty string). Because **Server** and **Username** properties are the same, all these connectors refer to the same default Teradata database for the specific Teradata server and user name.
- Database properties of all involved stages are set to explicit non-blank values, including literal values or job parameters.

InfoSphere DataStage Balanced Optimization does not support quoted identifiers in the Teradata connector.

DB2 connector

Only the DB2 connector is supported for optimization. Other DB2 access stages are not supported for optimization.

InfoSphere Balanced Optimization does not support quoted identifiers in the DB2 connector.

Oracle connector

Only the Oracle connector is supported for optimization. Other Oracle access stages are not supported for optimization.

InfoSphere Balanced Optimization does not support quoted identifiers in the Oracle connector.

Processing stage types

Database native SQL does not provide some of the functionality provided by certain stage types that you can use in job designs.

Transformer stage

InfoSphere DataStage Balanced Optimization optimizes the functionality contained in a Transformer stage. It does not attempt to optimize functionality contained in a BASIC Transformer stage.

SQL does not support static variables. Therefore any Transformer stage that contains a stage variable whose derivation either directly or indirectly refers to itself cannot be optimized. Such logic is often used to implement functionality such as accumulators and control break logic. To be optimized, this logic must be expressed in other ways, for example, by using aggregation.

SQL does not support temporary variables. There is no way for an expression that produces one select list element to refer to any other select list element. Therefore, any Transformer stage that contains an output column derivation that refers to another output column cannot be optimized. To enable optimization, compute common subexpressions as stage variables, and then use them in multiple-output column derivations.

The use of static variables or temporary variables is detected when you attempt to optimize a job that contains them. The InfoSphere Balanced Optimization tool provides detailed log information about why the job could not be optimized. It might be possible to reorganize the computations to allow optimization of that job.

By default, InfoSphere DataStage supports intermediate results with a precision/scale of decimal(38,10) (up to 38 total decimal digits, of which up to 10 can be fractional digits) during complex arithmetic calculations. SQL databases individually support different maximums. For example, Teradata releases earlier than V6.2 support a maximum precision of 18 decimal digits. So complex calculations with decimal data types, when pushed into database SQL, can exhibit rounding errors due to lower precision/scale. To ensure comparable results, you can set the InfoSphere DataStage environment variables APT_DECIMAL_INTERM_PRECISION, APT_DECIMAL_INTERM_SCALE, and APT_DECIMAL_INTERM_ROUNDMODE to control the precision, scale, and rounding of intermediate results.

NULL and 0 values are often equivalent in InfoSphere DataStage. In a root job, the logical expression Link.Column = 0 might evaluate to true if Link.Column has a NULL value. SQL has no such equivalence. If the intent is to test for NULL, then use the function IsNull(*Link.Column*). The equivalence of 0 and NULL can cause a problem where the Transformer stage consumes the output of any stage that can produce NULL values (for example, a Join stage performing an outer join).

Aggregator stage

The InfoSphere DataStage Balanced Optimization tool currently supports the calculation aggregation type. It does not support the count or recalculation aggregation types.

The InfoSphere Balanced Optimization tool does not support the following aggregation functions:

- Corrected sum of squares
- Missing value
- Preserve Type
- Summary

For certain statistical functions, such as standard deviation and variance, there might be small differences in calculated answers between the root job and an optimized job where aggregation is pushed into the database. This is because of differences in parallel calculation algorithms, and the precision of intermediate results in InfoSphere DataStage and the database. These differences are typically beyond the fourth significant digit and are insignificant for practical purposes. Values for most other calculations (count, sum, mean) are unaffected.

Join stage

The InfoSphere DataStage Balanced Optimization tool supports joins with two inputs.

By default, the Join stage sorts data on both input links. This sorting results in output data that is sorted on the join keys. If the InfoSphere DataStage project environment variable APT_NO_SORT_INSERTION is set, the sort is suppressed. Pushing a Join stage into a database source always generates an ORDER BY clause for the sort keys, as if APT_NO_SORT_INSERTION was not set.

InfoSphere DataStage performs implicit type conversions to make join keys of different types comparable. SQL, however, is much stricter and such implicit conversions can lead to runtime database errors. Therefore Join key columns within different type groups (for examples, numbers and strings) cannot be optimized.

Lookup stage

The InfoSphere DataStage Balanced Optimization tool supports single lookups with two input links.

The following lookup conditions are supported:

- Lookup key equality (case sensitive for character-type columns)
- Multi-key lookups

The following lookup conditions are not supported:

- Range lookups
- Case-insensitive key equality lookups on character-type columns.

Only continue and drop actions are supported for lookup failure and condition-not-met. Both lookup failure and condition-not-met must be set to the same actions. In any other case, a warning is given in the optimization log and optimization of the lookup stage is not performed.

InfoSphere DataStage performs implicit type conversions to make join keys of different types comparable. SQL, however, is much stricter and such implicit conversions can lead to runtime database errors. Therefore Lookup key columns within different type groups (for examples, numbers and strings) cannot be optimized.

By default, for a normal (non-sparse) lookup, InfoSphere DataStage uses only one row matching on the lookup key or keys from a lookup table, and ignores any other lookup table rows that match. This behavior is not supported by SQL joins. However, since the typical InfoSphere DataStage usage is for lookup tables to have unique rows, Balanced Optimization issues a warning to remind you to ensure that the lookup table has unique rows. To eliminate this warning, set the **Multiple rows returned from link** property on the Lookup stage to the name of the reference link, or set the **Lookup type** property of the

database connector supplying the reference table to **sparse**. If the lookup table does not have unique rows on the lookup keys, and the intended behavior is to use only one matching row, do not optimize this lookup.

Containers

The InfoSphere DataStage Balanced Optimization tool does not support containers. Optimization stops at any container boundaries.

For a job with a local container, use the local container **Deconstruct** option to make the container contents part of the job, and so eligible for optimization. Select the local container stage in the job, right-click, and select **Deconstruct**.

For a job with a shared container, use the **Convert to local container** option, and then deconstruct the local container. Select the shared container stage in the job, right-click, and select **Convert to local container**.

Stages with multiple output links or reject links

Processing stages that can have multiple output links cannot typically be pushed into database sources or targets because SQL does not support SELECT statements with multiple outputs. Multiple output paths to the same database lead to separate target SQL (INSERT, UPDATE, DELETE) statements, which operate independently, rather than being forced into a single sequence. This consideration applies also to reject output links, including reject links from Connectors.

User-defined SQL

If your job contains user-defined SQL, you might encounter limitations in optimization.

The InfoSphere DataStage Balanced Optimization tool attempts to parse user-defined SQL, but sometimes the SQL cannot be parsed, even if it is valid for the database.

For SQL in a target connector, failure to parse the SQL prevents pushing of processing into that database target.

For SQL in a source or reference connector, failure to parse the SQL causes a warning in the optimization trace log. The unparsed SQL statement is embedded in generated SQL for pushing processing to sources. Optimization attempts to drop any ORDER BY clauses in the unparsed SQL statement.

When the whole job is pushed into target connector SQL, any unparsed source or reference SQL is embedded in generated target SQL. Check the detailed execution log of such an optimized job to ensure that the combined SQL that includes such embedded source or reference SQL functions correctly.

If you use multiple database instances when defining SQL, fully qualify table names with the intended database instance name (for example, dbinstancename.tablename not tablename). When InfoSphere Balanced Optimization cannot parse user-defined SQL, fully qualified names ensure correct name resolution when SQL statements are combined.

Combining connectors in Join, Lookup, and Funnel stages

Balanced Optimization can combine multiple connectors if your job design follows some basic rules.

Balanced Optimization can combine connectors where they are attached to the input links of Join, Lookup, and Funnel stages. It can also combine connectors where one is an input to a Join, Lookup, or Funnel stage, and the other is the output.

Balanced Optimization can combine two connectors that are linked to Join or Lookup stages, or multiple connectors linked to Funnel stages in the root job, if your design obeys the following rules:

- The **Server** property (**Instance** property for DB2 connector) must specify the same literal value, or use the same job parameter to specify the value in all connectors.
- The **Database** property must either be blank in all connectors, or specify a database name or job parameter. The connectors can specify different database names or job parameters from each other. (The **Database** property is not required in the Oracle connector.)
- The **Username** property must specify the same literal value, or use the same job parameter to specify the value in both connectors.
- The **Password** property must specify the same literal value, or use the same job parameter to specify the value in both connectors.

Runtime column propagation

The InfoSphere DataStage Balanced Optimization tool does not support runtime column propagation.

InfoSphere DataStage Balanced Optimization turns off runtime column propagation at the input link to any target database connector when processing is pushed into that connector, or when the connector is changed to use bulk staging. A DETAIL message is posted in the optimization log when runtime column propagation is turned off. There is no way to anticipate what columns might be propagated at run time, so such columns cannot be included in generated SQL statements. This inability means that, if the root job had runtime column propagation on from source to target in the job, then additional columns propagated from data sources will not be created or populated in the target table.

Runtime column propagation remains unchanged for the parts of the job not pushed into a target connector. When processing is pushed into a connector used as a source or reference, it is possible that additional columns that might have been returned from the original source query will not be propagated from the source database. This lack is because the processing pushed into the source SQL might mask those additional columns that were not named explicitly in the job.

Sorting data and database sources

Where processing stages sort from a source database connector, the InfoSphere DataStage Balanced Optimization tool imposes certain rules.

The following rules are imposed:

- If the source connector SQL contains an ORDER BY clause and the following processing stage does a stable sort, a WARNING is posted in the optimization log, and the sort is treated as unstable.
- Any ORDER BY clause in the source connector SQL is dropped. If the processing stage does any sorting, stable or unstable, it is converted into a new ORDER BY clause in the new source SQL.

A stable sort preserves the order of rows from any previous sort within the new sort, and so acts like secondary sort keys. An unstable sort does not guarantee to preserve any previous ordering.

The rules are applied every time another attempt is made to push the logic in a processing stage into a source connector. For example, if there is a sequence of unstable sorts (even with intervening processing stages), only sorting of the last such stage is pushed into the source database connector SQL. To take best advantage of available optimizations, design jobs that require data sorted on multiple columns to sort late in the job, and perform all possible processing before sorting the data. You must also avoid sorting data before a join in your job design. Join implicitly inserts unstable sorts on join keys.

InfoSphere DataStageBalanced Optimization does not support the pushing of sorting into database sources in the following circumstances:

- Where there are different National Language Support code pages or collating sequences in the root job and in the source database.

- Where EBCDIC sorting has been specified in the job.
- Where case-insensitive sorting has been specified in the job.
- Where nulls sort last (rather than first).

Sorting data and database targets

Jobs often process or create sorted data. Many of these semantics of these sort operations cannot be directly reflected in SQL pushed to a database target.

In an UPDATE statement, if two or more data rows can affect a single row in a database table in a non-commutative way (for example, setting a salary field to two different values), the order in which the data rows are applied to the target table is important. In an INSERT statement, if two or more data rows with the same primary key attempt to INSERT rows into the target table, only the first succeeds.

The InfoSphere Balanced Optimization tool attempts to preserve correct semantics with respect to sorted data. In many cases InfoSphere DataStage Balanced Optimization can determine with certainty from the root job design whether the order of application of data rows to database tables is important, and perform or avoid certain optimizations.

If you know that there is a sort order dependency in your processing, or you think that there might be, turn off the **Push processing to database targets** option. This guarantees correct target database writing semantics.

Data types

Source or target databases might not support all the data types that InfoSphere DataStage supports.

Teradata V2R5.1, V2R6.0, and V2R6.1 do not support 64-bit integers (BIGINTs), so any InfoSphere DataStage functions that produce 64-bit integer results cannot be pushed into Teradata sources or targets.

The maximum precision of the DECIMAL type is 18 digits for Teradata V2R5.1, V2R6.0 and V2R6.1, and 38 digits for Teradata V12.

The DB2 9.1 TIME data type does not support fractional digits or microseconds.

The following table lists transformer data types and the corresponding data type for each database.

Table 144. Transformer data types

Transformer data type	Teradata data type	IBM DB2 data type	Oracle data type
BIGINT	BIGINT	BIGINT	NUMBER(19)
BINARY	BYTE	CHARACTER	RAW
BIT	SMALLINT	SMALLINT	NUMBER(5)
CHAR	CHAR	CHARACTER	CHAR
DATE	DATE	DATE	DATE
DECIMAL	DECIMAL	DECIMAL	NUMBER
DOUBLE	FLOAT	DOUBLE	BINARY_DOUBLE
FLOAT	FLOAT	DOUBLE	BINARY_FLOAT
INTEGER	INTEGER	INTEGER	NUMBER(10)
LONGVARBINARY	BLOB	BLOB	BLOB
LONGVARCHAR	CLOB	CLOB	CLOB
NUMERIC	DECIMAL	DECIMAL	NUMBER
REAL	FLOAT	REAL	BINARY_FLOAT

Table 144. Transformer data types (continued)

Transformer data type	Teradata data type	IBM DB2 data type	Oracle data type
SMALLINT	SMALLINT	SMALLINT	NUMBER(5)
TIME	TIME	TIME	INTERVAL DAY(2) TO SECOND(0)
TIMESTAMP	TIMESTAMP	TIMESTAMP	TIMESTAMP
TINYINT	SMALLINT	SMALLINT	NUMBER(5)
VARBINARY	VARBYTE	VARCHAR	RAW
VARCHAR	VARCHAR	VARCHAR	VARCHAR2
WCHAR	CHAR	GRAPHIC	NCHAR
WLONGVARCHAR	CLOB	LONG VARGRAPHIC	NCLOB
WVARCHAR	VARCHAR	VARGRAPHIC	NVARCHAR2

Conversion errors

You must resolve any conversion errors in your root job before you optimize it.

Type conversion errors

When you compile and run your job, check for implicit conversion warnings in the job log. Resolve the reported issues in the job design by using a Transformer stage with explicit type conversion functions. You must use explicit, rather than implicit, conversions to ensure that the conversions are correctly mapped to their SQL equivalents.

Nullable to non-nullable column conversion errors

Use the Transformer stage functions NullToZero, NullToEmpty, and NullToValue in your job designs to ensure correct non-null values. This process is especially important for database sources that feed a Join stage with an outer join, which will output NULL values for columns of unmatched rows unless the columns are explicitly not nullable.

Character set conversion errors

Ensure that Client Character Set property of a connector is compatible with the CHARACTER SET specifications in all source and target table columns. For example, if a source database table column is defined with CHARACTER SET UNICODE, then the Client Character Set on the source connector must specify a multi-byte character set such as UTF8 to ensure enough space in internal string buffers.

Complex write modes and circular table references

There are some special considerations where your jobs use complex write modes to update a table.

InfoSphere DataStage connectors offer three complex write modes: INSERT-then-UPDATE, UPDATE-then-INSERT, and DELETE-then-INSERT. When the INSERT-then-UPDATE or DELETE-then-INSERT modes are pushed into a target database, Balanced Optimization generates two consecutive SQL statements to implement the complex operation. When the UPDATE-then-INSERT operation is pushed into a target database, Balanced Optimization generates a SQL MERGE statement, except in the case of Teradata databases earlier than Teradata V12, where two SQL statements are generated.

Where two SQL statements are generated, a circular table reference problem can occur. A circular table reference is where the source and target refer to the same database table directly, or indirectly though database views. When the Balanced Optimization tool detects a direct circular reference, that part of the

optimization is abandoned with a warning, Balanced Optimization cannot detect an indirect circular reference, however. In such a case, you must use the optimization options to prevent this processing from being pushed into the target connector. Use one or more of the following options or properties:

- Set the **Name of a stage where optimization should stop** property to the name of the Join or Lookup stage where the circular reference is introduced.
- Turn off the **Push processing to database targets**, **Use bulk staging to database targets** and **Push all processing to the (target) database** options.

The circular table reference problem does not occur where an UPDATE-then-INSERT operation is pushed into an SQL MERGE statement.

Staging table management

Where you specify the bulk loading option, InfoSphere Balanced Optimization uses a staging table in the target database.

By default the staging table is created in the same database instance or table space as the target table, but you can use the **Staging table database instance name** (Teradata) or **Staging tablespace** (IBM DB2 and Oracle) properties to specify the name of a different database instance or table space in which to create the staging table.

The InfoSphere DataStage Balanced Optimization tool generates a unique staging table name. Target connector properties in the optimized job are set to create the staging table, bulk load it in parallel, post-process its contents into the real target table, and then drop the staging table. Constraints, such as not-null, are not used in creating the staging table, to minimize performance loss during bulk loading, and to accommodate mis-marked column definitions in InfoSphere DataStage job links.

After optimization, you can modify these connector properties if required (for example, you can retain the staging table to examine its contents). If you want to use a statically predefined staging table, you can use the generated CREATE TABLE statement in the TableAction section of the connector to create the staging table. Be aware, however, that subsequent reoptimizations of the root job with different options, or after changes in the root job design, might change the required schema, and you will have to respecify the CREATE TABLE statement.

The following table gives the parallel synchronization properties that are set in a target Teradata connector when a job has been optimized to bulk load a table:

Table 145. Teradata connector synchronization property settings for bulk loading

Property	Setting
Sync table	Generated name for a private synchronization table for the optimized job
Sync server	Blank (defaults to Server property value)
Sync user	Blank (defaults to User name property value)
Sync password	Blank (defaults to Password property value)
Sync database	Set the same as the staging table database instance name optimization property. User must have CREATE TABLE permission in this database
Sync ID	Set to a unique value for the specific optimized job. If you plan on running the same optimized job concurrently on different InfoSphere Information Server servers, you need to set this value differently on each such job instance.
Sync table action	Create (synchronization table will be created if it does not exist)

Table 145. Teradata connector synchronization property settings for bulk loading (continued)

Property	Setting
Sync table cleanup	Keep (synchronization table will be kept after the job runs)
Sync write mode	Insert (new rows are added with each execution of this job)
Sync poll	0 (default)
Sync timeout	0 (default)

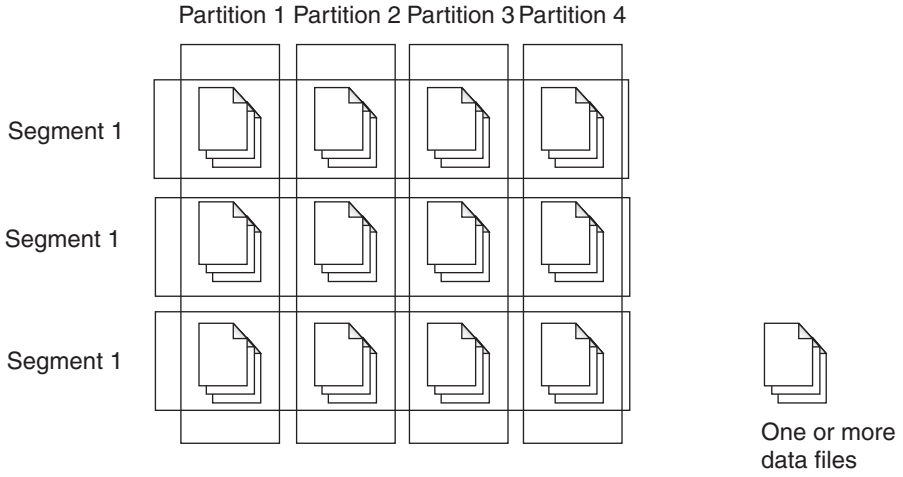
Chapter 12. Managing data sets

InfoSphere DataStage parallel jobs use data sets to store data being operated on in a persistent form. Data sets are operating system files, each referred to by a descriptor file, usually with the suffix .ds.

You can create and read data sets using the Data Set stage, which is described in "File set stage" on page 111. InfoSphere DataStage also provides a utility for managing data sets from outside a job. This utility is available from the InfoSphere DataStage Designer and Director clients.

Structure of data sets

A data set comprises a descriptor file and a number of other files that are added as the data set grows. These files are stored on multiple disks in your system. A data set is organized in terms of partitions and segments. Each partition of a data set is stored on a single processing node. Each data segment contains all the records written by a single InfoSphere DataStage job. So a segment can contain files from many partitions, and a partition has files from many segments.



The descriptor file for a data set contains the following information:

- Data set header information.
- Creation time and date of the data set.
- The schema of the data set.
- A copy of the configuration file use when the data set was created.

For each segment, the descriptor file contains:

- The time and date the segment was added to the data set.
- A flag marking the segment as valid or invalid.
- Statistical information such as number of records in the segment and number of bytes.
- Path names of all data files, on all processing nodes.

This information can be accessed through the Data Set Manager.

Starting the data set manager

About this task

To start the Data Set Manager from the InfoSphere DataStage Designer, or Director:

Procedure

1. Choose **Tools** > **Data Set Management**, a Browse Files dialog box appears.
2. Navigate to the directory containing the data set you want to manage. By convention, data set files have the suffix .ds.
3. Select the data set you want to manage and click **OK**. The Data Set Viewer appears. From here you can copy or delete the chosen data set. You can also view its schema (column definitions) or the data it contains.

Data set viewer

The Data Set viewer displays information about the data set you are viewing:

Partitions

The partition grid shows the partitions the data set contains and describes their properties:

- **#**. The partition number.
- **Node**. The processing node that the partition is currently assigned to.
- **Records**. The number of records the partition contains.
- **Blocks**. The number of blocks the partition contains.
- **Bytes**. The number of bytes the partition contains.

Segments

Click on an individual partition to display the associated segment details. This contains the following information:

- **#**. The segment number.
- **Created**. Date and time of creation.
- **Bytes**. The number of bytes in the segment.
- **Pathname**. The name and path of the file containing the segment in the selected partition.

Click the **Refresh** button to reread and refresh all the displayed information.

Click the **Output** button to view a text version of the information displayed in the Data Set Viewer.

You can open a different data set from the viewer by clicking the **Open** icon on the tool bar. The browse dialog open box opens again and lets you browse for a data set.

Viewing the schema

About this task

Click the Schema icon from the tool bar to view the record schema of the current data set. This is presented in text form in the Record Schema window.

Viewing the data
About this task

Click the Data icon from the tool bar to view the data held by the current data set. This options the Data Viewer Options dialog box, which allows you to select a subset of the data to view.

- **Rows to display**. Specify the number of rows of data you want the data browser to display.
- **Skip count**. Skip the specified number of rows before viewing data.
- **Period**. Display every *P*th record where *P* is the period. You can start after records have been skipped by using the Skip property. *P* must equal or be greater than 1.
- **Partitions**. Choose between viewing the data in **All** partitions or the data in the partition selected from the drop-down list.

Click **OK** to view the selected data, the Data Viewer window appears.

Copying data sets
About this task

Click the Copy icon on the tool bar to copy the selected data set. The Copy data set dialog box appears, allowing you to specify a path where the new data set will be stored.

The new data set will have the same record schema, number of partitions and contents as the original data set.

Note: You cannot use the UNIX *cp* command to copy a data set because InfoSphere DataStage represents a single data set with multiple files.

Deleting data sets
About this task

Click the Delete icon on the tool bar to delete the current data set data set. You will be asked to confirm the deletion.

Note: You cannot use the UNIX *rm* command to delete a data set because InfoSphere DataStage represents a single data set with multiple files. Using *rm* simply removes the descriptor file, leaving the much larger data files behind.

Chapter 13. The Parallel engine configuration file

One of the great strengths of InfoSphere DataStage is that, when designing parallel jobs, you don't have to worry too much about the underlying structure of your system, beyond appreciating its parallel processing capabilities. If your system changes, is upgraded or improved, or if you develop a job on one platform and implement it on another, you don't necessarily have to change your job design.

InfoSphere DataStage learns about the shape and size of the system from the *configuration file*. It organizes the resources needed for a job according to what is defined in the configuration file. When your system changes, you change the file not the jobs.

This chapter describes how to define configuration files that specify what processing, storage, and sorting facilities on your system should be used to run a parallel job. You can maintain multiple configuration files and read them into the system according to your varying processing needs.

When you install InfoSphere DataStage, the system is automatically configured to use the supplied default configuration file. This allows you to run parallel jobs right away, but is not optimized for your system. Follow the instructions in this chapter to produce configuration file specifically for your system.

Configurations editor

About this task

The InfoSphere DataStage Designer provides a configuration file editor to help you define configuration files for the parallel engine. To use the editor, choose **Tools** > **Configurations**, the Configurations dialog box appears.

To define a new file, choose **(New)** from the **Configurations** drop-down list and type into the upper text box. Guidance on the operation and format of a configuration file is given in the following sections. Click **Save** to save the file at any point. You are asked to specify a configuration name, the config file is then saved under that name with an .apt extension.

You can verify your file at any time by clicking **Check**. Verification information is output in the **Check Configuration Output** pane at the bottom of the dialog box.

To edit an existing configuration file, choose it from the **Configurations** drop-down list. You can delete an existing configuration by selecting it and clicking Delete. You are warned if you are attempting to delete the last remaining configuration file.

You specify which configuration will be used by setting the APT_CONFIG_FILE environment variable. This is set on installation to point to the default configuration file, but you can set it on a project wide level from the InfoSphere DataStage Administrator (see *InfoSphere DataStage Administrator Client Guide*) or for individual jobs from the Job Properties dialog.

Configuration considerations

The parallel engine's view of your system is determined by the contents of your current configuration file. Your file defines the processing nodes and disk space connected to each node that you allocate for use by parallel jobs. When invoking a parallel job, the parallel engine first reads your configuration file to determine what system resources are allocated to it and then distributes the job to those resources.

When you modify the system by adding or removing nodes or disks, you must modify your configuration file correspondingly. Since the parallel engine reads the configuration file every time it runs a parallel job, it automatically scales the application to fit the system without your having to alter the job code.

Your ability to modify the parallel engine configuration means that you can control the parallelization of a parallel job during its development cycle. For example, you can first run the job on one node, then on two, then on four, and so on. You can measure system performance and scalability without altering application code.

Logical processing nodes

A parallel engine configuration file defines one or more processing nodes on which your parallel job will run. The processing nodes are logical rather than physical. The number of processing nodes does not necessarily correspond to the number of CPUs in your system. Your configuration file can define one processing node for each physical node in your system, or multiple processing nodes for each physical node.

Optimizing parallelism

The degree of parallelism of a parallel job is determined by the number of nodes you define when you configure the parallel engine. Parallelism should be optimized for your hardware rather than simply maximized. Increasing parallelism distributes your work load but it also adds to your overhead because the number of processes increases. Increased parallelism can actually hurt performance once you exceed the capacity of your hardware. Therefore you must weigh the gains of added parallelism against the potential losses in processing efficiency.

Obviously, the hardware that makes up your system influences the degree of parallelism you can establish.

SMP systems allow you to scale up the number of CPUs and to run your parallel application against more memory. In general, an SMP system can support multiple logical nodes. Some SMP systems allow scalability of disk I/O. "Configuration Options for an SMP" discusses these considerations.

In a cluster or MPP environment, you can use the multiple CPUs and their associated memory and disk resources in concert to tackle a single computing problem. In general, you have one logical node per CPU on an MPP system. "Configuration Options for an MPP System" describes these issues.

The properties of your system's hardware also determines configuration. For example, applications with large memory requirements, such as sort operations, are best assigned to machines with a lot of memory. Applications that will access an RDBMS must run on its server nodes; and stages using other proprietary software, such as SAS, must run on nodes with licenses for that software.

Here are some additional factors that affect the optimal degree of parallelism:

- CPU-intensive applications, which typically perform multiple CPU-demanding operations on each record, benefit from the greatest possible parallelism, up to the capacity supported by your system.
- Parallel jobs with large memory requirements can benefit from parallelism if they act on data that has been partitioned and if the required memory is also divided among partitions.
- Applications that are disk- or I/O-intensive, such as those that extract data from and load data into RDBMSs, benefit from configurations in which the number of logical nodes equals the number of disk spindles being accessed. For example, if a table is fragmented 16 ways inside a database or if a data set is spread across 16 disk drives, set up a node pool consisting of 16 processing nodes.
- For some jobs, especially those that are disk-intensive, you must sometimes configure your system to prevent the RDBMS from having either to redistribute load data or to re-partition the data from an extract operation.

- The speed of communication among stages should be optimized by your configuration. For example, jobs whose stages exchange large amounts of data should be assigned to nodes where stages communicate by either shared memory (in an SMP environment) or a high-speed link (in an MPP environment). The relative placement of jobs whose stages share small amounts of data is less important.
- For SMPs, you might want to leave some processors for the operating system, especially if your application has many stages in a job. See "Configuration Options for an SMP" .
- In an MPP environment, parallelization can be expected to improve the performance of CPU-limited, memory-limited, or disk I/O-limited applications. See "Configuration Options for an MPP System" .

The most nearly-equal partitioning of data contributes to the best overall performance of a job run in parallel. For example, when hash partitioning, try to ensure that the resulting partitions are evenly populated.This is referred to as *minimizing skew*. Experience is the best teacher. Start with smaller data sets and try different parallelizations while scaling up the data set sizes to collect performance statistics.

Configuration options for an SMP

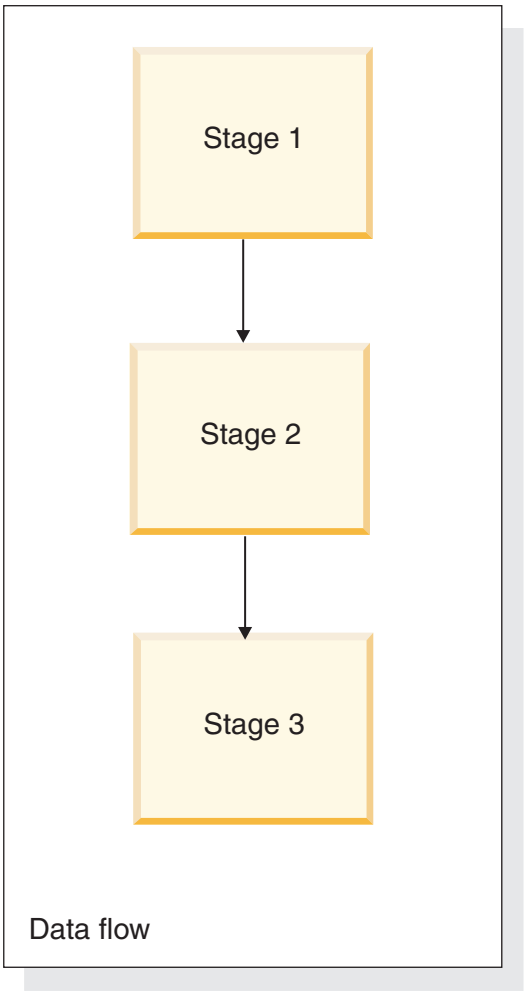
An SMP contains multiple CPUs which share operating system, disk, and I/O resources. Data is transported by means of shared memory. A number of factors contribute to the I/O scalability of your SMP. These include the number of disk spindles, the presence or absence of RAID, the number of I/O controllers, and the speed of the bus connecting the I/O system to memory.

SMP systems allow you to scale up the number of CPUs. Increasing the number of processors you use might or might not improve job performance, however, depending on whether your application is CPU-, memory-, or I/O-limited. If, for example, a job is CPU-limited, that is, the memory, memory bus, and disk I/O of your hardware spend a disproportionate amount of time waiting for the CPU to finish its work, it will benefit from being executed in parallel. Running your job on more processing units will shorten the waiting time of other resources and thereby speed up the overall application.

All SMP systems allow you to increase your parallel job's memory access bandwidth. However, none allow you to increase the memory bus capacity beyond that of the hardware configuration. Therefore, memory-intensive jobs will also benefit from increased parallelism, provided they do not saturate the memory bus capacity of your system. If your application is already approaching, or at the memory bus limit, increased parallelism will not provide performance improvement.

Some SMP systems allow scalability of disk I/O. In those systems, increasing parallelism can increase the overall throughput rate of jobs that are disk I/O-limited.

For example, the following figure shows a data flow containing three parallel stages:



For each stage in this data flow, the parallel engine creates a single UNIX process on each logical processing node (provided that stage combining is not in effect). On an SMP defined as a single logical node, each stage runs sequentially as a single process, and the parallel engine executes three processes in total for this job. If the SMP has three or more CPUs, the three processes in the job can be executed simultaneously by different CPUs. If the SMP has fewer than three CPUs, the processes must be scheduled by the operating system for execution, and some or all of the processors must execute multiple processes, preventing true simultaneous execution.

In order for an SMP to run parallel jobs, you configure the parallel engine to recognize the SMP as a single or as multiple logical processing node(s), that is:

```
1 <= M <= N logical processing nodes
```

where *N* is the number of CPUs on the SMP and *M* is the number of processing nodes on the configuration. (Although *M* can be greater than *N* when there are more disk spindles than there are CPUs.)

As a rule of thumb, it is recommended that you create one processing node for every two CPUs in an SMP. You can modify this configuration to determine the optimal configuration for your system and application during application testing and evaluation. In fact, in most cases the scheduling performed by the operating system allows for significantly more than one process per processor to be managed before performance degradation is seen. The exact number depends on the nature of the processes, bus bandwidth, caching effects, and other factors.

Depending on the type of processing performed in your jobs (sorting, statistical analysis, database I/O), different configurations might be preferable.

For example, on an SMP viewed as a single logical processing node, the parallel engine creates a single UNIX process on the processing node for each stage in a data flow. The operating system on the SMP schedules the processes to assign each process to an individual CPU.

If the number of processes is less than the number of CPUs, some CPUs might be idle. For jobs containing many stages, the number of processes might exceed the number of CPUs. If so, the processes will be scheduled by the operating system.

Suppose you want to configure the parallel engine to recognize an eight-CPU SMP, for example, as two or more processing nodes. When you configure the SMP as two separate processing nodes, the parallel engine creates two processes per stage on the SMP. For the three-stage job shown above, configuring an SMP as more than two parallel engine processing nodes creates at least nine UNIX processes, although only eight CPUs are available. Process execution must be scheduled by the operating system.

For that reason, configuring the SMP as three or more parallel engine processing nodes can conceivably degrade performance as compared with that of a one- or two-processing node configuration. This is so because each CPU in the SMP shares memory, I/O, and network resources with the others. However, this is not necessarily true if some stages read from and write to disk or the network; in that case, other processes can use the CPU while the I/O-bound processes are blocked waiting for operations to finish.

Example Configuration File for an SMP

This section contains a sample configuration file for the four-CPU SMP shown below:

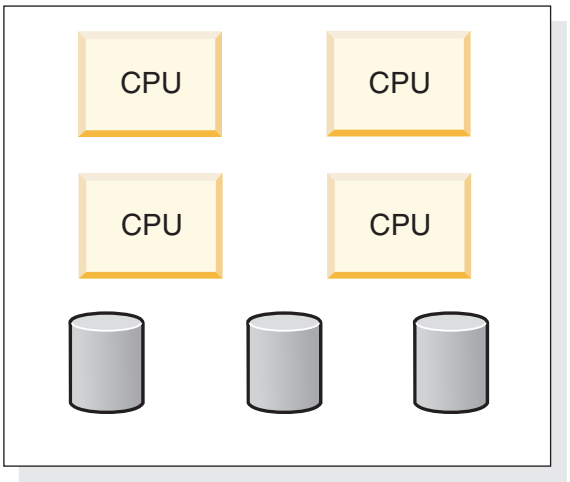


Figure 43. Example SMP system

The table below lists the processing node names and the file systems used by each processing node for both permanent and temporary storage in this example system:

Table 146. Nodes and file systems

Node name	Node name on fast network	Node pools	Directory for permanent storage	Directory for temp storage
`node0`	`node0_byn`	`"", node0, node0_fddi`	`/orch/s0 /orch/s1`	`/scratch`

Table 146. Nodes and file systems (continued)

Node name	Node name on fast network	Node pools	Directory for permanent storage	Directory for temp storage
node1	node0_byn	"", node1, node1_fddi	/orch/s0 /orch/s1	/scratch

The table above also contains a column for node pool definitions. Node pools allow you to execute a parallel job or selected stages on only the nodes in the pool. See "Node Pools and the Default Node Pool" for more details.

In this example, the parallel engine processing nodes share two file systems for permanent storage. The nodes also share a local file system (/scratch) for temporary storage.

Here is the configuration file corresponding to this system. "Configuration Files" discusses the keywords and syntax of configuration files.

```
{
   node "node0" {
      fastname "node0_byn"                  /* node name on a fast network */
      pools "" "node0" "node0_fddi"         /* node pools */
      resource disk "/orch/s0" {}
      resource disk "/orch/s1" {}
      resource scratchdisk "/scratch" {}
   }
   node "node1" {
      fastname "node0_byn"
      pools "" "node1" "node1_fddi"
      resource disk "/orch/s0" {}
      resource disk "/orch/s1" {}
      resource scratchdisk "/scratch" {}
   }
}
```

Configuration options for an MPP system

An MPP consists of multiple hosts, where each host runs its own image of the operating system and contains its own processors, disk, I/O resources, and memory. This is also called a *shared-nothing* environment. Each host in the system is connected to all others by a high-speed network. A host is also referred to as a physical node.

In an MPP environment, you can use the multiple CPUs and their associated memory and disk resources in concert. In this environment, each CPU has its own dedicated memory, memory bus, disk, and disk access.

When configuring an MPP, you specify the physical nodes in your system on which the parallel engine will run your parallel jobs. You do not have to specify all nodes.

An Example of a four-node MPP system configuration

The following figure shows a sample MPP system containing four physical nodes:

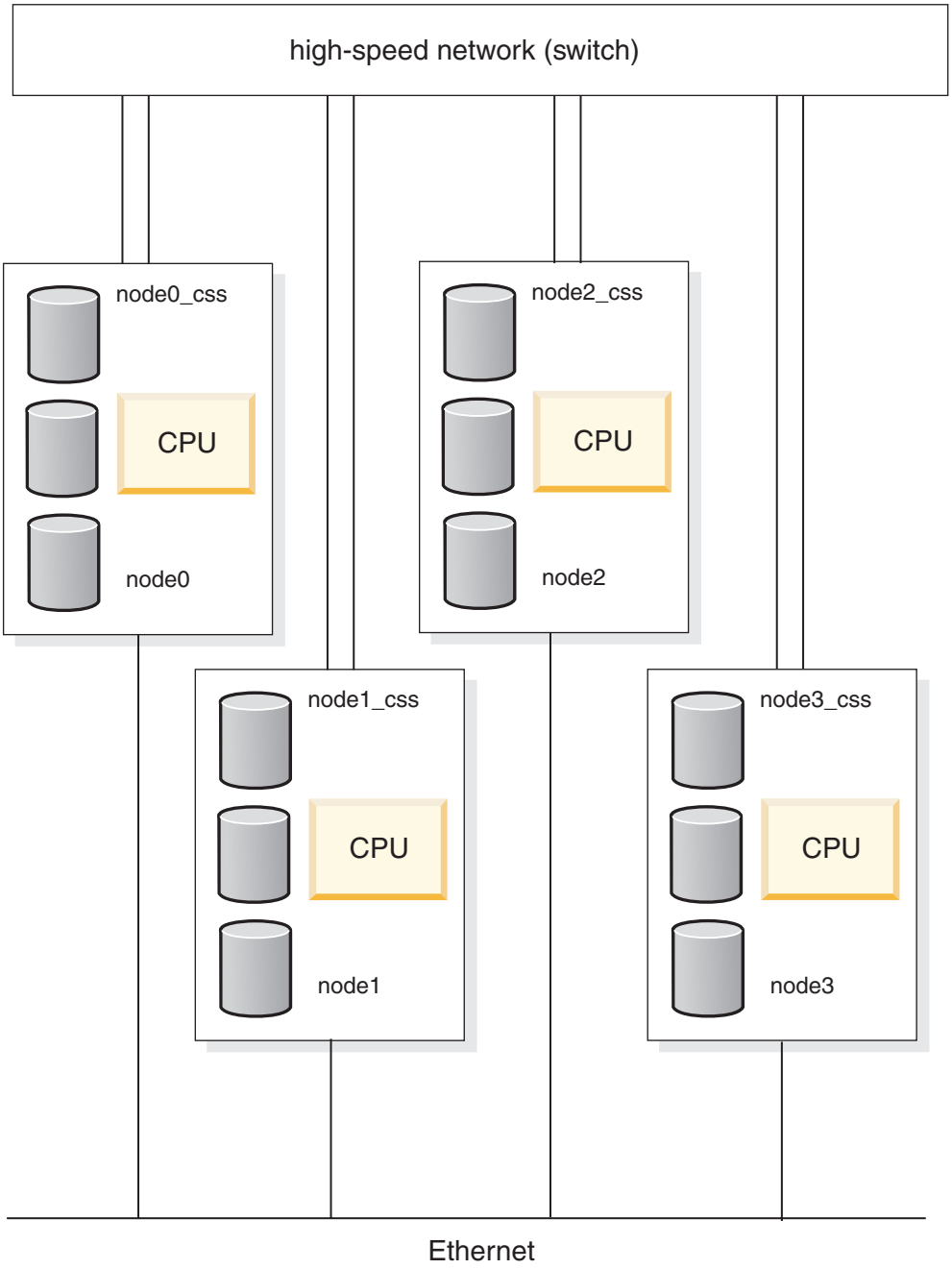


Figure 44. MPP system

This figure shows a *disk-everywhere* configuration. Each node is connected to both a high-speed switch and an Ethernet. Note that the configuration information below for this MPP would be similar for a cluster of four SMPs connected by a network.

The following table shows the storage local to each node:

Table 147. Storage local to nodes

Node name	Node name on fast network	Node pools	Directory for permanent storage	Directory for temp storage
`node0`	`node0_css`	`"", node0, node0_css`	`/orch/s0 /orch/s1`	`/scratch`

Table 147. Storage local to nodes (continued)

Node name	Node name on fast network	Node pools	Directory for permanent storage	Directory for temp storage
node1	node1_css	"", node1, node1_css	/orch/s0 /orch/s1	/scratch
node2	node2_css	"", node2, node2_css	/orch/s0 /orch/s1	/scratch
node3	node3_css	"", node3, node3_css	/orch/s0 /orch/s1	/scratch

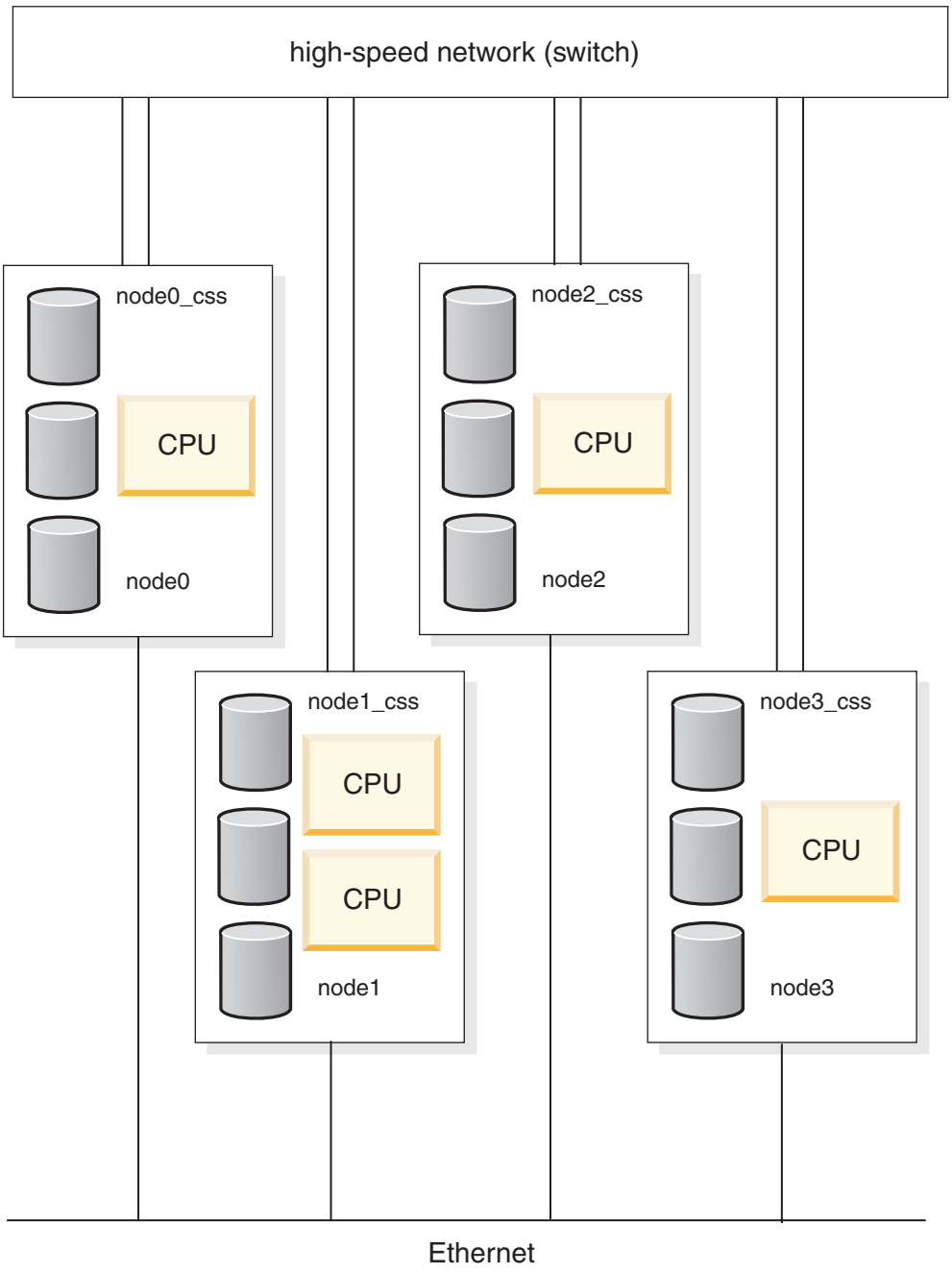
Note that because this is an MPP system, each node in this configuration has its own disks and hence its own /orch/s0, /orch/s1, and /scratch. If this were an SMP, the logical nodes would be sharing the same directories.

Here is the configuration file for this sample system. "Configuration Files" discusses the keywords and syntax of configuration files.

```
{
   node "node0" {
      fastname "node0_css"                  /* node name on a fast network*/
      pools "" "node0" "node0_css"          /* node pools */
      resource disk "/orch/s0" {}
      resource disk "/orch/s1" {}
          resource scratchdisk "/scratch" {}
       }
   node "node1" {
      fastname "node1_css"
      pools "" "node1" "node1_css"
      resource disk "/orch/s0" {}
      resource disk "/orch/s1" {}
      resource scratchdisk "/scratch" {}
       }
   node "node2" {
      fastname "node2_css"
      pools "" "node2" "node2_css"
      resource disk "/orch/s0" {}
      resource disk "/orch/s1" {}
      resource scratchdisk "/scratch" {}
       }
    node "node3" {
      fastname "node3_css"
      pools "" "node3" "node3_css"
      resource disk "/orch/s0" {}
              resource disk "/orch/s1" {}
      resource scratchdisk "/scratch" {}
       }
}
```

Configuration options for an SMP cluster

An SMP cluster consists of one or more SMPs and, optionally, single-CPU nodes connected by a high-speed network. In this case, each SMP in the cluster is referred to as a physical node. When you configure your system, you can divide a physical node into logical nodes. The following figure shows a cluster containing four physical nodes, one of which (node1) is an SMP containing two CPUs.



An example of an SMP cluster configuration

The following configuration file divides physical node1 into logical nodes node1 and node1a. Both are connected to the high-speed switch by the same fastname; in the configuration file, the same fastname is specified for both nodes. "Configuration Files" discusses the keywords and syntax of configuration files.

```
{
   node "node0" {
       fastname "node0_css"                 /* node name on a fast network */
       pools "" "node0" "node0_css"         /* node pools */
       resource disk "/orch/s0" {}
       resource disk "/orch/s1" {}
       resource scratchdisk "/scratch" {}
   }
```

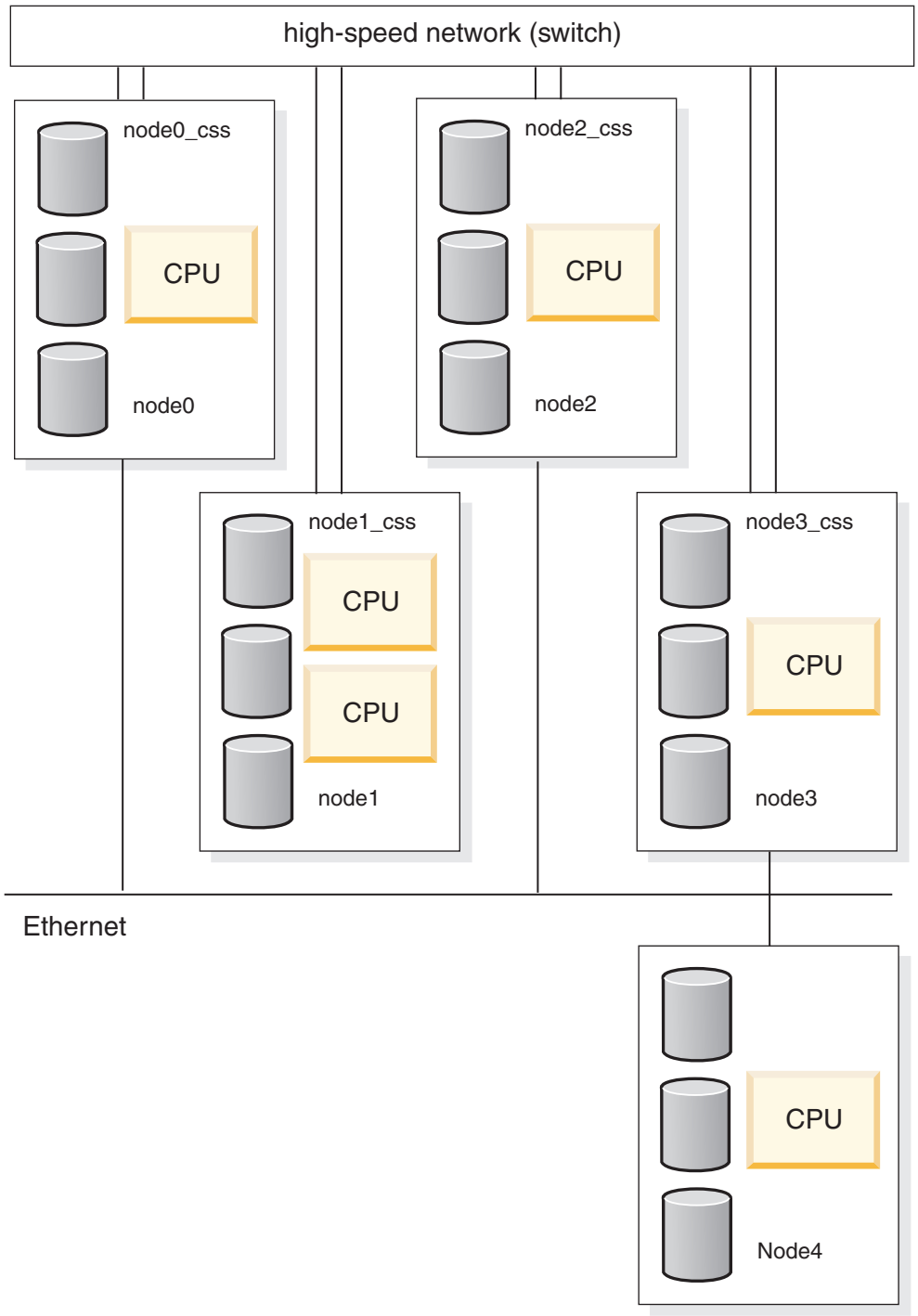
```
    node "node1" {
        fastname "node1_css"
        pools "" "node1" "node1_css"
        resource disk "/orch/s0" {}
        resource disk "/orch/s1" {}
        resource scratchdisk "/scratch" {}
    }
    node "node1a"            {
        fastname "node1_css"
        pools "" "node1" "node1_css"
        resource disk "/orch/s0" {}
        resource disk "/orch/s1" {}
        resource scratchdisk "/scratch" {}
    }
    node "node2"    {
        fastname "node2_css"
        pools "" "node2" "node2_css"
        resource disk "/orch/s0" {}
        resource disk "/orch/s1" {}
        resource scratchdisk "/scratch" {}
    }
    node "node3" {
        fastname "node3_css"
        pools "" "node3" "node3_css"
        resource disk "/orch/s0" {}
        resource disk "/orch/s1" {}
        resource scratchdisk "/scratch" {}
    }
}
```

In this example, consider the disk definitions for /orch/s0. Since node1 and node1a are logical nodes of the same physical node, they share access to that disk. Each of the remaining nodes, node0, node2, and node3, has its own /orch/s0 that is not shared. That is, there are four distinct disks called /orch/s0. Similarly, /orch/s1 and /scratch are shared between node1 and node1a but not the others.

Options for a cluster with the conductor unconnected to the high-speed switch

The parallel engine starts your parallel job from the Conductor node.

A cluster might have a node that is not connected to the others by a high-speed switch, as in the following figure.



:

In this example, node4 is the Conductor, which is the node from which you need to start your application. By default, the parallel engine communicates between nodes using the fastname, which in this example refers to the high-speed switch. But because the Conductor is not on that switch, it cannot use the `fastname` to reach the other nodes.

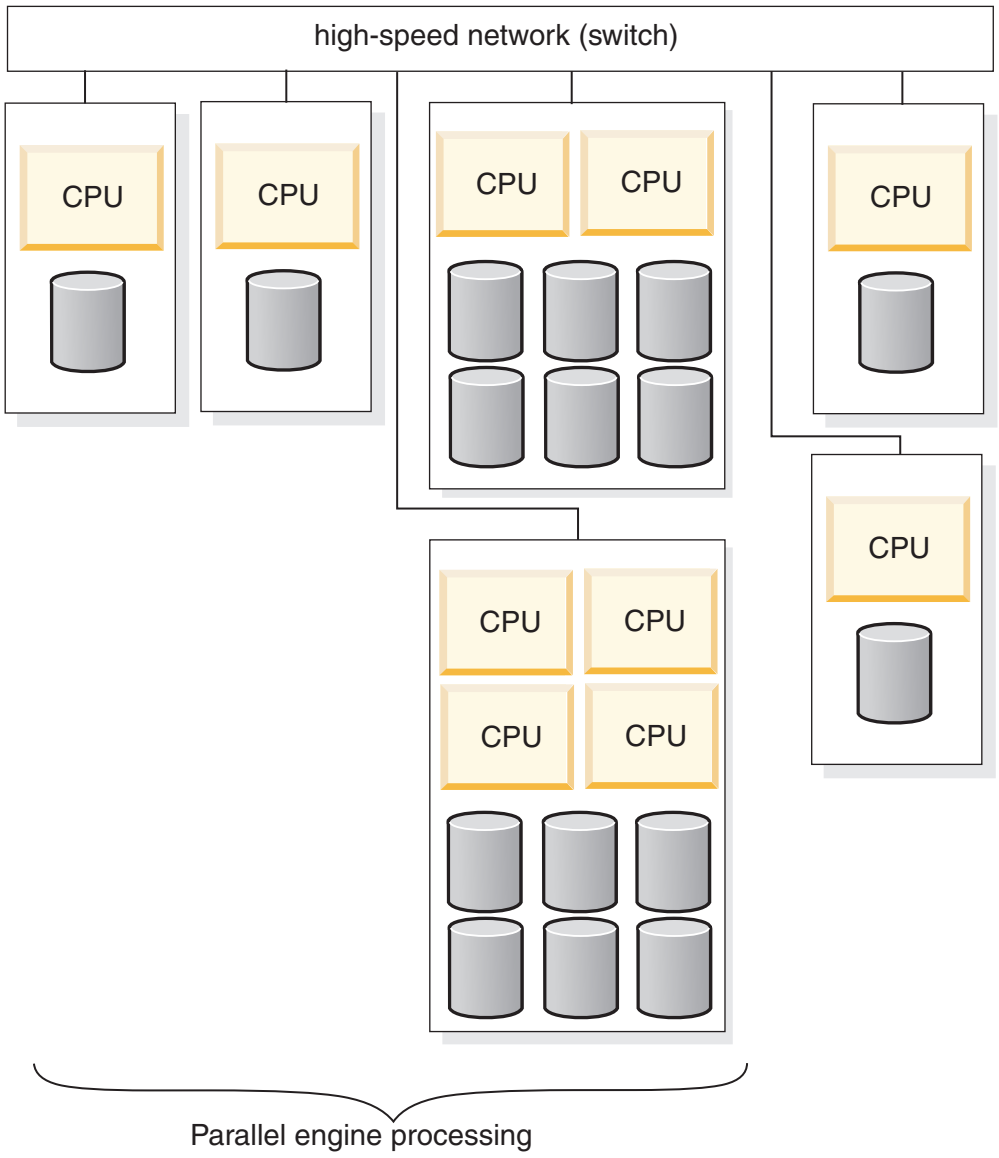
Therefore, to enable the Conductor node to communicate with the other nodes, you need to identify each node on the high-speed switch by its canonicalhostname and give its Ethernet name as its quoted attribute, as in the following configuration file. "Configuration Files" discusses the keywords and syntax of configuration files.

```
{ node "node0" {
      fastname "node0_css"
      resource canonicalhostname "node1-eth-1"
      pools "" "node0" "node0_css"
      resource disk "/orch/s0" {}
      resource disk "/orch/s1" {}
      resource scratchdisk "/scratch" {}
}
node "node1"    {
   fastname "node1_css"
   resource canonicalhostname "node1-eth-1"
   pools "" "node1" "node1_css"
   resource disk "/orch/s0" {}
   resource disk "/orch/s1" {}
   resource scratchdisk "/scratch" {}
   }
node "node2" {
   fastname "node2_css"
   resource canonicalhostname "node1-eth-1"
   pools "" "node2" "node2_css"
   resource disk "/orch/s0" {}
   resource disk "/orch/s1" {}
   resource scratchdisk "/scratch" {}
   }
node "node3" {
   fastname "node3_css"
   resource canonicalhostname "node1-eth-1"
   pools "" "node3" "node3_css"
   resource disk "/orch/s0" {}
   resource disk "/orch/s1" {}
   resource scratchdisk "/scratch" {}
   }
node "node4" {
   pools "" "conductor" "node4" "node4_css"
                                /* not in the default pool */
   resource disk "/orch/s0" {}
   resource disk "/orch/s1" {}
   resource scratchdisk "/scratch" {}
}
}
```

Note: Since `node4` is not on the high-speed switch and you are therefore using it only as the Conductor node, you have left it out of the default node pool (""). This causes the parallel engine to avoid placing stages on `node4`. See "Node Pools and the Default Node Pool" .

Diagram of a cluster environment

The following figure shows a mixed MPP and SMP cluster environment containing six physical nodes. Only the four nodes of the left are intended to be allocated for use by the parallel engine.



Configuration files

This section describes parallel engine configuration files, and their uses and syntax. The parallel engine reads a configuration file to ascertain what processing and storage resources belong to your system. Processing resources include nodes; and storage resources include both disks for the permanent storage of data and disks for the temporary storage of data (scratch disks). The parallel engine uses this information to determine, among other things, how to arrange resources for parallel execution.

You must define each processing node on which the parallel engine runs jobs and qualify its characteristics; you must do the same for each disk that will store data. You can specify additional information about nodes and disks on which facilities such as sorting or SAS operations will be run, and about the nodes on which to run stages that access the following relational data base management systems: DB2, INFORMIX, and Oracle.

You can maintain multiple configuration files and read them into the system according to your needs.

A sample configuration file is located in Configurations directory under the Server directory of your installation, and is called default.apt..

This section contains the following subsections:

- "The Default Path Name and the APT_CONFIG_FILE"
- "Syntax"
- "Node Names"
- "Options"
- "Node Pools and the Default Node Pool"
- "Disk and Scratch Disk Pools and Their Defaults"
- "Buffer Scratch Disk Pools"

The default path name and the APT_CONFIG_FILE

The default name of the configuration file is default.apt and it is located in the Configurations directory in the Server directory of your installation

You can give the configuration file a different name or location or both from their defaults. If you do, assign the new path and file name to the environment variable APT_CONFIG_FILE. If APT_CONFIG_FILE is defined, the parallel engine uses that configuration file rather than searching in the default locations. In a production environment, you can define multiple configurations and set APT_CONFIG_FILE to different path names depending on which configuration you want to use.

You can set APT_CONFIG_FILE on a project wide level from the InfoSphere DataStage Administrator (see *InfoSphere DataStage Administrator Client Guide*) or for individual jobs from the Job Properties dialog.

Note: Although the parallel engine might have been copied to all processing nodes, you need to copy the configuration file only to the nodes from which you start parallel engine applications (conductor nodes).

Syntax

Configuration files are text files containing string data. The general form of a configuration file is as follows:

```
/* commentary */
{
  node "node name" {
      <node information>
     .
     .
     .
   }
  .
  .
  .
 }
```

These are the syntactic characteristics of configuration files:

- Braces { } begin and end the file.
- The word `node` begins every node definition.
- The word `node` is followed by the name of the node enclosed in quotation marks. For a detailed discussion of node names, see "Node Names" .
- Braces { } follow the node name. They enclose the information about the node (its options), including an enumeration of each disk and scratch disk resource. The legal options are: `fastname`, `pools`, and `resource`.
- Spaces separate items.
- Quotation (") marks surround the attributes you assign to options, that is, the names of nodes, disks, scratch disks, and pools.

- Comments are demarcated by /* . . . */, in the style of the C programming language. They are optional, but are recommended where indicated in the examples.

Node names

Each node you define is followed by its name enclosed in quotation marks, for example:

```
node "orch0"
```

For a single CPU node or workstation, the node's name is typically the network name of a processing node on a connection such as a high-speed switch or Ethernet. Issue the following UNIX command to learn a node's network name:

```
$ uname -n
```

On an SMP, if you are defining multiple logical nodes corresponding to the same physical node, you replace the network name with a logical node name. In this case, you need a fast name for each logical node.

If you run an application from a node that is undefined in the corresponding configuration file, each user must set the environment variable APT_PM_CONDUCTOR_NODENAME to the fast name of the node invoking the parallel job.

Options

Each node takes options that define the groups to which it belongs and the storage resources it employs. Options are as follows:

`fastname`

Syntax:

```
fastname "name"
```

This option takes as its quoted attribute the name of the node as it is referred to on the fastest network in the system, such as an IBM switch, FDDI, or BYNET. The `fastname` is the physical node name that stages use to open connections for high volume data transfers. The attribute of this option is often the network name. For an SMP, all CPUs share a single connection to the network, and this setting is the same for all parallel engine processing nodes defined for an SMP. Typically, this is the principal node name, as returned by the UNIX command `uname -n`.

`pools`

Syntax:

```
pools "node_pool_name0" "node_pool_name1" ...
```

The `pools` option indicates the names of the pools to which this node is assigned. The option's attribute is the pool name or a space-separated list of names, each enclosed in quotation marks. For a detailed discussion of node pools, see "Node Pools and the Default Node Pool" .

Note that the `resource disk` and `resource scratchdisk` options can also take `pools` as an option, where it indicates disk or scratch disk pools. For a detailed discussion of disk and scratch disk pools, see "Disk and Scratch Disk Pools and Their Defaults" .

Node pool names can be dedicated. Reserved node pool names include the following names:

DB2 See the `DB2` resource below and "The resource DB2 Option" .

INFORMIX
See the `INFORMIX` resource below and "The resource INFORMIX Option" .

ORACLE See the `ORACLE` resource below and "The resource ORACLE option" .

sas See "The SAS Resources" .

sort See "Sort Configuration" .

Reserved disk pool names include the following names:

See "Buffer Scratch Disk Pools" .

export For use by the export stage.

lookup For use by the lookup stage.

sasdataset
See "The SAS Resources" .

sort See "Sort Configuration" .

`resource`

Syntax:

```
resource resource_type "location" [{pools "disk_pool_name"}] |
resource resource_type "value"
```

The *resource_type* can be one of the following:

`canonicalhostname`

Syntax:

```
canonicalhostname "ethernet name"
```

The canonicalhostname resource takes as its quoted attribute the ethernet name of a node in a cluster that is unconnected to the Conductor node by the high-speed network. If the Conductor node cannot reach the unconnected node by a fastname, you must define the unconnected node's canonicalhostname to enable communication.

`DB2`

Syntax:

```
resource DB2 "node_number" [{pools "instance_owner" ...}]
```

This option allows you to specify logical names as the names of DB2 nodes. For a detailed discussion of configuring DB2, see "The resource DB2 Option" .

`disk`

Syntax:

```
resource disk "directory_path"
[{pools "poolname"...}]
```

Assign to this option the quoted absolute path name of a directory belonging to a file system connected to the node. The node reads persistent data from and writes persistent data to this directory. One node can have multiple disks. Relative path names are not supported.

Typically, the quoted name is the root directory of a file system, but it does not have to be. For example, the quoted name you assign to disk can be a subdirectory of the file system.

You can group disks in `pools`. Indicate the pools to which a disk belongs by following its quoted name with a `pools` definition enclosed in braces. For a detailed discussion of disk pools, see "Disk and Scratch Disk Pools and Their Defaults" .

`INFORMIX`

Syntax:

```
resource INFORMIX "coserver_basename" [{pools "db_server_name" ... }]
```

This option allows you to specify logical names as the names of INFORMIX nodes. For a detailed discussion of configuring INFORMIX, see "The resource INFORMIX Option" .

`ORACLE`

Syntax:

```
resource ORACLE "nodename"
    [{pools "db_server_name" ...}]
```

This option allows you to define the nodes on which Oracle runs. For a detailed discussion of configuring Oracle, see "The resource ORACLE option" .

`sasworkdisk`

Syntax:

```
resource sasworkdisk "directory_path"
    [{pools "poolname"...}]
```

This option is used to specify the path to your SAS work directory. See "The SAS Resources" .

`scratchdisk`

Syntax:

```
resource scratchdisk "directory_path"
    [{pools "poolname"...}]
```

Assign to this option the quoted absolute path name of a directory on a file system where intermediate data will be temporarily stored. All users of this configuration must be able to read from and write to this directory. Relative path names are unsupported.

The directory should be local to the processing node and reside on a different spindle from that of any other disk space. One node can have multiple scratch disks.

Assign at least 500 MB of scratch disk space to each defined node. Nodes should have roughly equal scratch space. If you perform sorting operations, your scratch disk space requirements can be considerably greater, depending upon anticipated use.

You can ensure that:

- Every logical node in the configuration file that will run sorting operations have its own sort disk, where a *sort disk* is defined as a scratch disk available for sorting that resides in either the *sort* or default disk pool.
- Each logical node's sorting disk be a distinct disk drive. Alternatively, if it is shared among multiple sorting nodes, it should be *striped* to ensure better performance.
- For large sorting operations, each node that performs sorting have multiple distinct sort disks on distinct drives, or striped.

You can group scratch disks in pools. Indicate the pools to which a scratch disk belongs by following its quoted name with a pools definition enclosed in braces. For more information on disk pools, see "Disk and Scratch Disk Pools and Their Defaults" .

The following sample SMP configuration file defines four logical nodes.

```
{
      node "borodin0" {
          fastname "borodin"
          pools "compute_1" ""
```

```
                resource disk "/sfiles/node0" {pools ""}
          resource scratchdisk "/scratch0" {pools "" "sort"}
        }
       node "borodin1" {
          fastname "borodin"
          pools "compute_1" ""
          resource disk "/sfiles/node1" {pools ""}
              resource scratchdisk "/scratch1" {pools "" "sort"}
        }
       node "borodin2" {
                fastname "borodin"
          pools "compute_1" ""
                resource disk "/sfiles/node2" {pools ""}
          resource scratchdisk "/scratch2" {pools "" "sort"}
        }
        node "borodin3"
        {
                fastname "borodin"
             pools "compute_1" ""
             resource disk "/sfiles/node3" {pools ""}
                resource scratchdisk "/scratch3" {pools "" "sort"}
        }
}
```

In the example shown above:

- All nodes are elements of pool compute_1 and the default node pool, indicated by "".
- The resource disk of node borodin0 is the directory /sfiles/node0.
- The resource disks of nodes borodin1 to borodin3 are the directories /sfiles/node1, /sfiles/node2, and /sfiles/node3.
- All resource disks are elements of the default disk pool, indicated by "".
- For sorting, each logical node has its own scratch disk.
- All scratch disks are elements of the sort scratch disk pool and the default scratch disk pool which is indicated by "".

Node pools and the default node pool

Node pools allow association of processing nodes based on their characteristics. For example, certain nodes can have large amounts of physical memory, and you can designate them as compute nodes. Others can connect directly to a mainframe or some form of high-speed I/O. These nodes can be grouped into an I/O node pool.

The option pools is followed by the quoted names of the node pools to which the node belongs. A node can be assigned to multiple pools, as in the following example, where node1 is assigned to the default pool ("") as well as the pools node1, node1_css, and pool4.

```
node "node1"
{
fastname "node1_css"
pools "" "node1" "node1_css" "pool4"
resource disk "/orch/s0" {}
resource scratchdisk "/scratch" {}
}
```

A node belongs to the default pool unless you explicitly specify a pools list for it, and omit the default pool name ("") from the list.

Once you have defined a node pool, you can constrain a parallel stage or parallel job to run only on that pool, that is, only on the processing nodes belonging to it. If you constrain both an stage and a job, the stage runs only on the nodes that appear in both pools.

Nodes or resources that name a pool declare their membership in that pool.

When you initially configure your system you can place all nodes in pools that are named after the node's name and fast name. Additionally include the default node pool in this pool, as in the following example:

```
node "n1"
{
fastname "nfast"
pools "" "n1" "nfast"
}
```

By default, the parallel engine executes a parallel stage on all nodes defined in the default node pool. You can constrain the processing nodes used by the parallel engine either by removing node descriptions from the configuration file or by constraining a job or stage to a particular node pool.

Disk and scratch disk pools and their defaults

When you define a processing node, you can specify the options resource disk and resource scratchdisk. They indicate the directories of file systems available to the node. You can also group disks and scratch disks in pools. Pools reserve storage for a particular use, such as holding very large data sets. The syntax for setting up disk and scratch disk pools is as follows:

```
resource disk "disk_name"
     {pools "disk_pool0" ... "disk_poolN"}
  resource scratchdisk "s_disk_name"
      {pools "s_pool0" ... "s_poolN"}
```

where:

- *disk_name* and *s_disk_name* are the names of directories.
- *disk_pool*... and *s_pool*... are the names of disk and scratch disk pools, respectively.

Pools defined by disk and scratchdisk are not combined; therefore, two pools that have the same name and belong to both resource disk and resource scratchdisk define two separate pools.

A disk that does not specify a pool is assigned to the default pool. The default pool might also be identified by "" by and { } (the empty pool list). For example, the following code configures the disks for node1:

```
node "node1" {
   resource disk "/orch/s0" {pools "" "pool1"}
   resource disk "/orch/s1" {pools "" "pool1"}
   resource disk "/orch/s2" { } /* empty pool list */
   resource disk "/orch/s3" {pools "pool2"}
   resource scratchdisk "/scratch"      {pools "" "scratch_pool1"}
}
```

In this example:

- The first two disks are assigned to the default pool.
- The first two disks are assigned to pool1.
- The third disk is also assigned to the default pool, indicated by { }.
- The fourth disk is assigned to pool2 and is not assigned to the default pool.
- The scratch disk is assigned to the default scratch disk pool and to scratch_pool1.

Application programmers make use of pools based on their knowledge of both their system and their application.

Buffer scratch disk pools

Under certain circumstances, the parallel engine uses both memory and disk storage to buffer virtual data set records.The amount of memory defaults to 3 MB per buffer per processing node. The amount of disk space for each processing node defaults to the amount of available disk space specified in the default `scratchdisk` setting for the node.

The parallel engine uses the default scratch disk for temporary storage other than buffering. If you define a `buffer` scratch disk pool for a node in the configuration file, the parallel engine uses that scratch disk pool rather than the default scratch disk for buffering, and all other scratch disk pools defined are used for temporary storage other than buffering.

Here is an example configuration file that defines a buffer scratch disk pool:

```
{
   node node1 {
       fastname "node1_css"
       pools "" "node1" "node1_css"
       resource disk "/orch/s0" {}
       resource scratchdisk "/scratch0" {pools "buffer"}
       resource scratchdisk "/scratch1" {}
       }
   node node2 {
       fastname "node2_css"
       pools "" "node2" "node2_css"
       resource disk "/orch/s0" {}
       resource scratchdisk "/scratch0" {pools "buffer"}
       resource scratchdisk "/scratch1" {}
       }
}
```

In this example, each processing node has a single scratch disk resource in the buffer pool, so buffering will use /scratch0 but not /scratch1. However, if /scratch0 were not in the buffer pool, both /scratch0 and /scratch1 would be used because both would then be in the default pool.

The resource DB2 option

The DB2 file db2nodes.cfg contains information for translating DB2 node numbers to node names. You must define the node names specified in db2nodes.cfg in your configuration file, if you want the parallel engine to communicate with DB2. You can designate each node specified in db2nodes.cfg in one of the following ways:

- By assigning to node its quoted network name, as returned by the UNIX operating system command uname -n; for example, node "node4".
- By assigning to node a logical name, for example "DB2Node3". If you do so, you must specify the option resource DB2 followed by the node number assigned to the node in db2nodes.cfg.

The resource DB2 option can also take the pools option. You assign to it the user name of the owner of each DB2 instance configured to run on each node. DB2 uses the instance to determine the location of db2nodes.cfg.

Here is a sample DB2 configuration:

```
{
   node "Db2Node0" {
      /* other configuration parameters for node0 */
      resource DB2 "0" {pools "Mary" "Tom"}
   }
   node "Db2Node1" {
      /* other configuration parameters for node1 */
      resource DB2 "1" {pools "Mary" "Tom"}
   }
   node "Db2Node2" {
```

```
      /* other configuration parameters for node2 */
      resource DB2 "2" {pools "Mary" "Tom" "Bill"}
   }

   node "Db2Node3" {
      /* other configuration parameters for node3 */
      resource DB2 "3" {pools "Mary" "Bill"}
   }

   /* other nodes used by the parallel engine*/

}
```

In the example above:

- The resource DB2 option takes the DB2 node number corresponding to the processing node.
- All nodes are used with the DB2 instance Mary.
- Nodes 0, 1, and 2 are used with the DB2 instance Tom.
- Nodes 2 and 3 are used with the DB2 instance Bill.

If you now specify a DB2 instance of Mary in your jobs, the location of db2nodes.cfg is ~Mary/sqllib/db2nodes.cfg.

The resource INFORMIX option

To communicate with INFORMIX, the parallel engine must be configured to run on all processing nodes functioning as INFORMIX coservers. This means that the configuration must include a node definition for the coserver nodes. The list of INFORMIX coservers is contained in the file pointed to by the environment variable $INFORMIXSQLHOSTS or in the file $INFORMIXDIR/etc/sqlhosts.

There are two methods for specifying the INFORMIX coserver names in the configuration file.

1. Your configuration file can contain a description of each node, supplying the node name (not a synonym) as the quoted name of the node. Typically, the node name is the network name of a processing node as returned by the UNIX command uname -n.

   Here is a sample configuration file for a system containing INFORMIX coserver nodes node0, node1, node2, and node3:

```
{
       node "node0" {
      /* configuration parameters for node0 */
       }
       node "node1" {
      /* configuration parameters for node1 */
      }
       node "node2" {
      /* configuration parameters for node2 */
       }
       node "node3" {
      /* configuration parameters for node3 */
       }

       /* other nodes used by the parallel engine*/
}
```

1. You can supply a logical rather than a real network name as the quoted name of node. If you do so, you must specify the resource INFORMIX option followed by the name of the corresponding INFORMIX coserver.

   Here is a sample INFORMIX configuration:

```
{
       node "IFXNode0" {
      /* other configuration parameters for node0 */
```

```
      resource INFORMIX "node0" {pools "server"}
       }
       node "IFXNode1" {
      /* other configuration parameters for node1 */
      resource INFORMIX "node1" {pools "server"}
       }
       node "IFXNode2" {
      /* other configuration parameters for node2 */
      resource INFORMIX "node2" {pools "server"}
       }
       node "IFXNode3" {
      /* other configuration parameters for node3 */
      resource INFORMIX "node3" {pools "server"}
       }

   /* other nodes used by the parallel engine*/
    }
```

When you specify resource INFORMIX, you must also specify the pools parameter. It indicates the base name of the coserver groups for each INFORMIX server. These names must correspond to the coserver group base name using the shared-memory protocol. They also typically correspond to the DBSERVERNAME setting in the ONCONFIG file. For example, coservers in the group server are typically named server.1, server.2, and so on.

The resource ORACLE option

By default, the parallel engine executes Oracle stages on all processing nodes belonging to the default node pool, which typically corresponds to all defined nodes.

You can optionally specify the resource ORACLE option to define the nodes on which you want to run the Oracle stages. If you do, the parallel engine runs the Oracle stages only on the processing nodes for which resource ORACLE is defined. You can additionally specify the pools parameter of resource ORACLE to define resource pools, which are groupings of Oracle nodes.

Here is a sample Oracle configuration:

```
{
   node "node0" {
      /* other configuration parameters for node0 */
      resource ORACLE "node0" {pools "group1" "group2"                    "group3"}
   }
   node "node1" {
      /* other configuration parameters for node1 */
      resource ORACLE "node1" {pools "group1" "group2"}
   }
   node "node2" {
      /* other configuration parameters for node2 */
         resource ORACLE "node2" {pools "group1" "group3"}
   }
   node "node3" {
      /* other configuration parameters for node3 */
      resource ORACLE "node3" {pools "group1" "group2"         "group3"}
   }

   /* any other nodes used by the parallel engine*/
}
```

In the example above, Oracle runs on node0 to node3.

- node0-node3 are used with node pool group1.
- node0, node1, and node3 are used with node pool group2.
- node0, node2, and node3 are used with node pool group3.

The SAS resources

Adding SAS information to your configuration file

About this task

To configure your system to use the SAS stage, you need to specify the following information in your configuration file:

- The location of the SAS executable, if it is not in your PATH;
- An SAS work disk directory, one for each parallel engine node;
- Optionally, a disk pool specifically for parallel SAS data sets, called sasdataset.

The resource names sas and sasworkdisk and the disk pool name sasdataset are all reserved words. Here is an example of each of these declarations:

```
resource sas "/usr/sas612/" { }
resource sasworkdisk "/usr/sas/work/" { }
resource disk "/data/sas/" {pools "" "sasdataset"}
```

While the disks designated as sasworkdisk need not be a RAID configuration, best performance will result if each parallel engine logical node has its own reserved disk that is not shared with other parallel engine nodes during sorting and merging. The total size of this space for all nodes should optimally be equal to the total work space you use when running SAS sequentially (or a bit more, to allow for uneven distribution of data across partitions).

The number of disks in the sasdataset disk pool is the degree of parallelism of parallel SAS data sets. Thus if you have 24 processing nodes, each with its associated disk in the sasdataset disk pool, parallel SAS data sets will be partitioned among all 24 disks, even if the operation preceding the disk write is, for example, only four-way parallel.

Defining SAS paths

Here a single node, grappelli0, is defined, along with its fast name. Also defined are the path to a SAS executable, a SAS work disk (corresponding to the SAS work directory), and two disk resources, one for parallel SAS data sets and one for non-SAS file sets.

```
node "grappelli0"
{
fastname "grappelli"
pools "" "a"
resource sas "/usr/sas612" { }
resource scratchdisk "/scratch" { }
resource sasworkdisk "/scratch" { }
disk "/data/pds_files/node0" { pools "" "export" }
disk "/data/pds_files/sas" { pools "" "sasdataset" }
 }
```

Sort configuration

About this task

You might want to define a sort scratch disk pool to assign scratch disk space explicitly for the storage of temporary files created by the Sort stage. In addition, if only a subset of the nodes in your configuration have sort scratch disks defined, you can define a sort node pool, to specify the nodes on which the sort stage should run. Nodes assigned to the sort node pool should be those that have scratch disk space assigned to the sort scratch disk pool.

The parallel engine then runs sort only on the nodes in the sort node pool, if it is defined, and otherwise uses the default node pool. The Sort stage stores temporary files only on the scratch disks included in the sort scratch disk pool, if any are defined, and otherwise uses the default scratch disk pool.

When the parallel engine runs, it determines the locations of temporary files by using the following procedure.

Procedure

1. Searching the parallel engine configuration for any scratch disk resources in the sort resource pool on the nodes sort will run on. If found, the scratch disks are used as a location for temporary storage by sort.
2. If no scratch disk resources are found that belong to the disk pool sort, the system determines whether any scratch disk resources belong to the default scratch disk pool on the nodes sort will run on. If so, the scratch disks belonging to the default pool are used by tsort for temporary storage.
3. If no scratch disk resources are found that belong to either sort or the default scratch disk pool, the parallel engine issues a warning message and runs sort using the directory indicated by the TMPDIR environment variable or /tmp for temporary storage.

Allocation of resources

The allocation of resources for a given stage, particularly node and disk allocation, is done in a multi-phase process. Constraints on which nodes and disk resources are used are taken from the parallel engine arguments, if any, and matched against any pools defined in the configuration file. Additional constraints might be imposed by, for example, an explicit requirement for the same degree of parallelism as the previous stage. After all relevant constraints have been applied, the stage allocates resources, including instantiation of Player processes on the nodes that are still available and allocation of disks to be used for temporary and permanent storage of data.

Selective configuration with startup scripts

About this task

As part of running an application, the parallel engine creates a remote shell on all parallel engine processing nodes on which the application will be executed. After the parallel engine creates the remote shell, it copies the environment from the system on which the application was invoked to each remote shell. This means that all remote shells have the same configuration by default.

However, you can override the default and set configuration parameters for individual processing nodes. To do so, you create a parallel engine startup script. If a startup script exists, the parallel engine runs it on all remote shells before it runs your application.

When you invoke an application, the parallel engine looks for the name and location of a startup script as in the following procedure.

Procedure

1. It uses the value of the APT_STARTUP_SCRIPT environment variable.
2. It searches the current working directory for a file named `startup.apt`.
3. Searches for the file `install_dir/etc/startup.apt` on the system that invoked the parallel engine application, where `install_dir` is the top-level directory of the installation.
4. If the script is not found, it does not execute a startup script.

Results

Here is a template you can use with Korn shell to write your own startup script.

```
#!/bin/ksh               # specify Korn shell
# your shell commands go here

shift 2              # required for all shells
exec $*              # required for all shells
```

You must include the last two lines of the shell script. This prevents your application from running if your shell script detects an error.

The following startup script for the Bourne shell prints the node name, time, and date for all processing nodes before your application is run:

```
#!/bin/sh                # specify Bourne shell

echo `hostname` `date`
shift 2
exec $*
```

A single script can perform node-specific initialization by means of a case statement. In the following example, the system has two nodes named node1 and node2. This script performs initialization based on which node it is running on.

```
#!/bin/sh                    # use Bourne shell

# Example APT startup script.
case `hostname` in
   node1)
      # perform node1 init
      node-specific directives         ;;
   node2)
      # perform node2 init
      node-specific directives         ;;
esac
shift 2
exec $*
```

The parallel engine provides the APT_NO_STARTUP_SCRIPT environment variable to prevent the parallel engine from running the startup script. By default, the parallel engine executes the startup script. If the variable is set, the parallel engine ignores the startup script. This can be useful for debugging a startup script.

Hints and tips

The configuration file tells the engine how to exploit the underlying computer system. For a given system there is not necessarily one ideal configuration file because of the high variability between the way different jobs work. So where do you start?

Let's assume you are running on a shared-memory multi-processor system, that is, an SMP box (these are the most common platforms today). Let's assume these properties. You can adjust the illustration below to match your precise situation:

- computer's hostname "fastone"
- 6 CPUs
- 4 separate file systems on 4 drives named /fs0 /fs1 /fs2 /fs3

The configuration file to use as a starting point would look like the one below. Note the way the disk/scratchdisk resources are handled. That's the real trick here.

```
{ /* config file allows C-style comments. */
/*
   config files look like they have flexible syntax.
   They do NOT. Keep all the sub-items of the individual
   node specifications in the order shown here.
```

```
*/
node "n0" {
pools "" /* on an SMP node pools aren't used often. */
fastname "fastone"
resource scratchdisk "/fs0/ds/scratch" {} /*start with fs0*/
resource scratchdisk "/fs1/ds/scratch" {}
resource scratchdisk "/fs2/ds/scratch" {}
resource scratchdisk "/fs3/ds/scratch" {}
resource disk "/fs0/ds/disk" {} /* start with fs0 */
resource disk "/fs1/ds/disk" {}
resource disk "/fs2/ds/disk" {}
resource disk "/fs3/ds/disk" {}
}
node "n1" {pools ""
fastname "fastone"
resource scratchdisk "/fs1/ds/scratch" {} /*start with fs1*/
resource scratchdisk "/fs2/ds/scratch" {}
resource scratchdisk "/fs3/ds/scratch" {}
resource scratchdisk "/fs0/ds/scratch" {}
resource disk "/fs1/ds/disk" {} /* start with fs1 */
resource disk "/fs2/ds/disk" {}
resource disk "/fs3/ds/disk" {}
resource disk "/fs0/ds/disk" {}
}
node "n2" {
pools ""
fastname "fastone"
resource scratchdisk "/fs2/ds/scratch" {} /*start with fs2*/
resource scratchdisk "/fs3/ds/scratch" {}
resource scratchdisk "/fs0/ds/scratch" {}
resource scratchdisk "/fs1/ds/scratch" {}
resource disk "/fs2/ds/disk" {} /* start with fs2 */
resource disk "/fs3/ds/disk" {}
resource disk "/fs0/ds/disk" {}
resource disk "/fs1/ds/disk" {}
}
node "n3" {
pools ""
fastname "fastone"
resource scratchdisk "/fs3/ds/scratch" {} /*start with fs3*/
resource scratchdisk "/fs0/ds/scratch" {}
resource scratchdisk "/fs1/ds/scratch" {}
resource scratchdisk "/fs2/ds/scratch" {}
resource disk "/fs3/ds/disk" {} /* start with fs3 */
resource disk "/fs0/ds/disk" {}
resource disk "/fs1/ds/disk" {}
resource disk "/fs2/ds/disk" {}
}
node "n4" {
pools ""
fastname "fastone"
/*
*    Ok, now what. We rotated through starting with a
*   different disk, but we have a basic problem here which is
*   that there are more CPUs than disks. So what do we do
*   now? The answer: something that is not perfect. We're
*   going to repeat the sequence. You could shuffle
*   differently that is, use /fs0 /fs2 /fs1 /fs3 as an order.
*   I'm not sure it will matter all that much.
*/
resource scratchdisk "/fs0/ds/scratch" {}          /*start with fs0
                                        again*/
resource scratchdisk "/fs1/ds/scratch" {}
resource scratchdisk "/fs2/ds/scratch" {}
resource scratchdisk "/fs3/ds/scratch" {}
resource disk "/fs0/ds/disk" {} /* start with fs0 again */
resource disk "/fs1/ds/disk" {}
```

```
resource disk "/fs2/ds/disk" {}
resource disk "/fs3/ds/disk" {}
}
node "n5" {
pools ""
fastname "fastone"
resource scratchdisk "/fs1/ds/scratch" {} /*start with fs1*/
resource scratchdisk "/fs2/ds/scratch" {}
resource scratchdisk "/fs3/ds/scratch" {}
resource scratchdisk "/fs0/ds/scratch" {}
resource disk "/fs1/ds/disk" {} /* start with fs1 */
resource disk "/fs2/ds/disk" {}
resource disk "/fs3/ds/disk" {}
resource disk "/fs0/ds/disk" {}
}
} /* end of whole config */
```

The above config file pattern could be called "give everyone all the disk". This configuration style works well when the flow is complex enough that you can't really figure out and precisely plan for good I/O utilization. Giving every partition (node) access to all the I/O resources can cause contention, but the parallel engine tends to use fairly large blocks for I/O so the contention isn't as much of a problem as you might think. This configuration style works for any number of CPUs and any number of disks since it does not require any particular correspondence between them. The heuristic principle at work here is this "When it's too difficult to figure out precisely, at least go for achieving balance."

The alternative to the above configuration style is more careful planning of the I/O behavior so as to reduce contention. You can imagine this could be hard given the hypothetical 6-way SMP with 4 disks because setting up the obvious one-to-one correspondence does not work. Doubling up some nodes on the same disk is unlikely to be good for overall performance since it creates a hotspot. You could give every CPU 2 disks and rotate around, but that would be little different than the above strategy. So, let us imagine a less constrained environment and give ourselves 2 more disks /fs4 and /fs5. Now a config file might look like this:

```
{
node "n0" {
pools ""
fastname "fastone"
resource scratchdisk "/fs0/ds/scratch" {}
resource disk "/fs0/ds/disk" {}
}
node "n1" {
pools ""
fastname "fastone"
resource scratchdisk "/fs1/ds/scratch" {}
resource disk "/fs1/ds/disk" {}
}
node "n2" {
pools ""
fastname "fastone"
resource scratchdisk "/fs2/ds/scratch" {}
resource disk "/fs2/ds/disk" {}
}
node "n3" {
pools ""
fastname "fastone"
resource scratchdisk "/fs3/ds/scratch" {}
resource disk "/fs3/ds/disk" {}
}
node "n4" {
pools ""
fastname "fastone"
resource scratchdisk "/fs4/ds/scratch" {}
resource disk "/fs4/ds/disk" {}
}
node "n5" {
```

```
pools ""
fastname "fastone"
resource scratchdisk "/fs5/ds/scratch" {}
resource disk "/fs5/ds/disk" {}
}
} /* end of whole config */
```

This is simplest, but realize that no single player (stage/operator instance) on any one partition can go faster than the single disk it has access to.

You could combine strategies by adding in a node pool where disks have this one-to-one association with nodes. These nodes would then not be in the default node pool, but a special one that you would assign stages/operators to specifically.

Other configuration file hints:

- Consider avoiding the disk/disks that your input files reside on. Often those disks will be hotspots until the input phase is over. If the job is large and complex this is less of an issue since the input part is proportionally less of the total work.
- Ensure that the different file systems mentioned as the disk and scratchdisk resources hit disjoint sets of spindles even if they're located on a RAID system.
- Know what is real and what is NFS: Real disks are directly attached, or are reachable over a SAN (storage-area network - dedicated, just for storage, low-level protocols).
  - Never use NFS file systems for scratchdisk resources.
  - If you use NFS file system space for disk resources, then you need to know what you are doing. For example, your final result files might need to be written out onto the NFS disk area, but that does not mean the intermediate data sets created and used temporarily in a multi-job sequence should use this NFS disk area. Better to setup a "final" disk pool, and constrain the result sequential file or data set to reside there, but let intermediate storage go to local or SAN resources, not NFS.
- Know what data points are striped (RAID) and which are not. Where possible, avoid striping across data points that are already striped at the spindle level.

Chapter 14. Grid deployment

Grid computing can improve the ability of any IT organization to maximize resource value. Information integration solutions that are built on grid technology can increase computing capacity at a lower cost.

Resource manager software dynamically examines and allocates resources, enabling your IT group to be much more responsive to the needs of the enterprise while getting the most out of investments in information technology.

You can run your parallel jobs in a grid environment by using the grid configuration facilities supplied by IBM InfoSphere DataStage. This facility is available on Linux systems with resource management provided by Tivoli® Workload Scheduler LoadLeveler®.

Resource manager software requirements

The Tivoli Workload Scheduler LoadLeveler software has some requirements that you must consider when you add or remove compute nodes or users and when you need to tune your performance.

The following list describes the requirements for Tivoli Workload Scheduler LoadLeveler software settings when you add users to the system:

- Each new Tivoli Workload Scheduler LoadLeveler user must have the same user ID and group ID on the conductor node and on all the compute nodes.
- Every new Tivoli Workload Scheduler LoadLeveler user must be able to run SSH without a password.
- The administrator must verify that the Tivoli Workload Scheduler LoadLeveler executable directory (`/opt/ibmll/LoadL/full/bin`) is included in the environment variable PATH of the conductor node.
- If you add a new compute node to the system, the administrator must add *`compute_node_name`*`: type = machine` to the machine stanza of the LoadL_admin file that is on the conductor node.

Depending on the length of the jobs that you are running, you might want to adjust the wall clock limit. The limit is set to 100 hours by default. To change the wall clock limit:

1. Edit the file `/home/load/LoadL_admin`.
2. Locate the class stanza defined for the parallel class.
3. Change the wall_clock_limit parameter in the parallel class to a size that will be able to accommodate your jobs (you can specify the value in seconds or hours:minutes:seconds).

   Limit specified in hours, minutes, and seconds:

   ```
   wall_clock_limit = 200:00:00
   ```

   Limit specified in seconds:

   ```
   wall_clock_limit = 3600
   ```

Configuring the grid environment

You must configure your grid environment before you can run jobs on a grid system.

Before you begin

You must have already installed IBM InfoSphere Information Server.

Procedure

1. Install IBM Tivoli LoadLeveler 3.4.2 (loadl)
2. Configure passwordless log in for loadl from the conductor node to all compute nodes.

3. Configure LoadL_admin, LoadL_config, and LoadL_config.local on each machine. The LoadL_admin and LoadL_config files are the same on the conductor and all compute nodes. The LoadL_config.local is the same on all compute nodes, but different from that on the conductor node.
4. Create the configuration file $APT_ORCHHOME/etc/master_config.apt (mode must be 644).
5. Create $APT_ORCHHOME/etc/resource_manager (mode must be 644).
6. Start loadl by using the following command: `llctl -g start` (you can stop loadl by using the command `llctl -g stop`). Your PATH must include /opt/ibmll/LoadL/full/bin.
7. Check /etc/hosts file on conductor and each compute node, make sure these two lines are included in each file:
   - 127.0.0.1 localhost.localdomain localhost
   - IP_Adress short_host_name long_host_name with domain
8. Create $APT_ORCHHOME/etc/remsh and replace rsh with ssh
9. Enable grid support for parallel jobs on the project level by starting the IBM InfoSphere DataStage Administrator client and doing the following steps:
   a. Open the Properties window for the project that you are deploying on a grid.
   b. In the Properties window, click **Parallel** to open the Parallel page.
   c. Select the **Enable grid support for parallel jobs** option.

Example LoadL_admin file

You must configure the LoadL_admin file on your compute node, and each of your compute nodes.

In the following example, the fields that are specific to your environment (and therefore need customizing) are shown in **bold**.

```
# LoadL_admin file: Remove comments and edit this file to suit your installation.
# This file consists of machine, class, user, group and adapter stanzas.
# Each stanza has defaults, as specified in a "defaults:" section.
# Default stanzas are used to set specifications for fields which are
# not specified.
# Class, user, group, and adapter stanzas are optional.  When no adapter
# stanzas are specified, LoadLeveler determines adapters dynamically.  Refer to
# the Grid deployment topic for detailed information about
# keywords and associated values.  Also see LoadL_admin.1 in the
# ~loadl/samples directory for sample stanzas.
################################################################################
# DEFAULTS FOR MACHINE, CLASS, USER, AND GROUP STANZAS:
# Remove initial # (comment), and edit to suit.
#
default:   type = machine
      central_manager = false
      schedd_host = false
      submit_only = false machine

default:   type = class
      wall_clock_limit = 100:00:00

default:   type = user
      default_class = parallel
      default_group = No_Group
             default_interactive_class = inter_class

default:   type = group

################################################################################
# MACHINE STANZAS:
# These are the machine stanzas; the first machine is defined as
# the central manager.  mach1:, mach2:, etc. are machine name labels -
# revise these placeholder labels with the names of the machines in the
# pool, and specify any schedd_host and submit_only keywords and values
```

```
# (true or false), if required.
##############################################################################
PXRH4vm1.swg.usma.ibm.com: type = machine
          central_manager = true
     schedd_host = true
       master_node_exclusive = true

#-- SET OF COMPUTE BLADES
PXRH4vm2.swg.usma.ibm.com: type = machine
PXRH4vm3.swg.usma.ibm.com: type = machine

parallel:      type = class
       master_node_requirement = true
               priority = 100
              wall_clock_limit = 100:00:00

inter_class:    type = class
                wall_clock_limit = 30:00,25:00
```

Example LoadL_config file

You must configure the LoadL_config file on your compute node, and each of your compute nodes.

The following is an example LoadL_config file.

```
#
# Machine Description
#
ARCH = i386

#
#  Specify LoadLeveler Administrators here:
#
LOADL_ADMIN = loadl loadladm xpu dsadm

#
# Default to starting LoadLeveler daemons when requested
#
START_DAEMONS = TRUE

#
# Machine authentication
#
# If TRUE, only connections from machines in the ADMIN_LIST are accepted.
# If FALSE, connections from any machine are accepted.  Default if not
# specified is FALSE.
#
MACHINE_AUTHENTICATE = FALSE

#
# Specify which daemons run on each node
#
SCHEDD_RUNS_HERE = True
STARTD_RUNS_HERE = True

#
# Specify information for backup central manager
#
# CENTRAL_MANAGER_HEARTBEAT_INTERVAL = 300
# CENTRAL_MANAGER_TIMEOUT = 6

#
# Specify pathnames
#
RELEASEDIR       = /opt/ibmll/LoadL/full
LOCAL_CONFIG   = $(tilde)/LoadL_config.local
ADMIN_FILE       = $(tilde)/LoadL_admin
```

```
LOG            = $(tilde)/log
SPOOL           = $(tilde)/spool
EXECUTE     = $(tilde)/execute
HISTORY     = $(SPOOL)/history
RESERVATION_HISTORY = $(SPOOL)/reservation_history
BIN       = $(RELEASEDIR)/bin
LIB       = $(RELEASEDIR)/lib

#
# Specify port numbers
#
MASTER_STREAM_PORT     = 9616
NEGOTIATOR_STREAM_PORT = 9614
SCHEDD_STREAM_PORT     = 9605
STARTD_STREAM_PORT     = 9611
COLLECTOR_DGRAM_PORT   = 9613
STARTD_DGRAM_PORT      = 9615
MASTER_DGRAM_PORT      = 9617

#
# Specify a scheduler type:  LL_DEFAULT, API, BACKFILL
#   API specifies that internal LoadLeveler scheduling algorithms be
#   turned off and LL_DEFAULT specifies that the original internal
#   LoadLeveler scheduling algorithm be used.
#
SCHEDULER_TYPE = BACKFILL

#
# Specify a secure shell
#
LL_RSH_COMMAND=/usr/bin/ssh

#
# Specify accounting controls
# To turn reservation data recording on, add the flag A_RES to ACCT
#
ACCT       = A_OFF A_RES
ACCT_VALIDATION  = $(BIN)/llacctval
GLOBAL_HISTORY  = $(SPOOL)

#
# Specify prolog and epilog path names
#
# JOB_PROLOG =
# JOB_EPILOG =
# JOB_USER_PROLOG =
# JOB_USER_EPILOG =

#
# Specify checkpointing intervals
#
MIN_CKPT_INTERVAL    = 900
MAX_CKPT_INTERVAL    = 7200

# perform cleanup of checkpoint files once a day
# 24 hrs x 60 min/hr x 60 sec/min = 86400 sec/day

CKPT_CLEANUP_INTERVAL = 86400

# sample source for the ckpt file cleanup program is shipped with LoadLeveler
# and is found in:  /usr/lpp/LoadL/full/samples/llckpt/rmckptfiles.c
#
# compile the source and indicate the location of the executable
# as shown in the following example

CKPT_CLEANUP_PROGRAM = /u/mylladmin/bin/rmckptfiles
```

```
# LoadL_KeyboardD Macros
#
KBDD                = $(BIN)/LoadL_kbdd
KBDD_LOG            = $(LOG)/KbdLog
MAX_KBDD_LOG        = 64000
KBDD_DEBUG          =

#
# Specify whether to start the keyboard daemon
#
#if HAS_X
X_RUNS_HERE   = True
#else
X_RUNS_HERE   = False
#endif


#
#  LoadL_StartD Macros
#
STARTD          = $(BIN)/LoadL_startd
STARTD_LOG        = $(LOG)/StartLog
MAX_STARTD_LOG       = 64000
STARTD_DEBUG       =
POLLING_FREQUENCY     = 5
POLLS_PER_UPDATE     = 24
JOB_LIMIT_POLICY     = 120
JOB_ACCT_Q_POLICY     = 300
PROCESS_TRACKING     = FALSE
PROCESS_TRACKING_EXTENSION = $(BIN)

#DRAIN_ON_SWITCH_TABLE_ERROR = false
#RESUME_ON_SWITCH_TABLE_ERROR_CLEAR = false

#ifdef KbdDeviceName
KBD_DEVICE              = KbdDeviceName
#endif
#ifdef MouseDeviceName
MOUSE_DEVICE            = MouseDeviceName
#endif



#
#  LoadL_SchedD Macros
#
SCHEDD       = $(BIN)/LoadL_schedd
SCHEDD_LOG     = $(LOG)/SchedLog
MAX_SCHEDD_LOG   = 64000
SCHEDD_DEBUG    =
SCHEDD_INTERVAL   = 120

CLIENT_TIMEOUT   = 30


#
# Negotiator Macros
#
NEGOTIATOR      = $(BIN)/LoadL_negotiator
NEGOTIATOR_DEBUG   =
NEGOTIATOR_LOG        = $(LOG)/NegotiatorLog
MAX_NEGOTIATOR_LOG      = 64000
NEGOTIATOR_INTERVAL      = 60
MACHINE_UPDATE_INTERVAL        = 300
NEGOTIATOR_PARALLEL_DEFER   = 300
NEGOTIATOR_PARALLEL_HOLD    = 300
NEGOTIATOR_REDRIVE_PENDING   = 90
```

```
NEGOTIATOR_RESCAN_QUEUE    = 90
NEGOTIATOR_REMOVE_COMPLETED  = 0
NEGOTIATOR_CYCLE_DELAY         = 0
NEGOTIATOR_CYCLE_TIME_LIMIT    = 0

#
# Sets the interval between recalculation of the SYSPRIO values
# for all the jobs in the queue
#
NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL = 0

#
# GSmonitor Macros
#
GSMONITOR                       = $(BIN)/LoadL_GSmonitor
GSMONITOR_DEBUG                 =
GSMONITOR_LOG                   = $(LOG)/GSmonitorLog
MAX_GSMONITOR_LOG               = 64000

#
# Consumable Resources
#
SCHEDULE_BY_RESOURCES = ConsumableCpus LicenseA FloatingLicenseX

FLOATING_RESOURCES = FloatingLicenseX(5) FloatingLicenseZ(2)

#
# Starter Macros
#
STARTER        = $(BIN)/LoadL_starter
STARTER_DEBUG      =
STARTER_LOG              = $(LOG)/StarterLog
MAX_STARTER_LOG          = 64000

#
# LoadL_Master Macros
#
MASTER         = $(BIN)/LoadL_master
MASTER_LOG       = $(LOG)/MasterLog
MASTER_DEBUG      =
MAX_MASTER_LOG     = 64000
RESTARTS_PER_HOUR    = 12
PUBLISH_OBITUARIES   = TRUE
OBITUARY_LOG_LENGTH   = 25

#
# Specify whether log files are truncated when opened
#
TRUNC_MASTER_LOG_ON_OPEN     = False
TRUNC_STARTD_LOG_ON_OPEN     = False
TRUNC_SCHEDD_LOG_ON_OPEN     = False
TRUNC_KBDD_LOG_ON_OPEN       = False
TRUNC_STARTER_LOG_ON_OPEN    = False
TRUNC_NEGOTIATOR_LOG_ON_OPEN = False
TRUNC_GSMONITOR_LOG_ON_OPEN  = False

#
# Machine control expressions and macros
#

OpSys : "$(OPSYS)"
Arch   : "$(ARCH)"
Machine : "$(HOST).$(DOMAIN)"

#
# Expressions used to control starting and stopping of foreign jobs
#
```

```
MINUTE      = 60
HOUR       = (60 * $(MINUTE))
StateTimer    = (CurrentTime - EnteredCurrentState)

BackgroundLoad  = 0.7
HighLoad     = 1.5
StartIdleTime   = 15 * $(MINUTE)
ContinueIdleTime = 5 * $(MINUTE)
MaxSuspendTime  = 10 * $(MINUTE)
MaxVacateTime   = 10 * $(MINUTE)

KeyboardBusy   = KeyboardIdle < $(POLLING_FREQUENCY)
CPU_Idle     = LoadAvg <= $(BackgroundLoad)
CPU_Busy     = LoadAvg >= $(HighLoad)

#
#  See the Grid deployment topic for an explanation of these
#  control expressions
#
# START  : $(CPU_Idle) && KeyboardIdle > $(StartIdleTime)
# SUSPEND : $(CPU_Busy) || $(KeyboardBusy)
# CONTINUE : $(CPU_Idle) && KeyboardIdle > $(ContinueIdleTime)
# VACATE : $(StateTimer) > $(MaxSuspendTime)
# KILL  : $(StateTimer) > $(MaxVacateTime)

START  : T
SUSPEND  : F
CONTINUE : T
VACATE  : F
KILL  : F

#
# Expressions used to prioritize job queue
#
# Values which can be part of the SYSPRIO expression are:
#
# QDate    Job submission time
# UserPrio   User priority
# UserSysprio   System priority value based on userid (from the user
#     list file with default of 0)
# ClassSysprio   System priority value based on job class (from the
#     class list file with default of 0)
# GroupSysprio      System priority value based on the group (from the
#     group list file with default of 0)
# GroupQueuedJobs   Number of job steps running or queued for the group
# GroupRunningJobs  Number of job steps running for the group
# GroupTotalJobs    Total number of job steps for the group
# UserQueuedJobs    Number of job steps running or queued for the user
# UserRunningJobs   Number of job steps running for the user
# UserTotalJobs     Total number of job steps for the user
#
# The following expression is an example.
#
#SYSPRIO: (ClassSysprio * 100) + (UserSysprio * 10) + (GroupSysprio * 1)- (QDate)
#
# The following (default) expression for SYSPRIO creates a FIFO job queue.
#
SYSPRIO: 0 - (QDate)

#
# Expressions used to prioritize machines
#
# The following example orders machines by the load average
# normalized for machine speed:
#
#MACHPRIO: 0 - (1000 * (LoadAvg / (Cpus * Speed)))
#
```

```
# The following (default) expression for MACHPRIO orders
# machines by load average.
#
MACHPRIO: 0 - (LoadAvg)
#
# The MAX_JOB_REJECT value determines how many times a job can be
# rejected before it is canceled or put on hold.  The default value
# is 0, which indicates a rejected job will immediately be canceled
# or placed on hold. MAX_JOB_REJECT may be set to unlimited rejects
# by specifying a value of -1.
#
MAX_JOB_REJECT = 0
#
# When ACTION_ON_MAX_REJECT is HOLD, jobs will be put on user hold
# when the number of rejects reaches the MAX_JOB_REJECT value.  When
# ACTION_ON_MAX_REJECT is CANCEL, jobs will be canceled when the
# number of rejects reaches the MAX_JOB_REJECT value. The default
# value is HOLD.
#
ACTION_ON_MAX_REJECT = HOLD
#
# The REJECT_ON_RESTRICTED_LOGIN specifies if the user's AIX login status
# should be checked to determine if the user is locked out.
# REJECT_ON_RESTRICTED_LOGIN may only be specified in the global
# configuration file. Valid values are TRUE or FALSE.  FALSE is the
# default which indicates that no AIX login restriction check is to be
# performed. TRUE specifies that the AIX login restriction check is to be
# performed on every node the job is scheduled to run on. If the user
# has been locked out on any of these nodes then the job will be rejected.
#
REJECT_ON_RESTRICTED_LOGIN = FALSE

# Algorithm a used by the negotiator to determine if a machine has enough
# virtual memory to satisfy the image_size requirement of a job step
#
# can either be FREE_PAGING_SPACE which is the default
# or FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY

VM_IMAGE_ALGORITHM = FREE_PAGING_SPACE

# Filesystem Monitor Interval and Threshholds

# Monitoring Interval is in minutes and should be set according to how
# Fast the filesystem grows
FS_INTERVAL = 30

# Threshholds are specified in low,high pairs.  When the value being
# monitored goes below the low value, the action is taken (eg. the
# administrator is notified of a problem or LoadLeveler is suspended or
# terminated).  When the value being monitored rises above the high
# threshhold after being below the low threshhold, the complementary
# action is taken -- the administrator is notified that the problem
# has been resolved or LoadLeveler is resumed.  LoadLeveler is not
# restarted if it is terminated because the 'Terminate' action
# was taken.

# File System space threshholds are specified in Bytes.  Scaling factors
# such as K, M and G are allowed.
FS_NOTIFY       = 750KB,1MB
FS_SUSPEND      = 500KB, 750KB
FS_TERMINATE    = 100MB,100MB

# File System inode threshholds are specified in number of inodes.  Scaling factors
# such as K, M and G are allowed.
INODE_NOTIFY    = 1K,1.1K
INODE_SUSPEND   = 500,750
INODE_TERMINATE = 50,50
```

Example LoadL_config.local file for conductor node

You must configure the LoadL_config.local file on your conductor node.

The following example shows a LoadL_config.local file for a conductor node.

```
#
# file: LoadL_config.local
# Statements included here override on a PER MACHINE BASIS the statements in
# LoadL_config and may be used to tailor each machine individually.
# See samples in your RELEASEDIR/samples directory
#
START_DAEMONS = TRUE
MAX_STARTERS = 1000000

# Alternative ways of specifying the CLASS keyword
# the following is the old-style specification
#
CLASS = parallel(1000000)
```

Example LoadL_config.local file for compute node

You must configure the LoadL_config.local file on each compute node.

The following example shows a LoadL_config.local file for a compute node.

```
#
# file: LoadL_config.local
# Statements included here override on a PER MACHINE BASIS the statements in
# LoadL_config and may be used to tailor each machine individually.
# See samples in your RELEASEDIR/samples directory
#
START_DAEMONS = TRUE
MAX_STARTERS = 1

# Alternative ways of specifying the CLASS keyword
# the following is the old-style specification
#
CLASS = parallel(1)
```

Example master_config.apt configuration file

You must create a configuration file and store it at $APT_ORCHHOME/etc/master_config.apt.

In the following example, the fields that are specific to your environment (and therefore need customizing) are shown in **bold**.

```
{
        // Conductor node entry
        node "CONDUCTOR"
        {
                fastname "PXRH4vm1.swg.usma.ibm.com"
                pools "conductor"
                resource disk "/opt/IBM/InformationServer/Server/Datasets" {pools ""}
                resource scratchdisk "/opt/IBM/InformationServer/Server/Scratch" {pools ""}
        }
        // Compute node entry
        node "COMPUTE_default"
        {
                fastname "$HOSTNAME"
                pools ""
                resource disk "/opt/IBM/InformationServer/Server/Datasets" {pools ""}
                resource scratchdisk "/opt/IBM/InformationServer/Server/Scratch" {pools ""}
        }
}
```

Example resource_manager file

You must create a resource_manager file and store it at $APT_ORCHHOME/etc/resource_manager.

In the following example, the fields that are specific to your environment (and therefore need customizing) are shown in **bold**.

```
installDir=/opt/ibmll/LoadL/full/bin
computeNodes=3
class=parallel
maxPartitionCount=4

computeNodes represents the number of physical compute machines. maxPartitionCount
represents the number of logical partitions on each compute node. Recommended value
is half of the number of CPUs on each compute node.
```

Getting started with grid project design

When you design grid jobs, you edit the master template using the IBM InfoSphere DataStage Administrator client. The master template defines the static resources that are available in your grid. You can then further define resources for individual projects by generating a project template.

The following list defines terms used in this section:

Master template
: The master template defines the static resources that are available on your grid. There is only one master template for each grid (master_config.apt). Only the administrator can edit the master template.

Project template
: A project template represents a subset of the static grid resources that have been assigned to a particular project. The administrator defines which static resources to use for the project, and then uses the InfoSphere DataStage Administrator client to generate a project template. The project template contents are populated to the Grid job properties page. Immediately before job runtime, the Administrator client generates a job configuration file. For examples and more detail about templates, see "Configuration file templates" on page 620.

Static resources
: Static resources consist of fixed-name servers such as database servers, SAN servers, SAS servers, and remote storage disks.

Dynamic resources
: Dynamic resources consist of compute nodes that are in the default compute node pool and that are assigned dynamically at job runtime. A default compute node entry ($HOSTNAME) is included in the master template to represent a generic compute node. If selected by the administrator for a project, the default compute node entry is included in the project template. After the requested resources are allocated, the engine uses the default compute node entry and any user-defined job requirements to dynamically generate the configuration file.

Grid resources for a job are resolved as described in the following overview procedure. As indicated in the steps, some actions require the administrator-level permission, some require only the user-level permission.

1. The administrator defines static resources in the grid in a master template.
2. The administrator defines which grid resources are enabled for a particular project by selecting which resources to include in the project template.
3. The user defines resource requirements for particular jobs within that project.
4. The user defines resource constraints for particular stages within that job.
5. After the job is invoked but immediately before it runs, the resource manager calls InfoSphere DataStage to dynamically generate a configuration file.

By this method, InfoSphere DataStage can assure that jobs run immediately when enough resources of the correct types are available, instead of waiting for a fixed resource that might be tied up on another long-running job.

Designing grid projects

You can use IBM InfoSphere DataStage clients to design and run grid projects and jobs.

About this task

To design and run a grid-enabled parallel job:

Procedure

1. Open the Administrator client window. From the Windows Start menu, select **Start → Programs → IBM InfoSphere Information Server → IBM InfoSphere DataStage and QualityStage Administrator**.
2. Connect the Administrator client to the InfoSphere Information Server services tier by using the Attach to DataStage window:
3. Edit the master template, if necessary for your grid:

   Note: Only the administrator can alter and save the master template.

   a. Click the **General** tab of the Administrator client to open the General page.

      This General page displays the engine-wide properties for the installation.

   b. On the General page, click **Grid**.

      A view opens where you view and edit the master template. To see an example of a master template and learn more about template format, see "Configuration file templates" on page 620.

   c. Update the master template so that it accurately represents the resources in your grid.

      For example, if you added fixed-name servers that you will use as part of your grid, you must add them to the template.

   d. Click **Check** to validate the modified master template.

   e. When the master template has validated, save any changes that you made and then close the master template.
4. Edit the project template, if necessary for your project:

   Note: Only the administrator can alter and save the project template.

   a. Click the **Project** tab of the Administrator client.

      The Project page opens, showing a list of all your projects.

   b. In the list of projects, select the project that you want to work with, and then click **Properties**.

   c. On the Project Properties window, click the Parallel tab to open the Parallel page.

   d. On the Parallel page, verify that the **Enable grid support for parallel jobs** check box is selected.

   e. On the Parallel page, click the **Grid** button to open the Static Resource Configuration for Grid window.

      The list on this page references the master template but represents a project-level subset of the static resources. All resources from the master template are enabled by default.

   f. Clear the check box in the **Enable** column for any grid resources that you want to exclude from this particular project.

      For example, you might want to exclude a node pool that has the DB2 database Version 8 installed for a project that must run on DB2 Version 9. The **Enable** check boxes and the logical processing partitions for each node pool are the only editable fields in this window. By selecting which node pools to enable for this project, you define the project template. You can click the **View** button to generate and view this project template.

Note: The conductor node pool cannot be disabled. Every configuration file must contain the conductor node.

g. Optional: You can edit the number of logical processing partitions in the **Partition Count** column for each node pool that is listed on this page. The partition count is applied to all fixed-name servers that are assigned to the node pool. The default value is 1.

5. Start the Designer client window. Click **Start → Programs → IBM InfoSphere Information Server → IBM InfoSphere DataStage and QualityStage Designer**.
6. Connect the Designer client to the project that you are working on by using the Attach to DataStage window:
7. Use the Designer client to design jobs within the project.
8. Optional: Define grid resource requirements for each individual job within the project. To do this, open the job or select it in the Repository tree and then select **Edit → Job Properties**.

   Note: Alternatively, you can define grid resource requirements and apply them to all jobs in a sequence, rather than to individual jobs. See "Job operation considerations" on page 626.

   a. On the Job Properties window, click the **Grid** tab to open the Grid page.

      This Grid page references the project template and lists the grid resources that you enabled for the project. Editable fields on this Grid page help you to define resource requirements for the job, as described in the next steps.

   b. In the **Compute (physical) node requirements** area, set the value for the **Number requested** field.

      This is the requested number of compute nodes that you want the resource manager software to wait for before running the job. The default value is 1.

   c. Set the value for the **Minimum number** field.

      This is the minimum number of compute nodes that the resource manager waits for before starting the job, even if the requested number is not available. The default value is blank.

   d. Set the value for the **Partition count** field.

      This is the number of logical processing partitions required per physical compute node. Each logical node is listed as a node in the configuration file. The default value is 1.

   e. Optional: You can edit the logical processing partitions in the **Partition Count** column for each node pool of fixed-name servers that is listed. The partition count is applied to all fixed-name servers that are assigned to the node pool.The default value is 1.

      If you set partition count values on this job-level page, they override any partition count values that were set at the project level. Setting a partition count to zero for a node pool excludes all fixed-name servers that are assigned to that node pool from the dynamically generated configuration file.

9. Optional: Define grid resource constraints for individual stages within a job.

   For example, you can limit execution of a stage to a particular node or node pool. To do this, open the stage or select it in the Repository tree and then select **Edit → Stage**.

   a. On the Stage window, click the **Advanced** tab to open the Advanced page.

      This Advanced page references the job resource requirements and lists the grid resources that you enabled for the job.

   b. Specify node map, node pool, or resource pool constraints for this particular stage.

Configuration file templates

Templates for configuration files help you to define resources for the grid and for individual projects.

Format of templates

The entries in configuration file templates have a standard format.

Each node has its own entry that defines the logical node name, the server host name (fastname), the node pool names, and the storage resources, as shown in this example:

```
node "CONDUCTOR"
{
 fastname "production_conductor_node"
 pools "conductor"
 resource disk "/u1/Datasets" {pools ""}
 resource scratchdisk "/u1/Scratch" {pools ""}
}
```

The following list defines the lines in this example:

node Describes the general function of the logical node. For compute nodes, this logical node name is generated in the configuration file at job runtime, and a logical node name is shown for every processing partition in the physical server. The node names of fixed-name servers are added to the master template by the administrator.

fastname
This is the host name of the server.

pools Specifies the node pool to which the node is assigned. When jobs are exported between different processing environments, the jobs recognize which resources to use if you use common node pool names in the templates for both environments. For example, you define job requirements so that a job uses a "DB2" node pool that contains two fixed-name servers in the development environment; you can deploy this job later to a "DB2" node pool that has 20 fixed-name servers with no need to alter the job.

resource disk
Specifies the storage resource where the node stores persistent data.

resource scratchdisk
Specifies the storage resource where the node stores temporary data.

Master template sample

A sample master template is shown below. This master template is preinstalled on the conductor node of the system.

There is one master template for each IBM InfoSphere DataStage installation. The administrator can manually edit the master template to match actual grid resources. The conductor node entry is preset and must be present for all projects. The administrator can remove commenting marks to add fixed-name servers to the grid or add new entries as required.

Note: The fastname for the compute node, $HOSTNAME, acts as a placeholder for the actual host name that InfoSphere DataStage determines dynamically from the compute resources that are available at job runtime. The compute node entry can be excluded from the template if your job can run on fixed-name servers only.

```
// Information on static grid resources is defined and maintained in this file.
// This file supports C++ style comments. Remove comments and edit node entries
// for fixed-name servers to set up your grid environment.

{
// Conductor node entry
node "CONDUCTOR"
{
  fastname "development_conductor_node"
  pools "conductor"
  resource disk "/u1/Datasets" {pools ""}
  resource disk "/u1/Lookup" {pools "lookup"}
  resource disk "/u1/SurrogateKey" {pools "keystate"}
  resource scratchdisk "/u1/Scratch" {pools ""}
```

```
}
// Compute node entry
node "COMPUTE_default"
{
  fastname "$HOSTNAME"
  pools ""
  resource disk "/u1/Datasets" {pools ""}
  resource disk "/u1/Lookup" {pools "lookup"}
  resource disk "/u1/SurrogateKey" {pools "keystate"}
  resource scratchdisk "/u1/Scratch" {pools ""}
}
// Node entries for fixed-name servers
// node "DB2"
// {
//   fastname "development_db2"
//  pools "DB2"
//  resource scratchdisk "/u1/Scratch" {pools ""}
// }
//node "SAN"
// {
//   fastname "development_io_node"
//   pools "io"
//   resource disk "/u1/Datasets" {pools ""}
// }
//node "SORT"
// {
//   fastname "development_sort_node"
//   pools "sort"
//   resource scratchdisk "/u1/Scratch" {pools ""}
// }
//node "INFORMIX"
// {
//   fastname "development_informix"
//   pools "INFORMIX"
//   resource scratchdisk "/u1/Scratch" {pools ""}
// }
//node "SAS"
// {
//   fastname "development_sas"
//   pools "sas"
//   resource sas "/usr/sas612" { }
//   resource sasworkdisk "/u1/sas/work" { }
//   resource disk "/u1/sas" {pools ""}
//   resource scratchdisk "/u1/Scratch" {pools ""}
// }
}
```

Project template sample

A sample project template is shown below.

The project template references the contents of the master template and defines a subset of the grid resources to use for a specific project.

For example, assume that you use the master template sample. The administrator disables the sort and INFORMIX node pools for a project, but removes commenting to enable the DB2, io, and sas node pools. The resulting project template that is generated appears as shown here:

```
{
// Conductor node entry
node "CONDUCTOR"
{
 fastname "development_conductor_node"
 pools "conductor"
 resource disk "/u1/Datasets" {pools ""}
 resource disk "/u1/Lookup" {pools "lookup"}
```

```
 resource disk "/u1/SurrogateKey" {pools "keystate"}
 resource scratchdisk "/u1/Scratch" {pools ""}
}
// Compute node entry
node "COMPUTE_default"
{
 fastname "$HOSTNAME"
 pools ""
 resource disk "/u1/Datasets" {pools ""}
 resource disk "/u1/Lookup" {pools "lookup"}
 resource disk "/u1/SurrogateKey" {pools "keystate"}
 resource scratchdisk "/u1/Scratch" {pools ""}
}
// Node entries for fixed-name servers
node "DB2"
{
  fastname "development_db2"
  pools "DB2"
  resource scratchdisk "/u1/Scratch" {pools ""}
}
node "SAN"
{
  fastname "development_io_node"
  pools "io"
  resource disk "/u1/Datasets" {pools ""}
}
node "SAS"
{
  fastname "development_sas"
  pools "sas"
  resource sas "/usr/sas612" { }
  resource sasworkdisk "/u1/sas/work" { }
  resource disk "/u1/sas" {pools ""}
  resource scratchdisk "/u1/Scratch" {pools ""}
}
}
```

Configuration file sample

A sample configuration file is shown below.

For example, assume that you use the project template sample. Then, you define the job requirements for this job so that the requested number and minimum number of compute nodes is two and the number of logical partitions per compute node is two. The administrator also sets the partition count value for the fixed-name SAS server (the sas node pool) to zero.

When the job is run, the configuration file that is dynamically generated is shown below. Four logical compute nodes are listed, two for each of the physical servers, production_compute_0 and production_compute_1. The SAS server is excluded from the configuration file because the partition count of the sas node pool was set to zero.

```
{
node "CONDUCTOR"
{
 fastname "development_conductor_node"
 pools "conductor"
 resource disk "/u1/Datasets" {pools ""}
 resource disk "/u1/Lookup" {pools "lookup"}
 resource disk "/u1/SurrogateKey" {pools "keystate"}
 resource scratchdisk "/u1/Scratch" {pools ""}
}
node "DB2"
{
 fastname "development_db2"
 pools "DB2"
```

```
 resource scratchdisk "/u1/Scratch" {pools ""}
}

node "SAN"
{
 fastname "development_io_node"
 pools "io"
 resource disk "/u1/Datasets" {pools ""}
}

node "COMPUTE_0"
{
 fastname "production_compute_0"
 pools ""
 resource disk "/u1/Datasets" {pools ""}
 resource disk "/u1/Lookup" {pools "lookup"}
 resource disk "/u1/SurrogateKey" {pools "keystate"}
 resource scratchdisk "/u1/Scratch" {pools ""}
}

 node "COMPUTE_1"
{
 fastname "production_compute_0"
 pools ""
 resource disk "/u1/Datasets" {pools ""}
 resource disk "/u1/Lookup" {pools "lookup"}
 resource disk "/u1/SurrogateKey" {pools "keystate"}
 resource scratchdisk "/u1/Scratch" {pools ""}
}

 node "COMPUTE_2"
{
 fastname "production_compute_1"
 pools ""
 resource disk "/u1/Datasets" {pools ""}
 resource disk "/u1/Lookup" {pools "lookup"}
 resource disk "/u1/SurrogateKey" {pools "keystate"}
 resource scratchdisk "/u1/Scratch" {pools ""}
}

node "COMPUTE_3"
{
 fastname "production_compute_1"
 pools ""
 resource disk "/u1/Datasets" {pools ""}
 resource disk "/u1/Lookup" {pools "lookup"}
 resource disk "/u1/SurrogateKey" {pools "keystate"}
 resource scratchdisk "/u1/Scratch" {pools ""}
}
}
```

Other grid computing considerations

You must consider other factors when you design and run grid-based jobs.

You must configure SSH access for any IBM InfoSphere DataStage users that you add so that users can launch a command from the conductor node to all compute nodes without a password. Some file types cause constraints on grid computing operations. Job validation and execution also impact your design.

Configuring SSH for passwordless commands

You must configure SSH access for any new users that you add to your grid so that IBM InfoSphere DataStage users can launch a command from the conductor node to all compute nodes without a password. Instead, the user is authenticated only by public key encryption. This procedure must be done for each new user that will run jobs.

Before you begin

In this sample procedure, conductnode is the conductor node and dbnode is a remote node.

To set up SSH for passwordless commands:

Procedure

1. Generate a public and private RSA key pair on the conductor node:
   ```
   ssh-keygen –b 1024 -t rsa -f ~/.ssh/id_rsa
   ```
   The identification keys are saved in the ~/.ssh/id_rsa directory.
2. Press enter twice for a null pass-phrase.

   Note: If user home directories are auto-mounted, the **scp** command in the next step is not necessary. In this case, you still have to log into each remote node at least once before running a parallel engine job using those nodes. The first login generates a prompt that asks whether you want to add the node to the known hosts list. This list cannot be handled by non-interactive programs.
3. Send the public key to the remote node. The **scp** command is secure version of the **rcp** command.
   ```
   cd .ssh
   scp id_rsa.pub user@dbnode:~/.ssh
   ```
4. Log in to the remote node dbnode, add the public key to the list of authorized keys, and set its file permissions if not set already:
   ```
   cd .ssh
   cat id_rsa.pub >>authorized_keys
   chmod 640 authorized_keys
   rm -f id_rsa.pub
   ```

Results

You can now connect from conductnode to dbnode with no password. For example, the following command works from conductnode:

```
ssh dbnode ls
```

File type considerations

The file types in this section are commonly used in IBM InfoSphere DataStage jobs and operations. Specific considerations for using these file types in a grid-enabled environment are described.

Data sets

A data set consists of a descriptor file and one or more data files. The descriptor file for a data set must be stored on the conductor node. The delete operator that removes the descriptor file must run on the conductor node. The data files of a data set must be locally accessible on the servers where those files are stored. The resource disks of all the compute nodes are mounted to common file systems, so data sets are accessible from all the compute nodes.

The resource disks of all the compute nodes are mounted to common file systems, so no error occurs if any of the logical compute node names cannot be found in the generated job configuration file. The parallel engine assigns an available compute node to process the logical file. Optimization is performed to distribute I/O processes across the allocated compute nodes to achieve workload balance.

File sets

A file set consists of one or more logical files, each of which in turn consists of one or more physical files. The descriptor file for a file set must be stored on the conductor node. The delete operator that removes

the descriptor file must run on the conductor node. All physical files of a logical file are stored on the same node, and multiple logical files can coexist on the same node. One entry for each logical file is included in the node map of a file set.

Files

A file path name can take a host name as prefix (for example, `kitt-eth-1:/tmp/example.txt`). If the host name prefix is omitted, then the file is stored on the remote storage system (the default common file system). The host name can be an asterisk, indicating that files with the same name can coexist on all the physical nodes in the configuration file. If the file does not have a prefixed host name, the file I/O is performed on compute nodes to common file systems. Optimization is performed to distribute I/O processes across the allocated compute nodes to achieve workload balance.

File patterns

File patterns are supported by the import operator only in the following format:

hostname:/*directory_name*/example*.dat

If the host name prefix is omitted, then the file pattern is stored on the remote storage system. The host name can be an asterisk, indicating that file patterns with the same name can coexist on all the physical nodes in the configuration file. You can constrain the import operator to run in the file pattern node pool. The framework imports file patterns by using file sets by default.

Explicit node constraints

For debugging and performance tuning purposes, you can explicitly set the environment variable $APT_CONFIG_FILE to a user-created configuration file. No dynamic configuration file is generated in this case. InfoSphere DataStage retrieves the fastname of each compute node from the given configuration file. Then InfoSphere DataStage uses the fastname to request compute resources from the resource manager software.

Setting the $APT_CONFIG_FILE variable to a user-created configuration file is non-optimal, and is not a way of bypassing resource allocation. The job must wait for requested nodes to become available.

Job operation considerations

You must consider the internal grid computing operations when you run jobs.

Job execution

The run mechanism produces a job configuration file that is based on resource requirements and explicit and implicit resource constraints. The job is run from the project home directory, in common with the existing parallel engine infrastructure. The internal workflow of job execution is as follows:

1. The user submits a job run request to IBM InfoSphere DataStage (for example, using the Director client or the dsjob command-line utility).
2. InfoSphere DataStage performs setup of the job that is based upon environment variables.
3. The required resources are requested from the resource manager.
4. The job state is changed to Queued.
5. The resource manager examines the resource request and keeps the job in Queued state until resources are available.
6. The resource manager notifies InfoSphere DataStage when resources are available.
7. InfoSphere DataStage moves the job into the Running state.

8. InfoSphere DataStage creates the job configuration file that is based on the resources that are allocated by the resource manager and any fixed resources in the job design.
9. InfoSphere DataStage performs any before-job subroutines, if they are specified as part of the job design.
10. The parallel job is executed using the nodes that are named in the job configuration file.

Job validation

The same mechanism used to create a configuration file is used to generate a temporary configuration file for job validation.

One difference between job validation and job execution is that the job is not submitted to the resource manager for job validation. Therefore, node resolution for dynamic compute nodes is not considered. However, the program still considers node resolution for static servers, implicit node constraints by file type, and explicit node constraints that are defined by any user-created configuration file.

The temporary configuration file includes only the processing nodes that are associated with compute nodes if the job does not run on static servers. Node resolution for static servers must be done for jobs that require local access to static servers. If a job involves reading from or writing to a data set, file set, file, or file pattern, implicit constraints must also be considered (see "File type considerations" on page 625).

If the user specifies a user-created configuration file for debugging or performance tuning, that configuration file is used as the temporary configuration file. Because no actual processes are created on physical compute nodes, the host name of each compute node is not required. A dummy fastname is used for each processing node. For example:

```
node "COMPUTE_0"
{
 fastname "dummy_hostname"
 pools ""
 resource disk "/u1/Datasets" {pools ""}
 resource disk "/u1/Lookup" {pools "lookup"}
 resource disk "/u1/SurrogateKey" {pools "keystate"}
 resource scratchdisk "/u1/Scratch" {pools ""}
}
```

Job sequence

A job sequence consists of one or more activities that can be executables, routines, scripts, and jobs. You can choose to define computing resource requirements at the sequence level, so that all jobs in the sequence use the same resource requirement definition.

On the **Sequence Job → Edit → Job Properties → Grid** page, select the **Use sequence-level resources for parallel jobs in the sequence** check box to activate the resource requirement fields on the page. The resource requirements that you define are then applied to all jobs that run in the sequence.

In certain situations, jobs in the sequence do not use the same sequence-level resources:

- Any jobs with an explicit fixed sequence that is set by the $APT_CONFIG_FILE environment variable being used as a job parameter.
- Any jobs for which the user has applied the **Force Sequential Mode** property on the Job Properties Execution page.

Resources are allocated to a sequence at the start of the sequence. The resources are then held until the completion of the sequence; they are not released between the individual job runs within the sequencer.

Job batch and job control

Multiple jobs or separate invocations of the same job can be grouped to run in a job batch or as job control at the start of another job run. A batch file is a special case of a server job with job control, but with no stages of its own. The default action is for jobs in job control to run with their own resource settings.

One exception relates to job control from a parent parallel job. If the parent job sets the child's $APT_CONFIG_FILE setting with a job parameter, the child runs with the same configuration file as the parent. Job batches and jobs that are run by using other forms of job control on the Job Properties page generally run with their own resource requirements. If you need coordination of requirements between jobs, use job sequences to define resource requirements at the sequence level.

Deploying jobs in other environments

One powerful feature of IBM InfoSphere DataStage is the ability to design jobs in one processing environment and then to deploy them in another environment with no need for modifying the job design or resource configuration.

For example, you could design a job in a non-grid, symmetric multiprocessing (SMP) development environment, then deploy it to run in a grid-enabled production environment. The persistent information about resource configuration is exported with the job design. Because server names might not be consistent across different environments, they are not stored as part of the persistent resource information during job deployment. Instead, logical node and node pool names are used to interpret and coordinate the resource requirements across different environments.

Requirements for master templates in different environments

There is one master template for each engine installation. The information in the master template defines the processing environment in which jobs are deployed.

Important: If you design jobs in a non-grid environment that you want to run later in a grid environment, you must first enable grid support for parallel jobs in the non-grid environment.

The following list defines requirements for master templates for deploying jobs across different environments:

- Only one conductor node can be in the master template. The logical node name of the conductor node, CONDUCTOR, is reserved. This node is assigned to the "conductor" node pool. Only one node can be in the conductor node pool.
- Only one compute node can be in the master template. Both the logical node name (COMPUTE_default) and the fast name ($HOSTNAME) for the compute node are reserved. This node is assigned to the default node pool that is defined by two empty quotation marks (""). Only one node can be in the default node pool.
- A fixed-name server cannot be in the conductor node pool or the default node pool. The fixed-name server must be in a non default-node pool and can be included in multiple non default-node pools.
- For node pool constraints, node pool names must be the same in master templates across different environments.
- For node map constraints, the logical node name of each node map must be the same in master templates across different environments.

When a job is deployed, the following job resource requirements are saved along with the job design:

- Compute nodes: The requested and minimum number of compute nodes and the logical partition count for each physical node are saved. If the number requested, minimum number, and partition count are specified using job parameter strings, they can take different values at run time. This allows you to change the job resource requirements without having to modify the job design after the job is

deployed. You can design a job with a small number of partitions (in the development environment), and then deploy and run the job with a larger number of partitions (in the test and production environments).

- Fixed-name servers: The logical partition count for each physical server and the node pool name for the server are saved. The server name, logical node name, and storage resources of a fixed-name server are not saved. The node pool name that is saved is used to determine the static resources across different environments.

The user can follow regular job deployment procedures on the grid. For example, the user can import the job, compile it, and then run the job.

The following example shows how this works:

- For example purposes only, various versions of remote DB2 servers are used as the data sources in the following master template code snippets. For InfoSphere Information Server to access remote DB2 data sources, DB2 must already be installed and configured on the remote systems, you must install the DB2 client software on the conductor node, and you must include all fixed-name servers in `master_config.apt`.
- The development environment has two IBM DB2 software installations, one in the DB2V8 node pool, the other in the DB2V9 node pool.
- The testing environment has two IBM DB2 software installations in the DB2V8 node pool and four DB2 installs in the DB2V9 node pool.

The master template for the development environment is shown here:

```
{
  node "CONDUCTOR"
  {
    fastname "development_conductor_node"
    pools "conductor"
    resource disk "/u1/Datasets" {pools ""}
    resource disk "/u1/Lookup" {pools "lookup"}
    resource disk "/u1/SurrogateKey" {pools "keystate"}
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
  node "COMPUTE_default"
  {
    fastname "$HOSTNAME"
    pools ""
    resource disk "/u1/Datasets" {pools ""}
    resource disk "/u1/Lookup" {pools "lookup"}
    resource disk "/u1/SurrogateKey" {pools "keystate"}
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
  node "DB2_v8_0"
  {
    fastname "development_db2_v8_0"
    pools "DB2V8"
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
  node "DB2_v9_0"
  {
    fastname "development_db2_v9_0"
    pools "DB2V9"
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
}
```

The master template for the test environment is shown here:

```
{
  node "CONDUCTOR"
  {
```

```
    fastname "test_conductor_node"
    pools "conductor"
    resource disk "/u1/Datasets" {pools ""}
    resource disk "/u1/Lookup" {pools "lookup"}
    resource disk "/u1/SurrogateKey" {pools "keystate"}
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
  node "COMPUTE_default"
  {
    fastname "$HOSTNAME"
    pools ""
    resource disk "/u1/Datasets" {pools ""}
    resource disk "/u1/Lookup" {pools "lookup"}
    resource disk "/u1/SurrogateKey" {pools "keystate"}
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
  node "DB2_v8_0"
  {
    fastname "test_db2_v8_0"
    pools "DB2V8"
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
  node "DB2_v8_1"
  {
    fastname "test_db2_v8_1"
    pools "DB2V8"
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
  node "DB2_v9_0"
  {
    fastname "test_db2_v9_0"
    pools "DB2V9"
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
  node "DB2_v9_1"
  {
    fastname "test_db2_v9_1"
    pools "DB2V9"
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
  node "DB2_v9_2"
  {
    fastname "test_db2_v9_2"
    pools "DB2V9"
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
  node "DB2_v9_3"
  {
    fastname "test_db2_v9_3"
    pools "DB2V9"
    resource scratchdisk "/u1/Scratch" {pools ""}
  }
}
```

Because the same node pool names for fixed-name servers are used across the two environments, you can design a job with a small number of partitions (in the development environment) and then deploy and run the job with a larger number of partitions (in the test environment). In this example, the job can also run in a production environment that has an even larger number of DB2 partitions with no need to modify the job.

Note that the DB2 logical node names do not have to be the same in both master configuration files. However, if you use the logical node names consistently, it makes it easier to apply node map constraints.

Chapter 15. Remote deployment

Remote deployment of parallel jobs allows job scripts to be stored and run on a separate machine from the engine tier host. The remote deployment option can, for example, be used to run jobs on a computer grid.

Any remote system that has a job so deployed must have access to the Parallel Engine in order to run the job. Such systems must also have the correct runtime libraries for that platform type.

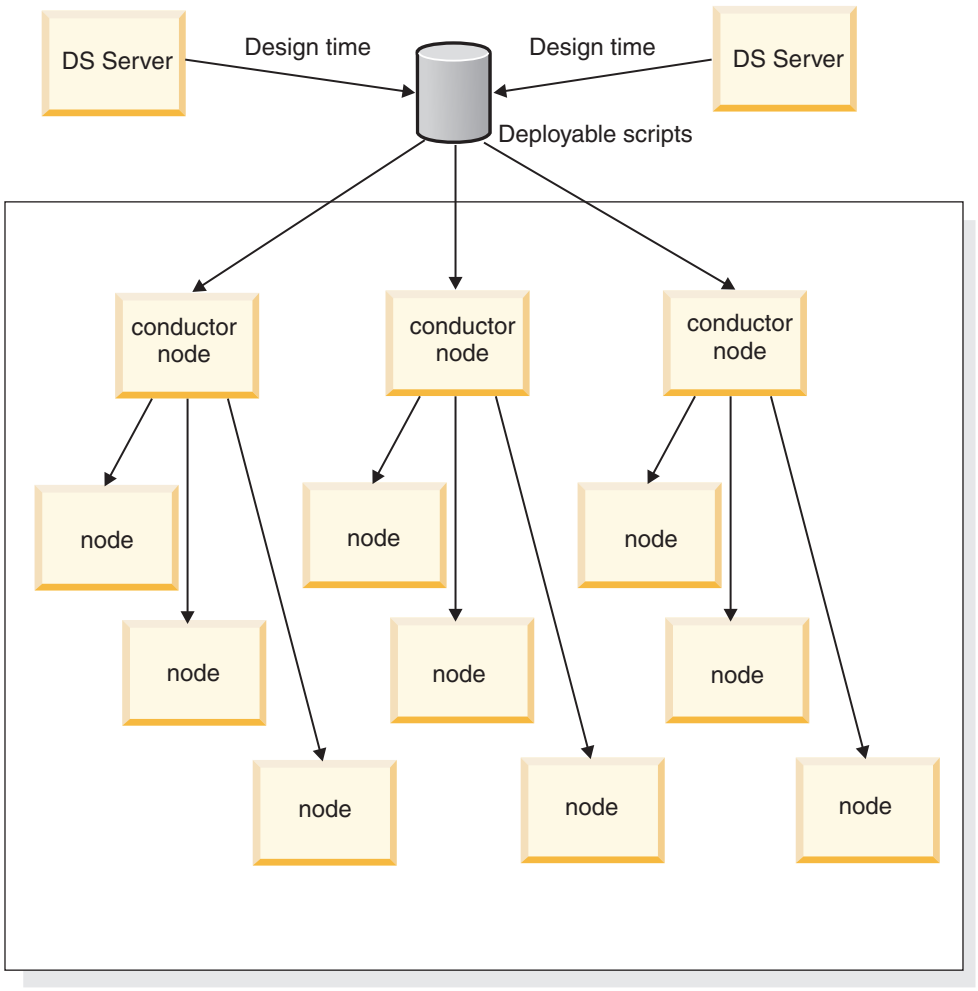
Because these jobs are not run on the InfoSphere DataStage Server, server components (such as BASIC Transformer stages, server shared containers, before and after subroutines, and job control routines) cannot be used. There is also a limited set of plug-in stages available for use in these jobs.

When you run the jobs the logging, monitoring, and operational meta data collection facilities provided by InfoSphere DataStage are not available. Deployed jobs do output logging information in internal parallel engine format, but provision for collecting this is the user's responsibility.

You develop a Parallel job for deployment using the InfoSphere DataStage Designer, and then compile it. A deployment package is automatically produced at compilation. Such jobs can also be run under the control of the engine (using Designer or Director clients, or the dsjob command) as per normal. (Note that running jobs in the `normal' way runs the executables in the project directory, not the deployment scripts.)

It is your responsibility to define a configuration file on the remote machine, transfer the deployment package to the remote computer and to run the job.

The following diagram gives a conceptual view of an example deployment system. In this example, deployable jobs are transferred to three `conductor node' machines. Each conductor node has a configuration file describing the resources that it has available for running the jobs. The jobs then run under the control of that conductor:



Note: The engine tier host system and the Node systems must be running the same operating system.

Enabling a project for job deployment

You can set up a project to make the jobs in the project deployable.

About this task

Projects are made capable of deploying jobs in this way from the InfoSphere DataStage Administrator client.

Procedure

1. Start the InfoSphere DataStage Administrator client.
2. Go to the Projects tab and select the project whose parallel jobs you want to make deployable from the list.
3. Click the Properties button to open the Project Properties dialog box.
4. Go to the Remote tab.
5. In the **Base directory name** field, provide a home directory location for deployment; in this directory there will be one directory for each project. This location has to be accessible from the engine tier host, but does not have to be a disk local to that machine. Providing a location here enables the job deployment features.

6. In the **Deployed job directory template** field, optionally specify an alternative name for the deployment directory associated with a particular job. This field is used in conjunction with **Base directory name** in the form *base_name*/*project_name*/job_directory. By default, if nothing is specified, the name corresponds to the internal script directory used on the engine project directory, RT_SC*jobnum*, where *jobnum* is the internal job number allocated to the job. Substitution strings provided are:
   - %j - jobname
   - %d - internal number

   The simplest case is just "%j" - use the jobname. A prefix can be used, that is, "job_%j". The default corresponds to RT_SC%d.
7. In the **Custom deployment commands** field, optionally specify further actions to be carried out at the end of a deployment compile. You can specify Unix programs and /or calls to user shell scripts as required. The actions take place in the deployment directory for the job.

   This field uses the same substitution strings as the directory template. For example:

Deployment package

When you compile a job in the InfoSphere DataStage Designer with project deployment enabled, the following files are produced:

- Command shell script
- Environment variable setting source script
- Main Parallel (osh) program script
- Script parameter file
- XML report file (if enabled - see *InfoSphere DataStage Parallel Job Advanced Developer Guide*).
- Compiled transformer binary files (if the job contains any Transformer stages)
- Transformer source compilation scripts

These are the files that will be copied to a job directory in the base directory specified in the Administrator client for the project. By default the job directory is called RT_SC*jobnum*, where *jobnum* is the internal job number allocated to the job (you can change the form of the name in the Administrator client).

If you have additional custom components designed outside the job (for example, custom, built, or wrapped stages) you should ensure that these are available when the job is run. They are not automatically packaged and deployed.

Command shell script - pxrun.sh

The command shell script sources the environment variable script, then calls the parallel engine, specifying the main osh program script and script parameter file as parameters. Run this script to run your job.

Environment variable setting source script - evdepfile

This file contains the environment variables for a deployed job when it is run. It is based on the environment variables set when the job was compiled.

It is possible to edit this file manually if required before running a job. The file can be removed altogether, but it is then your responsibility to set up the environment before running the job.

Main parallel (OSH) program script - OshScript.osh

The main parallel job script. You must execute the command shell script in order to run this, you should not run it directly.

Script parameter file - jpdepfile

This is used by *pxrun.sh*. It contains the job parameters for a deployed job when it is run. It is based on the default job parameters when the job was compiled.

It is possible to edit this file manually if required before running a job.

XML report file - <jobname>.xml

An XML report of the job design can be automatically generated at compile time (if enabled using an administration command - see *InfoSphere DataStage Parallel Job Advanced Developer Guide* and the report is included in the job deployment package. For more information on HTML and XML job reports, see *InfoSphere DataStage Designer Client Guide*.

Compiled transformer binary files - <jobnamestagename>.trx.so

There is one of these for each Transformer stage in your job.

Self-contained transformer compilation

In order to make the job self-contained with regard to transformer compilation, there are the following additional files which can optionally be used for transformer recompilation; none are present if there are no transformers in the job:

- Transformer source files (internal transformer language). It has a name in the form *<internalidentifier>_<jobname>_<stagename>.trx.* There is one such file for each Transformer stage in the job.
- Shell scripts to run transformer operator compile jobs. It has a name in the form *<internalidentifier>_<jobname>_<stagename>.trx.sh.* There is one such file for each Transformer stage in the job.
- Transformer compilation operator osh scripts. This is a Parallel job script to compile the corresponding Transformer stage. It is called from corresponding .sh file. It has a name in the form *<internalidentifier>_<jobname>_<stagename>.trx.osh.* There is one such file for each Transformer stage in the job.
- One master shell script to call all transformer compile scripts called *pxcompile.sh*.

If you want to recompile transformers on your deployment platform before running the job, you should run *pxcompile.sh*.

Deploying a job

About this task

This describes how to design a job on the InfoSphere DataStage system in a remote deployment project, transfer it to the deployment machine, and run it.

Procedure

1. In the InfoSphere DataStage Administrator, specify a remote deployment project as described in "Enabling a Project for Job Deployment" .

2. Define a configuration file on your remote deployment systems that will describe it. Use the environment variable APT_CONFIG_FILE to identify it on the remote machine. You can do this in one of three ways:
   - If you are always going to use the same configuration file on the same remote system, define APT_CONFIG_FILE on a project-wide basis in the InfoSphere DataStage Administrator. All your remote deployment job packages will have that value for APT_CONFIG_FILE.
   - To specify the value at individual job level, specify APT_CONFIG_FILE as a job parameter and set the default value to the location of the configuration file. This will be packaged with that particular job.
   - To specify the value at run time, set the value of APT_CONFIG_FILE to $ENV in the InfoSphere DataStage Administrator and then define APT_CONFIG_FILE as an environment variable on your remote machine. The job will pick up the value at run time.
3. In the InfoSphere DataStage Designer, design your parallel job as normal (but remember that you cannot use BASIC Transformer stages, shared containers, or plugin stages in remote deployment jobs).
4. When you are happy with your job design, compile it.
5. If your job contains Transformer stages, you can if required recompile the transformers on the deployment machine. To do this, execute the following file:

   pxcompile.sh
6. When your Transformer stages have successfully compiled, run the job by executing the following file:

   pxrun.sh

Server side plug-ins

InfoSphere DataStage XML and Java plug-ins operate on remote nodes. The following directories are required on the nodes in order to run a plug-in, these can be copied from a InfoSphere DataStage installation:

- *DSEngine/java*
- *DSEngine/lib*
- *DSCAPIOp*

Note: The Java plug-in does not run on Red Hat Enterprise Linux AS 2.1/Red Hat 7.3.

Appendix A. Schemas

Schemas are an alternative way for you to specify column definitions for the data used by parallel jobs. By default, most parallel job stages take their meta data from the Columns tab, which contains table definitions, supplemented, where necessary by format information from the Format tab. For some stages, you can specify a property that causes the stage to take its meta data from the specified schema file instead. Some stages also allow you to specify a partial schema. This allows you to describe only those columns that a particular stage is processing and ignore the rest.

The schema file is a plain text file, this appendix describes its format. A partial schema has the same format.

Note: If you are using a schema file on an NLS system, the schema file needs to be in UTF-8 format. It is, however, easy to convert text files between two different maps with a InfoSphere DataStage job. Such a job would read data from a text file using a Sequential File stage and specifying the appropriate character set on theNLS Map page. It would write the data to another file using a Sequential File stage, specifying the UTF-8 map on the NLS Map page.

Schema format

A schema contains a record (or row) definition. This describes each column (or field) that will be encountered within the record, giving column name and data type. The following is an example record schema:

```
record (
   name:string[255];
   address:nullable string[255];
   value1:int32;
   value2:int32
   date:date)
```

(The line breaks are there for ease of reading, you would omit these if you were defining a partial schema, for example `record(name:string[255];value1:int32;date:date)` is a valid schema.)

The format of each line describing a column is:

column_name:[*nullability*]*datatype*;

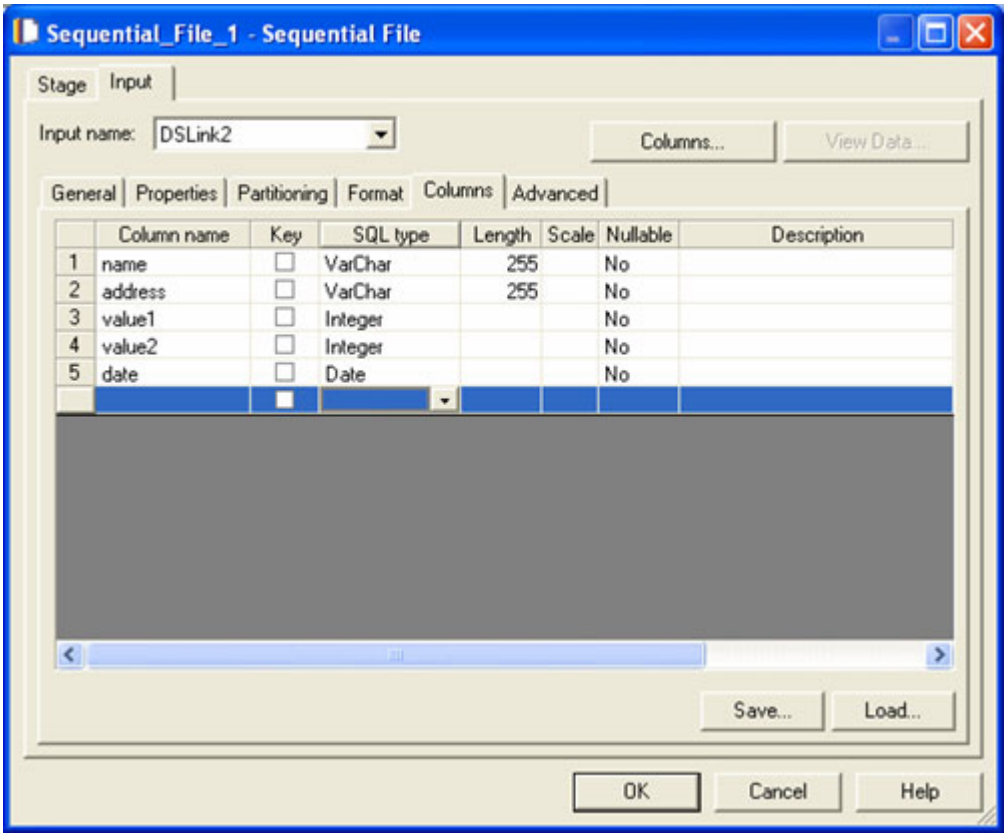
- *column_name*. This is the name that identifies the column. Names must start with a letter or an underscore (_), and can contain only alphanumeric or underscore characters. The name is not case sensitive. The name can be of any length.
- *nullability*. You can optionally specify whether a column is allowed to contain a null value, or whether this would be viewed as invalid. If the column can be null, insert the word 'nullable'. By default columns are not nullable.

  You can also include 'nullable' at record level to specify that all columns are nullable, then override the setting for individual columns by specifying `not nullable'. For example:

  ```
  record nullable (
           name:not nullable string[255];
           value1:int32;
           date:date)
  ```
- *datatype*. This is the data type of the column. This uses the internal data types, see Data Types, not the SQL data types as used on **Columns** tabs in stage editors.

You can include comments in schema definition files. A comment is started by a double slash //, and ended by a newline.

The example schema corresponds to the following table definition as specified on a Columns tab of a stage editor:



The following sections give special consideration for representing various data types in a schema file.

Date columns

The following examples show various different data definitions:

```
record (dateField1:date; ) // single date
record (dateField2[10]:date; ) // 10-element date vector
record (dateField3[]:date; ) // variable-length date vector
record (dateField4:nullable date;) // nullable date
```

(See "Complex Data Types" for information about vectors.)

Decimal columns

To define a record field with data type decimal, you must specify the column's precision, and you might optionally specify its scale, as follows:

column_name:decimal[*precision*, *scale*];

where *precision* is greater than or equal 1 and *scale* is greater than or equal to 0 and less than *precision*.

If the scale is not specified, it defaults to zero, indicating an integer value.

The following examples show different decimal column definitions:

```
record (dField1:decimal[12]; ) // 12-digit integer
record (dField2[10]:decimal[15,3]; )// 10-element
                          //decimal vector
record (dField3:nullable decimal[15,3];) // nullable decimal
```

Floating-point columns

To define floating-point fields, you use the sfloat (single-precision) or dfloat (double-precision) data type, as in the following examples:

```
record (aSingle:sfloat; aDouble:dfloat; ) // float definitions
record (aSingle: nullable sfloat;) // nullable sfloat
record (doubles[5]:dfloat;) // fixed-length vector of dfloats
record (singles[]:sfloat;) // variable-length vector of sfloats
```

Integer columns

To define integer fields, you use an 8-, 16-, 32-, or 64-bit integer data type (signed or unsigned), as shown in the following examples:

```
record (n:int32;) // 32-bit signed integer
record (n:nullable int64;) // nullable, 64-bit signed integer
record (n[10]:int16;)
       // fixed-length vector of 16-bit
                    //signed integer
record (n[]:uint8;) // variable-length vector of 8-bit unsigned
               //int
```

Raw columns

You can define a record field that is a collection of untyped bytes, of fixed or variable length. You give the field data type raw. The definition for a raw field is similar to that of a string field, as shown in the following examples:

```
record (var1:raw[];) // variable-length raw field
record (var2:raw;) // variable-length raw field; same as raw[]
record (var3:raw[40];) // fixed-length raw field
record (var4[5]:raw[40];)// fixed-length vector of raw fields
```

You can specify the maximum number of bytes allowed in the raw field with the optional property max, as shown in the example below:

```
record (var7:raw[max=80];)
```

The length of a fixed-length raw field must be at least 1.

String columns

You can define string fields of fixed or variable length. For variable-length strings, the string length is stored as part of the string as a hidden integer. The storage used to hold the string length is not included in the length of the string.

The following examples show string field definitions:

```
record (var1:string[];) // variable-length string
record (var2:string;) // variable-length string; same as string[]
record (var3:string[80];) // fixed-length string of 80 bytes
record (var4:nullable string[80];) // nullable string
record (var5[10]:string;) // fixed-length vector of strings
record (var6[]:string[80];) // variable-length vector of strings
```

You can specify the maximum length of a string with the optional property max, as shown in the example below:

```
record (var7:string[max=80];)
```

The length of a fixed-length string must be at least 1.

Time columns

By default, the smallest unit of measure for a time value is seconds, but you can instead use microseconds with the [microseconds] option. The following are examples of time field definitions:

```
record (tField1:time; ) // single time field in seconds
record (tField2:time[microseconds];)// time field in //microseconds
record (tField3[]:time; ) // variable-length time vector
record (tField4:nullable time;) // nullable time
```

Timestamp columns

Timestamp fields contain both time and date information. In the time portion, you can use seconds (the default) or microseconds for the smallest unit of measure. For example:

```
record (tsField1:timestamp;)// single timestamp field in //seconds
record (tsField2:timestamp[microseconds];)// timestamp in //microseconds
record (tsField3[15]:timestamp; )// fixed-length timestamp //vector
record (tsField4:nullable timestamp;)// nullable timestamp
```

Vectors

Many of the previous examples show how to define a vector of a particular data type. You define a vector field by following the column name with brackets []. For a variable-length vector, you leave the brackets empty, and for a fixed-length vector you put the number of vector elements in the brackets. For example, to define a variable-length vector of int32, you would use a field definition such as the following one:

```
intVec[]:int32;
```

To define a fixed-length vector of 10 elements of type sfloat, you would use a definition such as:

```
sfloatVec[10]:sfloat;
```

You can define a vector of any data type, including string and raw. You cannot define a vector of a vector or tagged type. You can, however, define a vector of type subrecord, and you can define that subrecord includes a tagged column or a vector.

You can make vector elements nullable, as shown in the following record definition:

```
record (vInt[]:nullable int32;
vDate[6]:nullable date; )
```

In the example above, every element of the variable-length vector vInt will be nullable, as will every element of fixed-length vector vDate. To test whether a vector of nullable elements contains no data, you must check each element for null.

Subrecords

Record schemas let you define nested field definitions, or *subrecord*s, by specifying the type subrec. A subrecord itself does not define any storage; instead, the fields of the subrecord define storage. The fields in a subrecord can be of any data type, including tagged.

The following example defines a record that contains a subrecord:

```
record ( intField:int16;
   aSubrec:subrec (
   aField:int16;
   bField:sfloat; );
   )
```

In this example, the record contains a 16-bit integer field, intField, and a subrecord field, aSubrec. The subrecord includes two fields: a 16-bit integer and a single-precision float.

Subrecord columns of value data types (including string and raw) can be nullable, and subrecord columns of subrec or vector types can have nullable elements. A subrecord itself cannot be nullable.

You can define vectors (fixed-length or variable-length) of subrecords. The following example shows a definition of a fixed-length vector of subrecords:

```
record (aSubrec[10]:subrec (
   aField:int16;
   bField:sfloat; );
)
```

You can also nest subrecords and vectors of subrecords, to any depth of nesting. The following example defines a fixed-length vector of subrecords, aSubrec, that contains a nested variable-length vector of subrecords, cSubrec:

```
record (aSubrec[10]:subrec (
   aField:int16;
   bField:sfloat;
   cSubrec[]:subrec (
      cAField:uint8;
      cBField:dfloat; );
      );
)
```

Subrecords can include tagged aggregate fields, as shown in the following sample definition:

```
record (aSubrec:subrec (
   aField:string;
   bField:int32;
   cField:tagged (
      dField:int16;
      eField:sfloat;
      );
   );
)
```

In this example, aSubrec has a string field, an int32 field, and a tagged aggregate field. The tagged aggregate field cField can have either of two data types, int16 or sfloat.

Tagged columns

You can use schemas to define tagged columns (similar to C unions), with the data type tagged. Defining a record with a tagged type allows each record of a data set to have a different data type for the tagged column. When your application writes to a field in a tagged column, InfoSphere DataStage updates the tag, which identifies it as having the type of the column that is referenced.

The data type of a tagged columns can be of any data type except tagged or subrec. For example, the following record defines a tagged subrecord field:

```
record ( tagField:tagged (
   aField:string;
   bField:int32;
   cField:sfloat;
)   ;
)
```

In the example above, the data type of tagField can be one of following: a variable-length string, an int32, or an sfloat.

Partial schemas

Some parallel job stages allow you to use a partial schema. This means that you only need define column definitions for those columns that you are actually going to operate on. The stages that allow you to do this are file stages that have a Format tab. These are:

- Sequential File stage
- File Set stage
- External Source stage
- External Target stage
- Column Import stage

You specify a partial schema using the Intact property on the Format tab of the stage together with the Schema File property on the corresponding Properties tab. To use this facility, you need to turn Runtime Column Propagation on, and provide enough information about the columns being passed through to enable InfoSphere DataStage to skip over them as necessary.

In the file defining the partial schema, you need to describe the record and the individual columns. Describe the record as follows:

- **intact**. This property specifies that the schema being defined is a partial one. You can optionally specify a name for the intact schema here as well, which you can then reference from the Intact property of the Format tab.
- **record_length**. The length of the record, including record delimiter characters.
- **record_delim_string**. String giving the record delimiter as an ASCII string in single quotes. (For a single character delimiter, use record_delim and supply a single ASCII character in single quotes).

Describe the columns as follows:

- **position**. The position of the starting character within the record.
- **delim**. The column trailing delimiter, can be any of the following:
  - **ws** to 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 ASCII null character.
  - *ASCII_char* specifies a single ASCII delimiter. Enclose ASCII_char in single quotation marks. (To specify multiple ASCII characters, use delim_string followed by the string in single quotes.)
- **text** specifies the data representation type of a field as being text rather than binary. Data is formatted as text by default. (Specify **binary** if data is binary.)

Columns that are being passed through intact only need to be described in enough detail to allow InfoSphere DataStage to skip them and locate the columns that are to be operated on.

For example, say you have a sequential file defining rows comprising six fixed width columns, and you are in interested in the last two. You know that the first four columns together contain 80 characters. Your partial schema definition might appear as follows:

```
record { intact=details, record_delim_string = '\r\n' }
    (    colstoignore: string [80]
         name: string [20] { delim=none };
         income: uint32 {delim = ",", text };
```

Your stage would not be able to alter anything in a row other than the name and income columns (it could also add a new column to either the beginning or the end of a row).

Appendix B. Parallel Transform functions

The parallel transform functions are accessed from the expression editor under the **Function** menu item. You can use these functions when you define a derivation in a Transformer stage. The functions are described by category.

This set includes functions that take string arguments or return string values. If you have national language support enabled, the arguments strings or returned strings can be strings or ustrings. The same function is used for either string type. The only exceptions are the functions StringToUstring () and UstringToString ().

Date and time functions

You can use the date and time functions to perform various operations on dates and times in the Transformer stage.

Functions that specify dates, times, or timestamps in the arguments use strings with specific formats:

- For a date, the format is *%yyyy-%mm-%dd*
- For a time, the format is *%hh:%nn:%ss*. If extended to include microseconds, the format is *%hh:%nn:%ss.x* where *x* gives the number of decimal places seconds is given to.
- For a timestamp, the format is *%yyyy-%mm-%dd %hh:%nn:%ss*. If extended to include microseconds, the format is *%yyyy-%mm-%dd %hh:%nn:%ss.x*, where *x* gives the number of decimal places seconds is given to.

Functions that have days of week in the argument take a string specifying the day of the week. The day is specified as a three-letter abbreviation, or the full name. For example, the strings "thu" and "thursday" are both valid.

The following functions are in the **Date & Time** category of the expression editor. Square brackets indicate an argument is optional. The examples show the function as it appears in a **Derivation** field in the Transformer stage.

CurrentDate

Returns the date that the job runs in date format.

- **Input**: -
- **Output**: date
- **Examples**. Use this function to add a new column containing the date to the data output by the Transformer stage:

```
CurrentDate()
```

CurrentTime

Returns the time at which the job runs in time format.

- **Input**: -
- **Output**: time
- **Examples**. Use this function to add a new column containing the time to the data output by the Transformer stage:

```
CurrentTime()
```

CurrentTimeMS

Returns the time at which the job runs in time format, the time includes microseconds.

- **Input**: -

- **Output**: time
- **Examples**. Use this function to add a new column containing the time to the data output by the Transformer stage. You must set the **Extended** field in the column metadata to **Microseconds** to contain the full time:

  ```
  CurrentTimeMS()
  ```

CurrentTimestamp

Returns a timestamp giving the date and time that the job runs in timestamp format.

- **Input**: -
- **Output**: timestamp
- **Examples**. Use this function to add a new column containing the timestamp to the data output by the Transformer stage:

  ```
  CurrentTimestamp()
  ```

CurrentTimestampMS

Returns a timestamp giving the date and time that the job runs in timestamp format, the time part includes microseconds.

- **Input**: -
- **Output**: timestamp
- **Examples**. Use this function to add a new column containing the timestamp to the data output by the Transformer stage. You must set the **Extended** field in the column metadata to **Microseconds** to contain the full timestamp.

  ```
  CurrentTimestampMS()
  ```

DateFromDaysSince

Returns a date by adding an integer to a baseline date. The integer can be negative to return a date that is earlier than the base date.

- **Input**: number (int32), [baseline_date (date)]
- **Output**: date
- **Examples**. If mylink.myintcol contains the integer 18250, and mylink.mydatecol contains the date 1958–08–18, then the three following functions are equivalent, and return the date 2008–08–05:

  ```
  DateFromDaysSince(18250,"1958-08-18")
  DateFromDaysSince(mylink.myintcol,"1958-08-18")
  DateFromDaysSince(mylink.myintcol,mylink.mydatecol)
  ```

  If mylink.mynegintcol contains the integer -1, and mylink.mydatecol contains the date 1958–08–18, then the following three functions are equivalent, and return the date 1958–08–17:

  ```
  DateFromDaysSince(-1,"1958-08-18")
  DateFromDaysSince(mylink.mynegintcol,"1958-08-18")
  DateFromDaysSince(mylink.mynegintcol,mylink.mydatecol)
  ```

DateFromComponents

Returns a date from the given years, months, and day of month given as three separate values.

- **Input**: years (int32), months (int32), dayofmonth (int32)
- **Output**: date
- **Examples**. If mylink.yearcol contains the value 2010, mylink.monthcol contains the value 12, and mylink.dayofmonthcol contains the value 2, then the two following functions are equivalent, and return the date 2010–12–02.

  ```
  DateFromComponents(2010, 12, 2)
  DateFromComponents(mylink.yearcol, mylink.monthcol, mylink.dayofmonthcol)
  ```

DateFromJulianDay

Returns a date from the given julian day.

- **Input**: julianday (uint32)

- **Output**: date
- **Examples**. If mylink.myjulcol contains the value 2454614, then the two following functions are equivalent, and return the date 2008–05–27.

```
DateFromJulianDay(2454614)
DateFromJulianDay(mylink.myjulcol)
```

DateOffsetByComponents

Returns the given date, with offsets applied from the given year offset, month offset, and day of month offset given as three separate values. The offset values can each be positive, zero, or negative.

- **Input**: basedate (date), year_offset (int32), month_offset (int32), dayofmonth_offset (int32)
- **Output**: date
- **Examples**. If mylink.basedate contains 2011-08-18 and mylink.yearos contains the value 2, mylink.monthos contains the value 0, and mylink.dayofmonthosol contains the value 0, then the two following functions are equivalent, and return the date 2013–08–18.

```
DateOffsetByComponents("2011-08-18", 2011, 8, 18)
DateOffsetByComponents(mylink.basedate, mylink.yearos, mylink.monthos,
mylink.dayofmonthos)
```

- If mylink.basedate contains 2011-08-18 and mylink.yearos contains the value -2, mylink.monthos contains the value 0, and mylink.dayofmonthosol contains the value 0, then the two following functions are equivalent, and return the date 2009–08–18.

```
DateOffsetByComponents("2011-08-18",  2011, 8, 18)
DateOffsetByComponents(mylink.basedate, mylink.yearos, mylink.monthos,
mylink.dayofmonthos)
```

DaysSinceFromDate

Returns the number of days from source date to the given date.

- **Input**: source_date, given_date
- **Output**: days_since (int32)
- **Examples**. If mylink.mysourcedate contains the date 1958–08–18 and mylink.mygivendate contains the date 2008–08–18, then the two following functions are equivalent, and return the integer value 18263.

```
DaysSinceFromDate(mylink.mysourcedate, mylink.mygivendate)
DaysSinceFromDate("1958-08-18","2008-08-18")
```

DaysInMonth

Returns the number of days in the month in the given basedate.

- **Input**: basedate (date)
- **Output**: daysinmonth (int32)
- **Examples**. If mylink.mysourcedate contains the date 1958–08–18, then the two following functions are equivalent, and return the integer value 31.

```
DaysInMonth(mylink.mysourcedate)
DaysInMonth("1958-08-18")
```

DaysInYear

Returns the number of days in the year in the given basedate.

- **Input**: basedate (date)
- **Output**: daysinyearh (int32)
- **Examples**. If mylink.mysourcedate contains the date 2012–08–18, then the two following functions are equivalent, and return the integer value 366.

```
DaysInYear(mylink.mysourcedate)
DaysInYear("2012-08-18")
```

If mylink.mysourcedate contains the date 2011–08–18, then the two following functions are equivalent, and return the integer value 365.

```
DaysInYear(mylink.mysourcedate)
DaysInYear("2011-08-18")
```

DateOffsetByDays

Returns the given date, offset by the given number of days. The offset value can be positive, zero, or negative.

- **Input**: basedate (date), dayoffset (int32)
- **Output**: date
- **Examples**. If mylink.basedate contains 2011-08-18 and mylink.dayoffset contains the value 2, then the two following functions are equivalent, and return the date 2011–08–20.

  ```
  DateOffsetByDays("2011-08-18", 2)
  DateOffsetByDays(mylink.basedate, mylink.dayoffset)
  ```

- If mylink.basedate contains 2011-08-18 and mylink.dayoffset contains the value -31, then the two following functions are equivalent, and return the date 2011–07–18.

  ```
  DateOffsetByDays("2011-08-18", -31)
  DateOffsetByDays(mylink.basedate, mylink.dayoffset)
  ```

HoursFromTime

Returns the hour portion of a time.

- **Input**: time
- **Output**: hours (int8)
- **Examples**. If mylink.mytime contains the time 22:30:00, then the following two functions are equivalent, and return the integer value 22.

  ```
  HoursFromTime(mylink.mytime)
  HoursFromTime("22:30:00")
  ```

JulianDayFromDate

Returns a julian day from the given date.

- **Input**: date
- **Output**: julianday (int32)
- **Examples**. If mylink.mydate contains the date 2008–05–27, then the two following functions are equivalent, and return the value 2454614.

  ```
  JulianDayFromDate("2008–05–27")
  JulianDayFromDate(mylink.mydate)
  ```

MicroSecondsFromTime

Returns the microsecond portion of a time.

- **Input**: time
- **Output**: microseconds (int32)
- **Examples**. If mylink.mytime contains the time 22:30:00.32, then the following function returns the value 320000:

  ```
  MicroSecondsFromTime(mylink.mytime)
  ```

MidnightSecondsFromTime

Returns the number of seconds from midnight to the given time.

- **Input**: time
- **Output**: seconds (int8)
- **Examples**. If mylink.mytime contains the time 00:30:52, then the two following functions are equivalent, and return the value 1852:

  ```
  MidnightSecondsFromTime("00:30:52")
  MidnightSecondsFromTime(mylink.mytime)
  ```

MinutesFromTime

Returns the minute portion of a time.

- **Input**: time

- **Output**: minutes (int8)
- **Examples**. If mylink.mytime contains the time 22:30:52, then the two following functions are equivalent, and return the value 30:

```
MinutesFromTime("22:30:52")
MinutesFromTime(mylink.mytime)
```

MonthDayFromDate

Returns the day of the month from the given date.

- **Input**: date
- **Output**: day (int8)
- **Examples**. If mylink.mydate contains the date 2008-08-18, then the two following functions are equivalent, and return the value 18:

```
MonthDayFromDate("2008-08-18")
MonthDayFromDate(mylink.mydate)
```

MonthFromDate

Returns the month number from the given date.

- **Input**: date
- **Output**: month_number (int8)
- **Examples**. If mylink.mydate contains the date 2008-08-18, then the two following functions are equivalent, and return the value 8:

```
MonthFromDate("2008-08-18")
MonthDayDate(mylink.mydate)
```

NextWeekdayFromDate

Returns the date of the specified day of the week soonest after the source date. The day of the week is specified as the full name, for example, thursday, or a three-letter abbreviation, for example, thu.

- **Input**: sourcedate (date), day_of_week (string)
- **Output**: date
- **Examples**. If mylink.mysourcedate contains the date 2008-08-18, then the two following functions are equivalent, and return the value 2008–08–21:

```
NextWeekdayFromDate("2008-08-18", "thursday")
NextWeekdayFromDate(mylink.mysourcedate, "thu")
```

NthWeekdayFromDate

Returns the date of the specified day of the week offset by the specified number of weeks from the source date. The day of the week is specified as the full name, for example, thursday, or a three-letter abbreviation, for example, thu. The offset can be positive, negative, or zero.

- **Input**: basedate (date), day_of_week (string), week_offset (int32)
- **Output**: date
- **Examples**. If mylink.mydate contains the date 2009-08-18, then the two following functions are equivalent, and return the value 2009–08–27:

```
NthWeekdayFromDate("2009-08-18", "thursday", 1)
NthWeekdayFromDate(mylink.mydate, "thu", 1)
```

If mylink.mydate contains the date 2009-08-18, then the two following functions are equivalent, and return the value 2009–08–06:

```
NthWeekdayFromDate("2009-08-18", "thursday", -2)
NthWeekdayFromDate(mylink.mydate, "thu", -2)
```

PreviousWeekdayFromDate

Returns the date of the specified day of the week most recent before the source date. The day of the week is specified as the full name, for example, thursday, or a three-letter abbreviation, for example, thu.

- **Input**: sourcedate, day_of_week (string)
- **Output**: date
- **Examples**. If mylink.mysourcedate contains the date 2008-08-18, then the two following functions are equivalent, and return the value 2008–08–14:

```
PreviousWeekdayFromDate("2008-08-18", "thursday")
PreviousWeekdayFromDate(mylink.mysourcedate, "thu")
```

SecondsFromTime

Returns the seconds portion of a time.

- **Input**: time
- **Output**: seconds (dfloat)
- **Examples**. If mylink.mytime contains the time 22:30:52, then the two following functions are equivalent, and return the value 52:

```
SecondsFromTime("22:30:52")
SecondsFromTime(mylink.mytime)
```

SecondsSinceFromTimestamp

Returns the number of seconds between two timestamps.

- **Input**: timestamp, timestamp_base
- **Output**: seconds (dfloat)
- **Examples**. If mylink.mytimestamp contains the timestamp 2008–08–18 22:30:52, and mylink.mytimestamp_base contains the timestamp 2008–08–19 22:30:52, then the two following functions are equivalent, and return the value -86400:

```
SecondsSinceFromTimestamp("2008–08–18 22:30:52","2008–08–19 22:30:52")
SecondsSinceFromTimestamp(mylink.mytimestamp, mylink.mytimestamp_base)
```

TimeDate

Returns the system time and date as a formatted string.

- **Input**: -
- **Output**: system time and date (string)
- **Examples**. If the job was run at 4.21 pm on June 20th 2008, then the following function returns the string “16:21:48 20 Jun 2008”.

```
TimeDate()
```

TimeFromComponents

Returns a time from the given hours, minutes, seconds and microseconds given as four separate values.

- **Input**: hours (int32), minutes (int32), seconds (int32), microseconds (int32)
- **Output**: time
- **Examples**. If mylink.hourcol contains the value 10, mylink.mincol contains the value 12, mylink.seccol contains the value 2, and mylink.mseccol contains 0, then the two following functions are equivalent, and return the time 10:12:02.0:

```
TimeFromComponents(10, 12, 2, 0)
TimeFromComponents(mylink.hourcol, mylink.mincol, mylink.seccol, mylink.mseccol)
```

TimeFromMidnightSeconds

Returns the time given the number of seconds since midnight.

- **Input**: seconds (dfloat)
- **Output**: time
- **Examples**. If mylink.mymidnightseconds contains the value 240, then the two following functions are equivalent, and return the value 00:04:00:

```
TimeFromMidnightSeconds("240")
TimeFromMidnightSeconds(mylink.mymidnightseconds)
```

TimeOffsetByComponents

Returns the given time, with offsets applied from the given hour offset, minute offset, and second offset, each given as separate values. The seconds offset can include partial seconds.

- **Input**: basetime (time), hour_offset (int32), minute_offset (int32), second_offset (dfloat)
- **Output**: time
- **Examples**. If mylink.basetime contains 14:05:29 and mylink.houros contains the value 2, mylink.minos contains the value 0, mylink.secos contains the value 20, then the two following functions are equivalent, and return the time 16:05:49.

```
TimeOffsetByComponents("14:05:29", 2, 0, 20)
TimeOffsetByComponents(mylink.basetime, mylink.houros, mylink.minos, mylink.secos)
```

TimeOffsetBySeconds

Returns the given time, with offsets applied from the given seconds offset. The seconds offset can include partial seconds.

- **Input**: basetime (time), second_offset (dfloat)
- **Output**: time
- **Examples**. If mylink.basetime contains 14:05:29.30 and mylink.secos contains the value 2.5, then the two following functions are equivalent, and return the time 14:05:31.80:

```
TimeOffsetByComponents("14:05:29.30", 2.5)
TimeOffsetByComponents(mylink.basetime, mylink.secos)
```

TimestampFromDateTime

Returns a timestamp form the given date and time.

- **Input**: date time
- **Output**: timestamp
- **Examples**. If mylink.mydate contains the date 2008–08–18 and mylink.mytime contains the time 22:30:52, then the two following functions are equivalent, and return the timestamp 2008–08–18 22:30:52:

```
TimestampFromDateTime("2008–08–18","22:30:52")
TimestampFromDateTime(mylink.mydate,mylink.mytime)
```

TimestampFromSecondsSince

Returns a timestamp derived from the number of seconds from the base timestamp.

- **Input**: seconds (dfloat), [base_timestamp]
- **Output**: timestamp
- **Examples**. If mylink.myseconds contains the value 2563 and mylink.timestamp_base contains the timestamp 2008–08–18 22:30:52, then the two following functions are equivalent, and return the timestamp 2008–08–18 23:13:35:

```
TimestampFromSecondsSince("2563","2008–08–18 22:30:52")
TimestampFromSecondsSince(mylink.myseconds,mylink.timestamp_base)
```

TimestampFromTimet

Returns a timestamp from the given UNIX time_t value.

- **Input**: timet (int32)
- **Output**: timestamp
- **Examples**. If mylink.mytimet contains the value 1234567890, then the two following functions are equivalent, and return the timestamp 2009–02–13 23:31:30:

```
TimestampFromTimet("1234567890")
TimestampFromTimet(mylink.mytimet)
```

TimestampOffsetByComponents

Returns the given timestamp, with offsets applied from the given year offset, month offset, day offset, hour offset, minute offset, and second offset, each given as separate values. The seconds offset can include partial seconds.

- **Input**: basetimestamp (timestamp), year_offset (int32), month_offset (int32), dayofmonth_offset (int32), hour_offset (int32), minute_offset (int32), second_offset (dfloat)
- **Output**: timestamp
- **Examples**. If mylink.basetimestamp contains 2009-08-18 14:05:29 and mylink.yearos contains 0, mylink.monthos contains the value 2, mylink.dayos contains the value -4, mylink.houros contains the value 2, mylink.minos contains the value 0, mylink.secos contains the value 20, then the two following functions are equivalent, and return the timestamp 2009-10-14 16:05:49.

```
TimestampOffsetByComponents("2009-08-18 14:05:29", 0, 2, -4, 2, 0, 20)
TimestampOffsetByComponents(mylink.basetimestamp, mylink.houros,
mylink.minos, mylink.secos)
```

TimestampOffsetBySeconds

Returns the given timestamp, with offsets applied from the given seconds offset. The seconds offset can include partial seconds.

- **Input**: basetimestamp (timestamp), second_offset (dfloat)
- **Output**: timestamp
- **Examples**. If mylink.basetimestamp contains 2009-08-18 14:05:29 and mylink.secos contains the value 32760, then the two following functions are equivalent, and return the timestamp 2009-08-18 23:11:29:

```
TimeOffsetBySeconds("2009-08-18 14:05:29", 32760)
TimeOffsetBySeconds
(mylink.basetimestamp, mylink.secos)
```

TimetFromTimestamp

Returns a UNIX time_t value from the given timestamp.

- **Input**: timestamp
- **Output**: timet (int32)
- **Examples**. If mylink.mytimestamp contains the value 2009–02–13 23:31:30, then the two following functions are equivalent, and return the value 1234567890:

```
TimestampFromTimet("2009–02–13 23:31:30")
TimestampFromTimet(mylink.mytimestamp)
```

WeekdayFromDate

Returns the day number of the week from the given date. Origin_day optionally specifies the day regarded as the first in the week and is Sunday by default.

- **Input**: date, [origin_day]
- **Output**: day (int8)
- **Examples**. If mylink.mydate contains the date 2008-08-18, then the two following functions are equivalent, and return the value 1:

```
WeekdayFromDate("2008-08-18")
WeekdayFromDate(mylink.mydate)
```

If mylink.mydate contains the date 2008-08-18, and mylink.origin_day contains saturday, then the two following functions are equivalent, and return the value 2:

```
WeekdayFromDate("2008-08-18","saturday")
WeekdayFromDate(mylink.mydate,mylink.origin_day)
```

YeardayFromDate

Returns the day number in the year from the given date.

- **Input**: date
- **Output**: day (int16)
- **Examples**. If mylink.mydate contains the date 2008-08-18, then the two following functions are equivalent, and return the value 231:

```
YeardayFromDate("2008-08-18")
YeardayFromDate(mylink.mydate)
```

YearFromDate

Returns the year from the given date.

- **Input**: date
- **Output**: year (int16)
- **Examples**. If mylink.mydate contains the date 2008-08-18, then the two following functions are equivalent, and return the value 2008:

```
YearFromDate("2008-08-18")
YearFromDate(mylink.mydate)
```

YearweekFromDate

Returns the week number in the year from the given date

- **Input**: date
- **Output**: week (int16)
- **Examples**. If mylink.mydate contains the date 2008-08-18, then the two following functions are equivalent, and return the value 33:

```
YearweekFromDate("2008-08-18")
YearweekFromDate(mylink.mydate)
```

Logical functions

The logical functions perform bit operations.

The logical functions are in the **Logical** category of the expression editor. Square brackets indicate an argument is optional. The examples show the function as it appears in a **Derivation** field in the Transformer stage.

BitAnd

Returns the bitwise AND of the two integer arguments.

- **Input**: number1 (uint64), number2 (uint64)
- **Output**: number (uint64)
- **Examples**. If mylink.mynumber1 contains the number 352 and mylink.mynumber2 contains the number 400, then the following two functions are equivalent, and return the value 256:

```
BitAnd(352,400)
BitAnd(mylink.mynumber1,mylink.mynumber2)
```

BitCompress

Returns the integer made from the string argument, which contains a binary representation of "1"s and "0"s.

- **Input**: string
- **Output**: number (uint64)
- **Examples**. If mylink.mynumber1 contains the string "0101100000", then the following two functions are equivalent, and return the number 352.

```
BitExpand("0101100000")
BitExpand(mylink.mynumber)
```

BitExpand

Returns a string containing the binary representation in "1"s and "0"s of the given integer.

- **Input**: number (uint64)
- **Output**: string
- **Examples**. If mylink.mynumber1 contains the number 352, then the following two functions are equivalent, and return the string "0101100000".

```
BitExpand(352)
BitExpand(mylink.mynumber)
```

BitOr Returns the bitwise OR of the two integer arguments.

- **Input**: number1 (uint64), number2 (uint64)
- **Output**: number (uint64)
- **Examples**. If mylink.mynumber1 contains the number 352 and mylink.mynumber2 contains the number 400, then the following two functions are equivalent, and return the value 496:

```
BitOr(352,400)
BitOr(mylink.mynumber1,mylink.mynumber2)
```

BitXOr

Returns the bitwise Exclusive OR of the two integer arguments.

- **Input**: number1 (uint64), number2 (uint64)
- **Output**: number (uint64)
- **Examples**. If mylink.mynumber1 contains the number 352 and mylink.mynumber2 contains the number 400, then the following two functions are equivalent, and return the value 240:

```
BitXOr(352,400)
BitXOr(mylink.mynumber1,mylink.mynumber2)
```

Not Returns the complement of the logical value of an expression. If the value of expression is true, the Not function returns a value of false (0). If the value of expression is false, the NOT function returns a value of true (1). A numeric expression that evaluates to 0 is a logical value of false. A numeric expression that evaluates to anything else, other than the null value, is a logical true. An empty string is logically false. All other string expressions, including strings that include an empty string, spaces, or the number 0 and spaces, are logically true.

- **Input**: expression
- **Output**: complement (int8)
- **Examples**. If mylink.myexpression contains the expression 5–5, then the following two functions are equivalent, and return the value 1:

```
Not(5-5)
Not(mylink.myexpression)
```

If mylink.myexpression contains the expression 5+5, then the following two functions are equivalent, and return the value 0:

-
```
Not(5+5)
Not(mylink.myexpression)
```

SetBit Returns an integer with specific bits set to a specific state, where *origfield* is the input value to perform the action on, *bitlist* is a string containing a list of comma-separated bit numbers to set the state of, and *bitstate* is either 1 or 0, indicating which state to set those bits.

- **Input**: origfield (uint64),bitlist (string),bitstate (uint8)
- **Output**: number (uint64)
- **Examples**. If mylink.origfield contains the number 352, mylink.bitlist contains the list "2,4,8", and mylink.bitstate contains the value 1, then the following two functions are equivalent, and return the value 494:

```
SetBit(356,"2,4,8",1)
SetBit(mylink.origfield,mylink.bitlist,mylink.bitstate)
```

Mathematical functions

The mathematical functions perform mathematical operations.

The mathematical functions are in the **Mathematical** category of the expression editor. Square brackets indicate an argument is optional. The examples show the function as it appears in a **Derivation** field in the Transformer stage.

Abs Returns the absolute value of any numeric expression. The absolute value of an expression is its unsigned magnitude.

- **Input**: numeric_expression (int32)
- **Output**: result (dfloat)
- **Examples**. If mylink.number1 contains the number 12 and mylink.number2 contains the number 34, then the following two functions are equivalent, and return the number 22:

```
Abs(12-34)
Abs(mylink.mynumber1-mylink.mynumber2)
```

  If mylink.number1 contains the number 34 and mylink.number2 contains the number 12, then the following two functions are equivalent, and return the number 22:

```
Abs(34-12)
Abs(mylink.mynumber1-mylink.mynumber2)
```

Acos Calculates the trigonometric arc-cosine of an expression. The expression must be a numeric value. The result is expressed in radians.

- **Input**: numeric_expression (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 0.707106781, then the following two functions are equivalent, and return the value 0.785398:

```
Acos(0.707106781)
Acos(mylink.mynumber)
```

Asin Calculates the trigonometric arc-sine of an expression. The expression must be a numeric value. The result is expressed in radians.

- **Input**: numeric_expression (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 0.707106781, then the following two functions are equivalent, and return the value 0.785398:

```
Asin(0.707106781)
Asin(mylink.mynumber)
```

Atan Calculates the trigonometric arc-tangent of an expression. The expression must be a numeric value. The result is expressed in radians.

- **Input**: numeric_expression (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 135, then the following two functions are equivalent, and return the value 1.56339, which is the angle that has an arc tangent of 135:

```
Atan(135)
Atan(mylink.mynumber)
```

Ceil Calculates the smallest integer value greater than or equal to the given decimal value.

- **Input**: number (dfloat)
- **Output**: result (int32)
- **Examples**. If mylink.number contains the number 2355.66, then the following two functions are equivalent, and return the value 2356:

```
Ceil(2355.66)
Ceil(mylink.mynumber)
```

Cos Calculates the trigonometric cosine of an expression. The expression must be a numeric value. The expression must produce a numeric value which is the angle in radians.

- **Input**: radians (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 0.785398, then the following two functions are equivalent, and return the value 0.7071:

```
Cos(0.785398)
Cos(mylink.mynumber)
```

Cosh Calculates the hyperbolic cosine of an expression. The expression must be a numeric value.

- **Input**: number (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 2, then the following two functions are equivalent, and return the value 3.7622:

```
Cosh(2)
Cosh(mylink.mynumber)
```

Div Outputs the whole part of the real division of two real numbers (dividend, divisor).

- **Input**: dividend (dfloat), divisor (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.dividend contains the number 100, and mylink.divisor contains the number 25, then the following two functions are equivalent, and return the value 4:

```
Div(100,25)
Div(mylink.dividend,mylink.divisor)
```

Exp Calculates the result of base 'e' raised to the power designated by the value of the expression. The value of 'e' is approximately 2.71828. The expression must evaluate to a numeric value.

- **Input**: number (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 5, then the following two functions are equivalent, and return the value 54.5982:

```
Exp(5-1)
Exp(mylink.number-1)
```

Fabs Calculates the absolute value of the given float value.

- **Input**: number (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number -26.53, then the following two functions are equivalent, and return the value 26.53:

```
Fabs(-26.53)
Fabs(mylink.number)
```

Floor Calculates the largest integer value less than or equal to the given decimal value.

- **Input**: number (dfloat)
- **Output**: result (int32)
- **Examples**. If mylink.number contains the number 203.25, then the following two functions are equivalent, and return the value 203:

```
Floor(203.25)
Floor(mylink.number)
```

Ldexp Returns a dfloat value from multiplying the mantissa by 2 raised to the power of the exponent.

- **Input**: mantissa (dfloat), exponent (int32)
- **Output**: result (dfloat)
- **Examples**. If mylink.mantissa contains the number 2, and mylink.exponent contains the number 3, then the following two functions are equivalent, and return the value 16:

```
Floor(2,3)
Floor(mylink.mantissa,mylink.exponent)
```

Llabs Calculates the absolute value of the given integer value.

- **Input**: number (integer)
- **Output**: result (unsigned integer)

- **Examples**. If mylink.number contains the number -26, then the following two functions are equivalent, and return the value 26:

```
Llabs(-26)
Llabs(mylink.number)
```

Ln Calculates the natural logarithm of an expression in base 'e'. The value of 'e' is approximately 2.71828. The expression must evaluate to a numeric value greater than 0.

- **Input**: number (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 6, then the following two functions are equivalent, and return the value 1.79176:

```
Ln(6)
Ln(mylink.number)
```

Log10 Returns the log to the base 10 of the given value

- **Input**: number (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 6, then the following two functions are equivalent, and return the value 0.778151:

```
Log10(6)
Log10(mylink.number)
```

Max Returns the greater of the two argument values.

- **Input**: number1 (int32),number2(int32)
- **Output**: result (int32)
- **Examples**. If mylink.number1 contains the number 6, and mylink.number1 contains the number 101, then the following two functions are equivalent, and return the value 101:

```
Max(6,101)
Max(mylink.number1,mylink.number2)
```

Min Returns the lower of the two argument values.

- **Input**: number1 (int32),number2(int32)
- **Output**: result (int32)
- **Examples**. If mylink.number1 contains the number 6, and mylink.number1 contains the number 101, then the following two functions are equivalent, and return the value 6:

```
Min(6,101)
Min(mylink.number1,mylink.number2)
```

Mod Calculates the modulo (the remainder) of two expressions (dividend, divisor).

- **Input**: dividend (int32),divisor (int32)
- **Output**: result (int32)
- **Examples**. If mylink.dividend contains the number 115, and mylink.divisor contains the number 25, then the following two functions are equivalent, and return the value 15:

```
Mod(115,25)
Mod(mylink.dividend,mylink.divisor)
```

Neg Negates a number.

- **Input**: number (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 123, then the following two functions are equivalent, and return the value -123:

```
Neg(123)
Neg(mylink.number)
```

Pwr Calculates the value of an expression when raised to a specified power (expression, power).

- **Input**: expression (dfloat),power (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.expression contains the number 2, and mylink.power contains the number 3, then the following two functions are equivalent, and return the value 8:

```
Pwr(2,3)
Pwr(mylink.expression,mylink.power)
```

Rand Return a psuedo random integer between 0 and 2^{32}-1

- **Input**: -
- **Output**: result (uint32)
- **Examples**. Use this function to add a column to your output containing a random number:

```
Rand()
```

Random

Returns a random number between 0 and 2^{32}-1

- **Input**: -
- **Output**: result (uint32)
- **Examples**. Use this function to add a column to your output containing a random number:

```
Random()
```

Sin Calculates the trigonometric sine of an expression. The expression must be a numeric value. The expression must produce a numeric value which is the angle in radians.

- **Input**: radians (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 0.785398, then the following two functions are equivalent, and return the value 0.7071:

```
Sin(0.785398)
Sin(mylink.mynumber)
```

Sinh Calculates the hyperbolic sine of an expression. The expression must be a numeric value.

- **Input**: number (dfloat)
- **Output**: result (dfloat)
- **Examples**: If mylink.number contains the number 2, then the following two functions are equivalent, and return the value 3.62686:

```
Sinh(2)
Sinh(mylink.mynumber)
```

Sqrt Calculates the square root of a number.

- **Input**: number (dfloat)
- **Output**: result (dfloat)
- **Examples**: If mylink.number contains the number 450, then the following two functions are equivalent, and return the value 21.2132:

```
Sqrt(450)
Sqrt(mylink.mynumber)
```

Tan Calculates the trigonometric tangent of an expression. The expression must produce a numeric value which is the angle in radians.

- **Input**: radians (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 0.7853981, then the following two functions are equivalent, and return the value 0.7071:

```
Tan(0.7853981)
Tan(mylink.mynumber)
```

Tanh Calculates the hyperbolic tangent of an expression. The expression must be a numeric value.

- **Input**: number (dfloat)
- **Output**: result (dfloat)
- **Examples**: If mylink.number contains the number 2, then the following two functions are equivalent, and return the value 0.964028:

```
Tanh(2)
Tanh(mylink.mynumber)
```

Null handling functions

You can use the null handling functions in the Transformer stage to handle nulls in derivations.

If you use input columns in an output column expression, a null value in any input column causes a null to be written to the output column. You can, however, use the null handling functions to handle nulls explicitly.

The following functions are available in the **Null Handling** category. Square brackets indicate an argument is optional. The examples show the function as it appears in a **Derivation** field in the Transformer stage.

IsNotNull
Returns true when an expression does not evaluate to the null value.

- **Input**: any
- **Output**: true/false (int8)
- **Examples**. If the Derivation field for an output column contained the following code, then the Transformer stage checks if the input column named mylink.mycolumn contains a null value. If the input column does not contain a null, the output column contains the value of the input column. If the input column does contain a null, then the output column contains the string NULL.

```
If IsNotNull(mylink.mycolumn) Then mylink.mycolumn Else "NULL"
```

IsNull Returns true when an expression evaluates to the null value.

- **Input**: any
- **Output**: true/false (int8)
- **Examples**. If the Derivation field for an output column contained the following code, then the Transformer stage checks if the input column named mylink.mycolumn contains a null value. If the input column contains a null, the output column contains the string NULL. If the input column does not contain a null, then the output column contains the value of the input column.

```
If IsNull(mylink.mycolumn) Then "NULL" Else mylink.mycolumn
```

NullToEmpty
Returns an empty string if the input column is null, otherwise returns the input column value.

- **Input**: input column
- **Output**: input column value or empty string
- **Examples**. If the Derivation field for an output column contained the following code, then the Transformer stage checks if the input column named mylink.mycolumn contains a null value. If the input column contains a null, the output column contains an empty string. If the input column does contain a null, then the output column contains the value from the input column.

```
NullToEmpty(mylink.mycolumn)
```

NullToZero
Returns zero if the input column is null, otherwise returns the input column value.

- **Input**: input column

- **Output**: input column value or zero
- **Examples**. If the Derivation field for an output column contained the following code, then the Transformer stage checks if the input column named mylink.mycolumn contains a null value. If the input column contains a null, the output column contains zero. If the input column does contain a null, then the output column contains the value from the input column.

  ```
  NullToZeroy(mylink.mycolumn)
  ```

NullToValue

Returns the specified value if the input column is null, otherwise returns the input column value.

- **Input**: input column, *value*
- **Output**: input column value or *value*
- **Examples**. If the Derivation field for an output column contained the following code, then the Transformer stage checks if the input column named mylink.mycolumn contains a null value. If the input column contains a null, the output column contains 42. If the input column does contain a null, then the output column contains the value from the input column.

  ```
  NullToValue(mylink.mycolumn,42)
  ```

SetNull

Assigns a null value to the target column.

- **Input**: -
- **Output**: -
- **Examples**. If the Derivation field for an output column contained the following code, then the Transformer stage sets the output column to null:

  ```
  setnull()
  ```

Number functions

Use the number functions to extract the mantissa from a decimal or floating point number. The **Number** category in the expression editor also contains the type casting functions, which you can use to cast numbers as double, float, or integer data types.

Square brackets indicate an argument is optional. The examples show the function as it appears in a **Derivation** field in the Transformer stage.

The type casting functions help you when you perform mathematical calculations using numeric fields. For example, if you have a calculation using an output column of type float derived from an input column of type integer in a Parallel Transformer stage the result is derived as an integer regardless of its float type. If you want a non-integral result for a calculation using integral operands, you can use the type casting functions to cast the integer operands into non-integral operands.

AsDouble

Treat the given number as a double.

- **Input**: number
- **Output**: number (double)
- **Examples**. In the following expression, the input column mynumber contains an integer, but the function outputs a double. If mylink.mynumber contains the value 56, then the following two functions are equivalent, and return the value 1.29629629629629E+01:

  ```
  AsDouble(56/4.32)
  AsDouble(mylink.mynumber/4.32)
  ```

AsFloat

Treat the given number as a float.

- **Input**: number
- **Output**: number (float)

- **Examples**. In the following expression, the input column mynumber contains an integer, but the function outputs a float. If mylink.mynumber contains the value 56, then the following two functions are equivalent, and return the value 1.29629629629629E+01:

```
AsFloat(56/4.32)
AsFloat(mylink.mynumber/4.32)
```

AsInteger

Treat the given number as an integer.

- **Input**: number
- **Output**: number (integer)
- **Examples**. In the following expression, the input column mynumber contains a double, but the function is output an integer. If mylink.mynumber contains the value 56, then the following two functions are equivalent, and return the value 12:

```
AsInteger(56/4.32)
AsInteger(mylink.mynumber/4.32)
```

MantissaFromDecimal

Returns the mantissa from the given decimal.

- **Input**: number (decimal)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 243.7675, then the following two functions are equivalent, and return the value 7675:

```
MantissaFromDecimal(243.7675)
MantissaFromDecimal(mylink.mynumber)
```

MantissaFromDFloat

Returns the mantissa from the given dfloat.

- **Input**: number (dfloat)
- **Output**: result (dfloat)
- **Examples**. If mylink.number contains the number 1.234412000000000010E +4, then the following function returns the value 1:

```
MantissaFromDFloat(mylink.mynumber)
```

Raw functions

Use the Raw function to obtain the length of the data in a column containing raw data.

The function is in the **Raw** category. The examples show the function as it appears in a **Derivation** field in the Transformer stage.

RawLength

Returns the length of a raw string.

- **Input**: input string (raw)
- **Output**: result (int32)
- **Examples**. If mylink.rawdata contains the raw data from a bitmap, then the following function returns the size of the bitmap in bytes:

```
RawLength(mylink.rawdata)
```

String functions

Use the string functions to manipulate strings.

The following functions are in the **String** category of the expression editor. Square brackets indicate an argument is optional. The examples show the function as it appears in a **Derivation** field in the Transformer stage.

AlNum

Checks whether the given string contains only alphanumeric characters.

- **Input**: string (string)
- **Output**: true/false (int8)
- **Examples**. If mylink.mystring1 contains the string "OED_75_9*E", then the following function would return the value 0 (false).

  ```
  AlNum(mylink.mystring1)
  ```

  If mylink.mystring2 contains the string "12 red roses", then the following function would return the value 1 (true).

  ```
  AlNum(mylink.mystring2)
  ```

Alpha Checks whether the given string contains only alphabetic characters.

- **Input**: string (string)
- **Output**: true/false (int8)
- **Examples**. If mylink.mystring1 contains the string "12 red roses", then the following function would return the value 0 (false).

  ```
  Alpha(mylink.mystring1)
  ```

  If mylink.mystring2 contains the string "twelve red roses", then the following function would return the value 1 (true).

  ```
  Alpha(mylink.mystring2)
  ```

CompactWhiteSpace

Return the string after reducing all consecutive white space to a single space.

- **Input**: string (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring contains the string "too many spaces", then the following function returns the string "too many spaces":

  ```
  CompactWhiteSpace(mylink.mystring)
  ```

Compare

Compares two strings for sorting. The comparison can be left-justified (the default) or right-justified. A right-justified comparison compares numeric substrings within the specified strings as numbers. The numeric strings must occur at the same character position in each string. For example, a right-justified comparison of the strings AB100 and AB99 indicates that AB100 is greater than AB99 since 100 is greater than 99. A right-justified comparison of the strings AC99 and AB100 indicates that AC99 is greater since C is greater than B.

- **Input**: string1 (string), string2 (string), [justification (L or R)]
- **Output**: result (int8), can be -1 for string1 is less than string2, 0 for both strings are the same, 1 for string1 is greater than string2.
- **Examples**. If mylink.mystring1 contains the string "AB99" and mylink.mystring2 contains the string "AB100", then the following function returns the result 1.

  ```
  Compare(mylink.mystring1,mylink.mystring2,L)
  ```

  If mylink.mystring1 contains the string "AB99" and mylink.mystring2 contains the string "AB100", then the following function returns the result -1.

  ```
  Compare(mylink.mystring1,mylink.mystring2,R)
  ```

CompareNoCase

Compares two strings for sorting, ignoring their case.

- **Input**: string1 (string), string2 (string)
- **Output**: result (int8), can be -1 for string1 is less than string2, 0 for both strings are the same, 1 for string1 is greater than string2.
- **Examples**. If mylink.mystring1 contains the string "Chocolate Cake" and mylink.mystring2 contains the string "chocolate cake", then the following function returns the result 0.

  ```
  CompareNoCase(mylink.mystring1,mylink.mystring2)
  ```

CompareNum

Compares the first *n* characters of two strings.

- **Input**: string1 (string), string2 (string), length (int16)
- **Output**: result (int8), can be -1 for string1 is less than string2, 0 for both strings are the same, 1 for string1 is greater than string2.
- **Examples**. If mylink.mystring1 contains the string "Chocolate" and mylink.mystring2 contains the string "Choccy Treat", then the following function returns the result 0.

  ```
  CompareNum(mylink.mystring1,mylink.mystring2,4)
  ```

CompareNumNoCase

Compares the first *n* characters of two strings, ignoring their case.

- **Input**: string1 (string), string2 (string), length (int16)
- **Output**: result (int8), can be -1 for string1 is less than string2, 0 for both strings are the same, 1 for string1 is greater than string2.
- **Examples**. If mylink.mystring1 contains the string "chocolate" and mylink.mystring2 contains the string "Choccy Treat", then the following function returns the result 0.

  ```
  CompareNumNoCase(mylink.mystring1,mylink.mystring2,4)
  ```

Convert

Converts characters in the string designated in *expression*. Converts the characters specified in *fromlist* to the characters specified in *tolist*.

- **Input**: fromlist (string), tolist (string), expression (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring1 contains the string "NOW IS THE TIME", then the following function returns the string "NOW YS XHE XYME".

  ```
  Convert("TI","XY",mylink.mystring1)
  ```

Count Counts the number of times a substring occurs in a string.

- **Input**: string (string), substring (string)
- **Output**: result (int32)
- **Examples**. If mylink.mystring1 contains the string "chocolate drops, chocolate ice cream, chocolate bars", then the following function returns 3.

  ```
  Count(mylink.mystring1,"choc")
  ```

Dcount

Counts the number of delimited fields in a string.

- **Input**: string (string), delimiter (string)
- **Output**: result (int32)
- **Examples**. If mylink.mystring1 contains the string "chocolate drops, chocolate ice cream, chocolate bars", then the following function returns 3.

  ```
  Dcount(mylink.mystring1,",")
  ```

DownCase

Changes all uppercase letters in a string to lowercase.

- **Input**: string (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring1 contains the string "CaMel cAsE", then the following function returns the string "camel case".

  ```
  DownCase(mylink.mystring1)
  ```

DQuote

Encloses a string in double quotation marks.

- **Input**: string (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring1 contains the string needs quotes, then the following function returns the string "needs quotes".

  ```
  DQuote(mylink.mystring1)
  ```

Field Returns one or more substrings located between specified delimiters in a string. The argument *occurrence* specifies which occurrence of the delimiter is to be used as a terminator. The argument *number* optionally specifies how many substrings to return.

- **Input**: string (string), delimiter (string), occurrence (int32), [number (int32)]
- **Output**: result (string)
- **Examples**. If mylink.mystring1 contains the string "chocolate drops, chocolate ice cream, chocolate bars, chocolate dippers", then the following function returns the string " chocolate ice cream".

  ```
  Field(mylink.mystring1,",",2)
  ```

  If mylink.mystring1 contains the string "chocolate drops, chocolate ice cream, chocolate bars, chocolate dippers", then the following function returns the string " chocolate ice cream, chocolate bars".

  ```
  Field(mylink.mystring1,",",2,2)
  ```

Index Finds the starting character position of a substring. The argument *occurrence* specifies which occurrence of the substring is to be located.

- **Input**: string (string), substring (string), occurrence (int32)
- **Output**: result (int32)
- **Examples**. If mylink.mystring1 contains the string "chocolate drops, chocolate ice cream, chocolate bars, chocolate dippers", then the following function returns the value 18.

  ```
  Index(mylink.mystring1,"chocolate",2)
  ```

Left Returns the leftmost *n* characters of a string.

- **Input**: string (string) number (int32)
- **Output**: result (string)
- **Examples**. If mylink.mystring1 contains the string "chocolate drops, chocolate ice cream, chocolate bars, chocolate dippers", then the following function returns the string "chocolate".

  ```
  Left(mylink.mystring1,9)
  ```

Len Returns the length of a string in characters.

- **Input**: string (string)
- **Output**: result (int32)
- **Examples**. If mylink.mystring1 contains the string "chocolate", then the following function returns the value 9.

  ```
  Len(mylink.mystring1)
  ```

Num Returns 1 if string can be converted to a number, or 0 otherwise.

- **Input**: string (string)

- **Output**: result (int32)
- **Examples**. If mylink.mystring1 contains the string "22", then the following function returns the value 1.

  ```
  Num(mylink.mystring1)
  ```

  If mylink.mystring1 contains the string "twenty two", then the following function returns the value 0.

  ```
  Num(mylink.mystring1)
  ```

PadString

Return the string padded with the specified number of pad characters.

- **Input**: string (string) padstring (string) padlength (int32)
- **Output**: result (string)
- **Examples**. IIf mylink.mystring1 contains the string "AB175", then the following function returns the string "AB17500000".

  ```
  PadString(mylink.mystring1,"0",5)
  ```

Right Returns the rightmost *n* characters of a string.

- **Input**: string (string) number (int32)
- **Output**: result (string)
- **Examples**. If mylink.mystring1 contains the string "chocolate drops, chocolate ice cream, chocolate bars, chocolate dippers", then the following function returns the string "dippers".

  ```
  Right(mylink.mystring1,7)
  ```

Soundex

Returns a code which identifies a set of words that are (roughly) phonetically alike based on the standard, open algorithm for SOUNDEX evaluation.

- **Input**: string (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring1 contains the string "Griffin" then the following function returns the code "G615".

  ```
  Soundex(mylink.mystring1)
  ```

  If mylink.mystring1 contains the string "Griphin" then the following function also returns the code "G615".

  ```
  Soundex(mylink.mystring1)
  ```

Space Returns a string of *n* space characters.

- **Input**: length (int32)
- **Output**: result (string)
- **Examples**. If mylink.mylength contains the number 100, then the following function returns a string that contains 100 space characters.

  ```
  Space(mylink.mylength)
  ```

SQuote

Encloses a string in single quotation marks.

- **Input**: string (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring1 contains the string needs quotes, then the following function returns the string 'needs quotes'.

  ```
  SQuote(mylink.mystring1)
  ```

Str Repeats a string the specified number of times.

- **Input**: string (string) repeats (int32)
- **Output**: result (string)
- **Examples**. If mylink.mystring1 contains the string needs "choc", then the following function returns the string "chocchocchocchocchoc".

  ```
  Str(mylink.mystring1,5)
  ```

StripWhiteSpace

Returns the string after removing all whitespace characters from it.

- **Input**: string (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring contains the string "too many spaces", then the following function returns the string "toomanyspaces":

  ```
  StripWhiteSpace(mylink.mystring)
  ```

Trim Remove all leading and trailing spaces and tabs plus reduce internal occurrences to one. The argument *stripchar* optionally specifies a character other than a space or a tab. The argument *options* optionally specifies the type of trim operation to be performed and contains one or more of the following values:

A Remove all occurrences of *stripchar*

B Remove both leading and trailing occurrences of *stripchar*

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 *stripchar*

R Remove leading, trailing, and redundant occurrences of *stripchar*

T Remove all trailing occurrences of *stripchar*

- **Input**: string (string) [stripchar (string)] [options (string)]
- **Output**: result (string)
- **Examples**. If mylink.mystring contains the string " String with whitespace ", then the following function returns the string "String with whitespace":

  ```
  Trim(mylink.mystring)
  ```

  If mylink.mystring contains the string "..Remove..redundant..dots....", then the following function returns the string "Remove.redundant.dots":

  ```
  Trim(mylink.mystring,".")
  ```

  If mylink.mystring contains the string "Remove..all..dots....", then the following function returns the string "Removealldots":

  ```
  Trim(mylink.mystring,".","A")
  ```

  If mylink.mystring contains the string "Remove..trailing..dots....", then the following function returns the string "Remove..trailing..dots":

  ```
  Trim(mylink.mystring,".","T")
  ```

TrimB Removes all trailing spaces and tabs from a string.

- **Input**: string (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring contains the string "too many trailing spaces ", then the following function returns the string "too many trailing spaces":

  ```
  TrimB(mylink.mystring)
  ```

TrimF Removes all leading spaces and tabs from a string.

- **Input**: string (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring contains the string " too many leading spaces", then the following function returns the string "too many leading spaces":

  ```
  TrimF(mylink.mystring)
  ```

TrimLeadingTrailing
Removes all leading and trailing spaces and tabs from a string.

- **Input**: string (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring contains the string " too many spaces ", then the following function returns the string "too many spaces":

  ```
  TrimLeadingTrailing(mylink.mystring)
  ```

UpCase
Changes all lowercase letters in a string to uppercase.

- **Input**: string (string)
- **Output**: result (string)
- **Examples**. If mylink.mystring1 contains the string "CaMel cAsE", then the following function returns the string "CAMEL CASE".

  ```
  UpCase(mylink.mystring1)
  ```

Vector function

Use the vector function to access an element in a vector column.

The function is in the **Vector** category of the expression editor. The examples show the function as it appears in a **Derivation** field in the Transformer stage.

ElementAt
Accesses an element of a vector. The vector index starts at 0. The function can be used as part of, or the whole of an expression.

- **Input**: input_column (column name) index (int)
- **Output**: element of vector
- **Examples**. The following example outputs the second element of the vector in the column mylink.myvector.

  ```
  ElementAt(mylink.myvector, 2)
  ```

  The following example outputs the second element of the vector in the column mylink.myvector and adds one it.

  ```
  ElementAt(mylink.myvector, 2) + 1
  ```

Type conversion functions

Use the type conversion functions to change the type of an argument.

The following functions are in the **Type Conversion** category of the expression editor. Square brackets indicate an argument is optional. The default date format is %yyyy-%mm-%dd.

The examples show the function as it appears in a **Derivation** field in the Transformer stage.

Char Generates an ASCII character from its numeric code value. You can optionally specify the allow8bits argument to convert 8-bit ASCII values.

- **Input**: code (number), [allow8bits]
- **Output**: result (char)
- **Examples**. The following example outputs the ASCII code 65 as the character A.

  ```
  Char(65)
  ```

DateToString

Returns the string representation of the given date. The format of the string can optionally be specified.

- **Input**: date (date), [format (string)]
- **Output**: result (string)
- **Examples**. The following example outputs the date contained in the column mylink.mydate to a string. If mylink.mydate contains the date 18th August, 2009, then the output string is "2009-08-18":

  ```
  DateToString(mylink.mydate)
  ```

  The following example outputs the date contained in the column mylink.mydate to a string with the format dd:mm:yyyy. If mylink.mydate contained the date 18th August, 2009, then the output string would be "18:08:2009":

  ```
  DateToString(mylink.mydate, "%dd:%mm:%yyyy")
  ```

DateToDecimal

Returns the given date as a packed decimal value. If your target decimal specifies a scale, part of the date appears after the decimal point. You can optionally specify a format string that specifies how the date is stored in the decimal number. The default format string is "%yyyy%mm%dd", so, for example, the date 2009-08-25 is stored as the decimal number 20090825. Format strings can only specify a format that contains numbers. For example, you cannot specify a format string such as "%yyyy-%mm-%dd", because the hyphen character (-) cannot be stored in a packed decimal value. The following tokens are valid for conversions to or from decimal values:

%yyyy (four-digit year)

%yy (two-digit year)

%NNNNyy (two-digit year with cutoff)

%mm (two-digit month)

%dd (two-digit day of month)

%ddd (three-digit day of year)

The literal digits 0 to 9 are also valid.

- **Input**: basedate (date) [, format (string)]
- **Output**: converted_date (decimal)
- **Examples**. If the column mylink.basedate contains the date 2012-08-18, then the following function stores the date as the decimal number 18082012:

  ```
  DateToDecimal (mylink.basedate, "%dd%mm%yyyy")
  ```

  If the column mylink.basedate contains the date 2012-08-18, and the target column has a length of 10 and a scale of 2, then the following function stores the date as the decimal number 201208.18:

  ```
  DateToDecimal (mylink.basedate)
  ```

DecimalToDate

Returns the given packed decimal as a date. Both the sign and the scale of the decimal number are ignored when it is converted to a date. You can optionally specify a format string that specifies how the date is stored in the decimal number. The default format string is "%yyyy%mm%dd", so, for example, the date 2009-08-25 is stored as the decimal number

20090825. Format strings can only specify a format that contains numbers. For example, you cannot specify a format string such as "%yyyy-%mm-%dd", because the hyphen character (-) cannot be stored in a packed decimal value. The following tokens are valid for conversions to or from decimal values:

%yyyy (four-digit year)

%yy (two-digit year)

%NNNNyy (two-digit year with cutoff)

%mm (two-digit month)

%dd (two-digit day of month)

%ddd (three-digit day of year)

The literal digits 0 to 9 are also valid.

- **Input**: basedec (decimal) [, format (string)]
- **Output**: date
- **Examples**. If the column mylink.mydecdata contains the value 18082012, then the following function returns the date 2012-08-18:

  ```
  DecimalToDate (mylink.basedate, "%dd%mm%yyyy")
  ```

  If the column mylink.mydecdata contains the value -201208.18, then the following function returns the date 2012-08-18:

  ```
  DecimalToDate (mylink.basedate)
  ```

DecimalToDecimal

Returns the given decimal in decimal representation with precision and scale specified in the target column definition. The argument *rtype* optionally specifies a rounding type, and is set to one of the following values:

ceil. Round the source field toward positive infinity. For example, 1.4 -> 2, -1.6 -> -1.

floor. Round the source field toward negative infinity. For example, 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. For example, 1.4 -> 1, 1.5 -> 2, -1.4 -> -1, -1.5 -> -2.

trunc_zero. Discard any fractional digits to the right of the rightmost 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. For example, 1.6 -> 1, -1.6 -> -1.

- **Input**: decimal (decimal) [,rtype (string)]
- **Output**: result (decimal)
- **Examples**. If the column mylink.mydec contains the decimal number 2.5345, the following function returns the decimal number 00000002.54.

  ```
  DecimalToDecimal(mylink.mydec,"ceil")
  ```

  The following function returns the decimal number 00000002.53.

  ```
  DecimalToDecimal(mylink.mydec,"floor")
  ```

  The following function returns the decimal number 00000002.53.

  ```
  DecimalToDecimal(mylink.mydec,"trunc_zero")
  ```

  The following function returns the decimal number 00000002.53.

  ```
  DecimalToDecimal(mylink.mydec,"round_inf")
  ```

In all these examples, the target decimal has a length of 10 and a scale of 2.

DecimalToDFloat

Returns the given decimal in dfloat representation. The argument "fix_zero" optionally specifies that all zero decimal values are regarded as valid (by default, decimal numbers comprising all zeros are treated as invalid).

- **Input**: decimal (decimal) [,"fix_zero"]
- **Output**: result (dfloat)
- **Examples**. If the column mylink.mydec contains the decimal number 00000004.00 the following function returns the dfloat number 4.00000000000000000E+00.

```
DecimalToDFloat(mylink.mydec,"fix_zero")
```

If the column mylink.mydec contains the decimal number 00012344.00 the following function returns the dfloat number 1.23440000000000000E+04.

```
DecimalToDFloat(mylink.mydec,"fix_zero")
```

If the column mylink.mydec contains the decimal number 00012344.120 the following function returns the dfloat number 1.23441200000000010E+04.

```
DecimalToDFloat(mylink.mydec,"fix_zero")
```

If the column mylink.mydec contains the decimal number 00012344.120 the following function returns the dfloat number 1.23441200000000010E+04.

```
DecimalToDFloat(mylink.mydec)
```

If the column mylink.mydec contains the decimal number 00012344.000 the following function returns the dfloat number 1.23440000000000000E+04.

```
DecimalToDFloat(mylink.mydec)
```

DecimalToString

Returns the given decimal as a string. The argument "fix_zero" optionally specifies that all zero decimal values are regarded as valid (by default, decimal numbers comprising all zeros are treated as invalid). This covers the case where the sign bits of the packed decimal representation are all 0 as well as all the content digits. This cast is not considered valid unless "fix_zero" is true.

- **Input**: decimal (decimal) [,"fix_zero"]
- **Output**: result (string)
- **Examples**. If the column mylink.mydec contains the decimal number 00000004.00, the following function returns the string "4":

```
DecimalToString(mylink.mydec,"suppress_zero")
```

If the column mylink.mydec contains the decimal number 00000004.00, the following function returns the string "00000000000000000000000004.0000000000".

```
DecimalToString(mylink.mydec,"fix_zero")
```

If the column mylink.mydec contains the decimal number 00012344.00, the following function returns the string "12344".

```
DecimalToString(mylink.mydec,"suppress_zero")
```

If the column mylink.mydec contains the decimal number 00012344.00, the following function returns the string "00000000000000000000012344.0000000000".

```
DecimalToString(mylink.mydec,"fix_zero")
```

If the column mylink.mydec contains the decimal number 00012344.120, the following function returns the string "00000000000000000000012344.1200000000".

```
DecimalToString(mylink.mydec,"fix_zero")
```

If the column mylink.mydec contains the decimal number 00012344.120, the following function returns the string "12344.12":

```
DecimalToString(mylink.mydec,"suppress_zero")
```

If the column mylink.mydec contains the decimal number 00012344.120, the following function returns the string "00000000000000000000000012344.120000000".

```
DecimalToString(mylink.mydec)
```

If the column mylink.mydec contains the decimal number 00012344.000, the following function returns the string "00000000000000000000000012344.0000000000".

```
DecimalToString(mylink.mydec)
```

DecimalToTime

Returns the given packed decimal as a time. You can optionally specify a format string that specifies how the time is stored in the decimal number. The default format string is " %hh%nn%ss", so, for example, the time 14:03:22 is stored as the decimal number 140322. Format strings can only specify a format that contains numbers. For example, you cannot specify a format string such as "%hh:%nn:%ss", because the colon character (:) cannot be stored in a packed decimal value. The following tokens are valid for conversions to or from decimal values:

%hh (two-digit hours using 24-hour clock)

%nn (two-digit minutes)

%ss (two-digit seconds)

%ss.*N* (two-digit seconds, plus the number of fractional digits allowed. The number of fractional digits is from one to six inclusive).

The literal digits 0 to 9 are also valid.

If your specified format includes microseconds (for example, %ss.4), then the position of the decimal point is inferred in the decimal value. The position of the decimal point does not have to coincide with the specified scale of the decimal (for example, scale = 4).

- **Input**: time (time) [, format (string)]
- **Output**: result (decimal)
- **Examples**: If the column mylink.mytimedec contains the decimal value 200658, then the following function returns the time 20:06:58:

  ```
  DecimalToTime(mylink.mytimedec)
  ```

  If the column mylink.mytimedec contains the decimal value 580620, then the following function returns the time 20:06:58:

  ```
  DecimalToTime(mylink.mytimedec, "%ss%nn%hh")
  ```

DecimalToTimestamp

Returns the given packed decimal as a timestamp. You can optionally specify a format string that specifies how the timestamp is stored in the decimal number. The default format string is " %yyyy%mm%dd%hh%nn%ss", so, for example, the timestamp 2009-08-25 14:03:22 is stored as the decimal number 20090825140322. Format strings can only specify a format that contains numbers. For example, you cannot specify a format string such as "%yyyy/%mm/%dd%hh:%nn:%ss", because the slash character (/) and the colon character (:) cannot be stored in a packed decimal value. The following tokens are valid for conversions to or from decimal values:

%yyyy (four-digit year)

%yy (two-digit year)

%NNNNyy (two-digit year with cutoff)

%mm (two-digit month)

%dd (two-digit day of month)

%ddd (three-digit day of year)

%hh (two-digit hours using 24-hour clock)

%nn (two-digit minutes)

%ss (two-digit seconds)

%ss.*N* (two-digit seconds, plus the number of fractional digits allowed. The number of fractional digits is from one to six inclusive).

The literal digits 0 to 9 are also valid.

If your specified format includes microseconds (for example, %ss.4), then the position of the decimal point is inferred in the decimal value. The position of the decimal point does not have to coincide with the specified scale of the decimal (for example, scale = 4).

- **Input**: timestamp (timestamp) [, format (string)]
- **Output**: result (decimal)
- **Examples**: If the column mylink.mytimestampdec contains the value 19580818200658, then the following function returns the timestamp 1958–08–18 20:06:58:

  ```
  DecimalToTimestamp(mylink.mytimestampdec)
  ```

  If the column mylink.mytimestampdec contains the decimal value 200658580818, then the following function returns the timestamp 1958–08–18 20:06:58:

  ```
  DecimalToTimestamp(mylink.mytimestampdec, "%hh%nn%ss%yy%mm%dd")
  ```

DFloatToDecimal

Returns the given dfloat in decimal representation. The argument *rtype* optionally specifies a rounding type, and is set to one of the following values:

ceil. Round the source field toward positive infinity. For example, 1.4 -> 2, -1.6 -> -1.

floor. Round the source field toward negative infinity. For example, 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. For example, 1.4 -> 1, 1.5 -> 2, -1.4 -> -1, -1.5 -> -2.

trunc_zero. Discard any fractional digits to the right of the rightmost 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. For example, 1.6 -> 1, -1.6 -> -1.

- **Input**: number (dfloat), [rtype (string)]
- **Output**: result (decimal)
- **Examples**. If the column mylink.myfloat contains the dfloat number 2.534, the following function returns the decimal number 00000002.54.

  ```
  DFloatToDecimal(mylink.mydec,"ceil")
  ```

  If the column mylink.myfloat contains the dfloat number 2.534, the following function returns the decimal number 00000002.53.

  ```
  DFloatToDecimal(mylink.mydec,"floor")
  ```

  If the column mylink.myfloat contains the dfloat number 2.534, the following function returns the decimal number 00000002.53.

  ```
  DFloatToDecimal(mylink.mydec,"trunc_zero")
  ```

  If the column mylink.myfloat contains the dfloat number 2.534, the following function returns the decimal number 00000002.53.

```
DFloatToDecimal(mylink.mydec,"round_inf")
```

DfloatToStringNoExp

Returns the given dfloat in its string representation with no exponent, using the specified scale.

- **Input**: number (dfloat), scale (string)
- **Output**: result (string)
- **Examples**. If the column mylink.myfloat contains the dfloat number 2.534, then the following function returns the string 00000002.50:

```
DfloatToStringNoExp(mylink.myfloat,2)
```

IsValid

Returns whether the given string is valid for the given type. Valid types are "date", "decimal", "dfloat", "sfloat", "int8", "uint8", "int16", "uint16", "int32", "uint32", "int64", "uint64", "raw", "string", "time", "timestamp", "ustring". For data types of date, time, and timestamp, you can optionally specify a format string. The format string describes the format that your input data uses when it differs from the default formats for date, time, or timestamp. The default format for date is "%yyyy-%mm-%dd". The default format for time is "%hh:%mm:%ss". The default format for timestamp is "%yyyy-%mm-%dd %hh:%mm:%ss". This function does not log warnings.

- **Input**: type (string), teststring (string) [, format (string)]
- **Output**: result (int8)
- **Examples**. If the column mylink.mystring contains the string "1", then the following function returns the value 1.

```
IsValid("int8",mylink.mystring)
```

If the column mylink.mystring contains the string "380096.06", then the following function returns the value 0.

```
IsValid("int8",mylink.mystring)
```

IsValidDate

Returns whether the given value is valid for the type date. This function logs warnings.

- **Input**: testdate (date)
- **Output**: result (int8)
- **Examples**. If the column mylink.mydate contains the date 2011-09-13, then the following function returns the value 1.

```
IsValidDate(mylink.mydate)
```

If the column mylink.mydate contains the string "380096.06", then the following function returns the value 0, because the converted string is not a valid date.

```
IsValidDate(StringToDate (mylink.mydate))
```

IsValidDecimal

Returns whether the given value is valid for the type decimal. If the allzerosflag is set to 0, then an all-zeroes representation is not valid. The allzerosflag is set to zero by default.

- **Input**: testvalue (decimal) [, allzerosflag (uint8)]
- **Output**: result (int8)
- **Examples**. If the column mylink.mynum contains the value 310007.65, then the following function returns the value 1.

```
IsValidDecimal(mylink.mynum)
```

If the column mylink.mynum contains the string "wake-robin", then the following function returns the value 0, because the converted string is not a valid decimal.

```
IsValidDecimal(StringToDecimal (mylink.mynum))
```

IsValidTime

Returns whether the given time is valid for the type time.

- **Input**: testtime (time)
- **Output**: result (int8)
- **Examples**. If the column mylink.mytime contains the time 23:09:22, then the following function returns the value 1:

  ```
  IsValidTime(mylink.mytime)
  ```

  If the column mylink.mydate contains the string "IbnKayeed", then the following function returns the value 0, because the converted string is not a valid time.

  ```
  IsValidTime(StringToTime (mylink.mytime))
  ```

IsValidTimestamp

Returns whether the given timestamp is valid for the type timestamp.

- **Input**: testtimestamp (timestamp)
- **Output**: result (int8)
- **Examples**. If the column mylink.mytimestamp contains the time 2011-09-13 23:09:22, then the following function returns the value 1:

  ```
  IsValidTimestamp(mylink.mytimestamp)
  ```

  If the column mylink.mytimestamp contains the string "one of two", then the following function returns the value 0, because the converted string is not a valid timestamp.

  ```
  IsValidTimestamp(StringToTimestamp (mylink.mytimestamp))
  ```

RawNumAt

Returns the integer value at the specified index value in the specified raw field. The index starts at 0.

- **Input**: rawfield (raw), index (int32)
- **Output**: result (int32)
- **Examples**. If the column mylink.myraw contains a raw value derived from the string "hello", then the following function returns the integer 0x68 (the ASCII code for the character h):

  ```
  RawNumAt(mylink.myraw, 0)
  ```

  If the column mylink.myraw contains a raw value derived from the string "hello", then the following function returns 0 because the specified index is out of range:

  ```
  RawNumAt(mylink.myraw, 12)
  ```

RawToString

Returns the given raw value as a string representation. You must ensure that the raw input value contains a sequence of bytes that are valid as characters in the target character set in which the output string is used. For example, the raw value { 0xE0 0x41 0x42 } is not a valid sequence of UTF-8 characters, since the lead byte, 0xE0, is supposed to be followed by a byte in the range [0x80..0xBF]. If a raw value { xE0 x41 x42 } is passed to the RawToString function, there could be an error if the output string is then accessed as if it were encoded in UTF-8.

- **Input**: rawfield (raw)
- **Output**: result (string)
- **Examples**. If the column mylink.myraw contains the value { 0x31 0x31 0x30 0x35 0x32 0x32 0x30 0x39 }, then the following function returns the string "11052209".

  ```
  RawNumAt(mylink.myraw)
  ```

Seq Generates a numeric code value from an ASCII character. You can optionally specify the allow8bits argument to convert 8-bit ASCII values.

- **Input**: Seq (char)

- **Output**: result (number)
- **Examples**. The following example outputs the character A as the ASCII code 65.

```
Seq("A")
```

SeqAt Returns the numeric code point value of the character at the specified position in the given string. The index starts at 0. If the specified index is out of range, the function returns 0.

- **Input**: basestring (string), index (int32)
- **Output**: result (int32)
- **Examples**. If the column mylink.mystring contains the string "horse", then the following function returns the value 0x6F (that is, the ASCII value of the character o).

```
SeqAt(mylink.mystring, 1)
```

StringToDate

Returns a date from the given string in the given format. You do not have to specify a format string if your string contains a date in the default format yyyy-mm-dd.

- **Input**: string (string) [,format (string)]
- **Output**: result (date)
- **Examples**: If the column mylink.mystring contains the string ␣1958–08–18␣, then the following function returns the date 1958–08–18.

```
StringToDate(mylink.mystring)
```

If the column mylink.mystring contains the string ␣18:08:1958␣, then the following function returns the date 1958–08–18.

```
StringToDate(mylink.mystring,"%dd:%mm:%yyyy")
```

StringToDecimal

Returns the given string as a decimal representation. The argument *rtype* optionally specifies a rounding type, and is set to one of the following values:

ceil. Round the source field toward positive infinity. For example, 1.4 -> 2, -1.6 -> -1.

floor. Round the source field toward negative infinity. For example, 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. For example, 1.4 -> 1, 1.5 -> 2, -1.4 -> -1, -1.5 -> -2.

trunc_zero. Discard any fractional digits to the right of the rightmost 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. For example, 1.6 -> 1, -1.6 -> -1.

- **Input**: string (string), [rtype (string)]
- **Output**: result (decimal)
- **Examples**. If the column mylink.mystring contains the string "19982.22", and the target is defined as having a precision of 7 and a scale of 2, then the following function returns the decimal 19983.22.

```
StringToDecimal(mylink.mystring)
```

If the column mylink.mystring contains the string "19982.2276", and the target is defined as having a precision of 7 and a scale of 2, then the following function returns the decimal 19983.23.

```
StringToDecimal(mylink.mystring,"ceil")
```

StringToRaw

Returns a string in raw representation.

- **Input**: string (string)

- **Output**: result (raw)
- **Examples**. If the column mylink.mystring contains the string "hello", and the target column is defined as being of type Binary then the following function returns the value { 0x68 0x65 0x6C 0x6C 0x6F }.

  ```
  StringToRaw(mylink.mystring)
  ```

StringToTime

Returns a time representation of the given string.

- **Input**: string (string), [format (string)]
- **Output**: result (time)
- **Examples**: If the column mylink.mystring contains the string "20:06:58", then the function returns a time of 20:06:58.

  ```
  StringToTime(mylink.mystring)
  ```

  If the column mylink.mystring contains the string "20: 6:58", then the function returns a time of 20:06:58.

  ```
  StringToTime(mylink.mystring,"%(h,s):$(n,s):$(s,s)")
  ```

StringToTimestamp

Returns a time representation of the given string.

- **Input**: string (string) [format (string)]
- **Output**: result (time)
- **Examples**: If the column mylink.mystring contains the string "1958–08–08 20:06:58", then the function returns the timestamp 1958–08–08 20:06:58.

  ```
  StringToTimestamp(mylink.mystring)
  ```

  If the column mylink.mystring contains the string "8/ 8/1958 20: 6:58", then the function returns the timestamp 1958–08–08 20:06:58.

  ```
  StringToTimestamp(mylink.mystring, "%(d,s)/%(m,s)/%yyyy%(h,s):$(n,s):$(s,s)")
  ```

StringToUstring

Returns a ustring from the given string, optionally using the specified map (otherwise uses project default).

- **Input**: string (string), [mapname(string)]
- **Output**: result (ustring)
- **Examples**: If the column mylink.mystring contains the string "11052009", then the following function returns the ustring "11052009"

  ```
  StringToUstring(mylink.mystring)
  ```

TimestampToDate

Returns a date from the given timestamp.

- **Input**: timestamp (timestamp)
- **Output**: result (date)
- **Examples**: If the column mylink.mytimestamp contains the timestamp 1958–08–18 20:06:58, then the following function returns the date 1958–08–18:

  ```
  TimestampToDate(mylink.mytimestamp)
  ```

TimestampToDecimal

Returns the given timestamp as a packed decimal. You can optionally specify a format string that specifies how the timestamp is stored in the decimal number. The default format string is " %yyyy%mm%dd%hh%nn%ss", so, for example, the timestamp 2009-08-25 14:03:22 is stored as the decimal number 20090825140322. Format strings can only specify a format that contains numbers. For example, you cannot specify a format string such as "%yyyy/%mm/%dd%hh:%nn:%ss",

because the slash character (/) and the colon character (:) cannot be stored in a packed decimal value. The following tokens are valid for conversions to or from decimal values:

%yyyy (four-digit year)

%yy (two-digit year)

%NNNNyy (two-digit year with cutoff)

%mm (two-digit month)

%dd (two-digit day of month)

%ddd (three-digit day of year)

%hh (two-digit hours using 24-hour clock)

%nn (two-digit minutes)

%ss (two-digit seconds)

%ss.*N* (two-digit seconds, plus the number of fractional digits allowed. The number of fractional digits is from one to six inclusive).

The literal digits 0 to 9 are also valid.

If your specified format includes microseconds (for example, %ss.4), then the position of the decimal point is inferred in the decimal value. The position of the decimal point does not have to coincide with the specified scale of the decimal (for example scale = 4).

- **Input**: timestamp (timestamp) [, format (string)]
- **Output**: result (decimal)
- **Examples**: If the column mylink.mytimestamp contains the timestamp 1958–08–18 20:06:58, then the following function returns the decimal value 19580818200658:

  ```
  TimestampToDecimal(mylink.mytimestamp)
  ```

  If the column mylink.mytimestamp contains the timestamp 1958–08–18 20:06:58, then the following function returns the decimal value 200658580818:

  ```
  TimestampToDecimal(mylink.mytimestamp, "%hh%nn%ss%yy%mm%dd")
  ```

TimestampToString

Returns a string from the given timestamp.

- **Input**: timestamp (timestamp) [format (string)]
- **Output**: result (string)
- **Examples**: If the column mylink.mytimestamp contains the timestamp 1958–08–1820:06:58, then the function returns the string "1958–08–1820:06:58".

  ```
  TimestampToString(mylink.mytimestamp)
  ```

  If the column mylink.mytimestamp contains the timestamp 1958–08–1820:06:58, then the function returns the string "18/08/1958 20:06:58":

  ```
  TimestampToString(mylink.mytimestamp, "%dd/%mm/%yyyy %hh:$nn:$ss")
  ```

TimestampToTime

Returns the string representation of the given timestamp.

- **Input**: timestamp (timestamp)
- **Output**: result (time)
- **Examples**: If the column mylink.mytimestamp contains the timestamp 1958–08–1820:06:58, then the function returns the time 20:06:58:

  ```
  TimestampToTime(mylink.mytimestamp)
  ```

TimeToString

Returns a string from the given time.

- **Input**: timestamp (timestamp) [format (string)]
- **Output**: result (time)
- **Examples**: If the column mylink.mytime contains the time 20:06:58, then the following function returns the string "20:06:58":

  ```
  TimeToString(mylink.mytime)
  ```

  If the column mylink.mytime contains the time 20:06:58, then the following function returns the string "58:06:20":

  ```
  TimeToString(mylink.mytime, "%ss:$nn:$hh")
  ```

TimeToDecimal

Returns the given time as a packed decimal. You can optionally specify a format string that specifies how the time is stored in the decimal number. The default format string is " %hh%nn%ss", so, for example, the time 14:03:22 is stored as the decimal number 140322. Format strings can only specify a format that contains numbers. For example, you cannot specify a format string such as "%hh:%nn:%ss", because the colon character (:) cannot be stored in a packed decimal value. The following tokens are valid for conversions to or from decimal values:

%hh (two-digit hours using 24-hour clock)

%nn (two-digit minutes)

%ss (two-digit seconds)

%ss.*N* (two-digit seconds, plus the number of fractional digits allowed. The number of fractional digits is from one to six inclusive).

The literal digits 0 to 9 are also valid.

If your specified format includes microseconds (for example, %ss.4), then the position of the decimal point is inferred in the decimal value. The position of the decimal point does not have to coincide with the specified scale of the decimal (for example scale = 4).

- **Input**: time (time) [, format (string)]
- **Output**: result (decimal)
- **Examples**: If the column mylink.mytime contains the time 20:06:58, then the following function returns the decimal value 200658:

  ```
  TimeToDecimal(mylink.mytime)
  ```

  If the column mylink.mytime contains the time 20:06:58, then the following function returns the decimal value 580620:

  ```
  TimeToDecimal(mylink.mytime, "%ss%nn%hh")
  ```

UstringToString

Returns a string from the given ustring, optionally using the specified map (otherwise uses project default).

- **Input**: string (ustring) [, mapname(string)]
- **Output**: result (string)
- **Examples**: If the column mylink.myustring contains the ustring "11052009", then the following function returns the string "11052009":

  ```
  UstringToString(mylink.myustring)
  ```

Utility functions

The utility functions have a variety of purposes.

The following functions are available in the Utility category (square brackets indicate an argument is optional):

GetEnvironment

Returns the value of the given environment variable.

- **Input**: environment variable (string)
- **Output**: result (string)
- **Examples**. If you queried the value of the environment variable name APT_RDBMS_COMMIT_ROWS then the following derivation might return the value "2048".

```
GetEnvironment("APT_RDBMS_COMMIT_ROWS")
```

GetSavedInputRecord

This function is used to implement the aggregating of data on the input link of a Transformer stage. You call the GetsSavedInputRecord function to a retrieve a copy of an input row that you have previously saved to a cache area. The function retrieves the next input row from the cache (in the order in which they were saved to the cache) and makes it the current input row. The retrieved row overrides what was the current input row, and so any derivation using an input column value will use the value of that column in the input row retrieved from the cache, not what was previously the current input row. You must call GetSavedInputRecord in a loop variable derivation, you cannot call it from anywhere else. For example, you cannot call GetSavedInputRecord in the Loop Condition expression. You can call GetSavedInputRecord, multiple times and retrieve the next cached row on each call. Use the SaveInputRecord function to store rows to the cache. GetSavedInputRecord returns the cache index number of the record retrieved from that cache.

- **Input**: -
- **Output**: cache_index_number
- **Examples**. The following example is the derivation of a loop variable named SavedRecordIndex in a Transformer stage:

```
SavedRecordIndex:  GetSavedInputRecord()
```

NextSKChain

This function is used in the Slowly Changing Dimension stage as the derivation for a column with the SKChain purpose code. The function is not used in the Transformer stage. NextSKChain returns the value of the surrogate key column for the next row in the chain, or the value that has been specified to use for the last record in the chain.

- **Input**: last_chain_value (int64)
- **Output**: surrogate_key_value (int64)
- **Examples**. If you specify the following function in the derivation field for a SKChain column in an SCD stage, the output column contains the value of the surrogate key of the next record in the chain, or the value 180858 if this is the last row in the chain.

```
NextSKChain(180858)
```

NextSurrogateKey

Returns the value of the next surrogate key. You must have previously set up your surrogate key source, and defined details on the Surrogate Key tab of the Stage page of the Transformer properties window.

- **Input**: -
- **Output**: surrogate_key_value (int64)
- **Example**. The derivation field of your surrogate key column contains the following function:

```
NextSurrogateKey()
```

PrevSKChain

This function is used in the Slowly Changing Dimension stage as the derivation for a column with the SKChain purpose code. The function is not used in the Transformer stage. PrevSKChain

Returns the value of the surrogate key column for the previous record in the chain, or the value that has been specified to use for the first record in the chain.

- **Input**: first_chain_value (int64)
- **Output**: surrogate_key_value (int64)
- **Examples**. If you specify the following function in the derivation field for a SKChain column in an SCD stage, the output column contains the value of the surrogate key of the previous record in the chain, or the value 121060 if this is the last row in the chain.

  ```
  PrevSKChain(121060)
  ```

SaveInputRecord

This function is used to implement the aggregating of data on the input link of a Transformer stage. You call the SaveInputRecord function to save a copy of the current input row to a cache area. The function returns the count of records in the cache, starting from 1. You can call SaveInputRecord from within the derivation of a stage variable in the Transformer stage. You can call SaveInputRecord multiple times for the same input row. The first call adds the input row to the cache and each subsequent call adds a duplicate of that same input row into the cache. So, for example, if SaveInputRecord is called three times for one input record, then the cache will contain three rows, each identical to the original input row. Use the GetSavedInputRecord function to retrieve the rows that you have stored.

- **Input**: -
- **Output**: cache_record_count (int64)
- **Examples**. The following example is the derivation of a stage variable named NumSavedRecords in a Transformer stage:

  ```
  NumSavedRecords:  SaveInputRecord()
  ```

Appendix C. Fillers

Fillers are created when you load columns from COBOL file definitions that represent simple or complex flat files. Since these file definitions can contain hundreds of columns, you can choose to collapse sequences of unselected columns into FILLER items. This maintains the byte order of the columns, and saves storage space and processing time.

This appendix gives examples of filler creation for different types of COBOL structures, and explains how to expand fillers later if you need to reselect any columns.

Creating fillers

Unlike other parallel stages, Complex Flat File stages have stage columns. You load columns on the Records tab of the Complex Flat File Stage page.

First, you select a table from the Table Definitions window. Next, you select columns from the Select Columns From Table window. This window has an **Available columns** tree that displays COBOL structures such as groups and arrays, and a **Selected columns** list that displays the columns to be loaded into the stage. The **Create fillers** check box is selected by default.

When columns appear on the Records tab, FILLER items are shown for the unselected columns. FILLER columns have a native data type of CHARACTER and a name of FILLER_XX_YY, where XX is the start offset and YY is the end offset. Fillers for elements of a group array or an OCCURS DEPENDING ON (ODO) column have the name of FILLER_NN, where NN is the element number. The NN begins at 1 for the first unselected group element and continues sequentially. Any fillers that follow an ODO column are also numbered sequentially.

Level numbers of column definitions, including filler columns, are changed after they are loaded into the Complex Flat File stage. However, the underlying structure is preserved.

Filler creation rules

The rules for filler creation are designed to preserve the storage length of all columns that are replaced by a filler column. They allow a filler column to be expanded back to the original set of defining columns, each having the correct name, data type, and storage length.

Procedure

1. A filler column will replace one or more original columns with a storage length that is equal to the sum of the storage length of each individual column that is being replaced.
2. Separate fillers are created when column level numbers decrease. For example, if an unselected column at level 05 follows an unselected column at level 10, separate fillers are created for the columns at the 05 and 10 levels.
3. Any ODO column and its associated `depending on' column will be automatically selected and not replaced by a filler column.
4. If a REDEFINE column is selected, the column that it is redefining is also automatically selected and will not be included as part of a filler column.
5. If two fillers share the same storage offset (such as for a REDEFINE) the name of the subsequent fillers will be FILLER_XX_YY_NN, where NN is a sequential number that begins at 1.
6. If the starting or ending column for a filler is the child of a parent that contains an OCCURS clause, then the generated FILLER name will be FILLER_NN instead of FILLER_XX_YY.

Results

The remaining rules are explained through the following set of examples.

Filler creation examples

The following examples explain filler creation for groups, REDEFINES, and arrays using different scenarios for column selection. The source table contains single columns, a group that redefines a column, a nested group with an array, and a column that redefines another column:

```
05   A              PIC X(1).
05   B              PIC X(8).
05   GRP1           REDEFINES B.
     10   C1        PIC X(1).
     10   C2        PIC X(1).
     10   GRP2      OCCURS 2 TIMES.
          15   D1   PIC X(1).
          15   D2   PIC X(1).
          15   D3   PIC X(1).
05   E              PIC X(1).
05   F              PIC 9(1) REDEFINES E.
05   G              PIC X(1).
```

Select a simple column

Suppose a simple column, A, is selected from the table. On the Columns tab, a single filler is created with the name FILLER_2_11 and a length of 10. The length represents the sum of the lengths of columns B (8), E (1), and G (1). GRP1 and its elements, along with column F, are excluded because they redefine other columns that were not selected.

The file layout is:

Table 148. File layout

Column	Picture Clause	Starting column	Ending column	Storage length
A	PIC X(1).	1	1	1
FILLER_2_11	PIC X(10).	2	11	10

Select a column that is redefined by a group

Suppose column B is selected, which is redefined by GRP1. Two fillers are created, one for column A and the other for columns E and G. Since GRP1 redefines column B, it is dropped along with its elements. Column F is also dropped because it redefines column E. The dropped columns will be available during filler expansion (see “Expanding fillers” on page 687).

The file layout is:

Table 149. File layout

Column	Picture Clause	Starting column	Ending column	Storage length
FILLER_1_1	PIC X(1).	1	1	1
B	PIC X(8).	2	9	8
FILLER_2_11	PIC X(2).	10	11	2

Select a group column that redefines a column

Next, suppose a group column is selected (GRP1) that redefines another column (B). This time three fillers are created: one for column A, one for columns C1 through D3 (which are part of GRP1), and one for columns E and G. Column B is preserved since it is redefined by the selected group column.

The file layout is:

Table 150. File layout

Column	Picture Clause	Starting column	Ending column	Storage length
FILLER_1_1	PIC X(1).	1	1	1
B	PIC X(8).	2	9	8
GRP1	REDEFINES B.	2	9	8
FILLER_2_9	PIC X(8).	2	9	8
FILLER_10_11	PIC X(2).	10	11	2

Select a group element

Consider what happens when a group element (C1) is selected by itself. Three fillers are created: one for column A, one for columns C2 through D3, and one for columns E and G. Since an element of GRP1 is selected and GRP1 redefines column B, both column B and GRP1 are preserved.

The file layout is:

Table 151. File layout 1

Column	Picture Clause	Starting column	Ending column	Storage length
FILLER_1_1	PIC X(1).	1	1	1
B	PIC X(8).	2	9	8
GRP1	REDEFINES B.	2	9	8
C1	PIC X(1).	2	2	1
FILLER_3_9	PIC X(7).	3	9	7
FILLER_10_11	PIC X(2).	10	11	2

The next example shows what happens when a different group element is selected, in this case, column C2. Four fillers are created: one for column A, one for column C1, one for GRP2 and its elements, and one for columns E and G. Column B and GRP1 are preserved for the same reasons as before.

The file layout is:

Table 152. File layout 2

Column	Picture Clause	Starting column	Ending column	Storage length
FILLER_1_1	PIC X(1).	1	1	1
B	PIC X(8).	2	9	8
GRP1	REDEFINES B.	2	9	8
FILLER_2_2	PIC X(1).	2	2	1
C2	PIC X(1).	3	3	1
FILLER_4_9	PIC X(6).	4	9	6
FILLER_10_11	PIC X(2).	10	11	2

Select a group array column

Consider what happens when a group array column (GRP2) is selected and passed as is. Four fillers are created: one for column A, one for columns C1 and C2, one for columns D1 through D3, and one for columns E and G. Since GRP2 is nested within GRP1, and GRP1 redefines column B, both column B and GRP1 are preserved.

The file layout is:

Table 153. File layout 1

Column	Picture Clause	Starting column	Ending column	Storage length
FILLER_1_1	PIC X(1).	1	1	1
B	PIC X(8).	2	9	8
GRP1	REDEFINES B.	2	9	8
FILLER_2_3	PIC X(2).	2	3	2
GRP2	OCCURS 2 TIMES.	4	9	6
FILLER_1	PIC X(3).			
FILLER_10_11	PIC X(2).	10	11	2

If the selected array column (GRP2) is flattened, both occurrences (GRP2 and GRP2_2) are loaded into the stage. The file layout is:

Table 154. File layout 2

Column	Picture Clause	Starting column	Ending column	Storage length
FILLER_1_1	PIC X(1).	1	1	1
B	PIC X(8).	2	9	8
GRP1	REDEFINES B.	2	9	8
FILLER_2_3	PIC X(2).	2	3	2
GRP2		4	6	3
FILLER_1	PIC X(3).	4	6	3
GRP2_2		7	9	3
FILLER_1_2	PIC X(3).	7	9	3
FILLER_10_11	PIC X(2).	10	11	2

Select an array element

Suppose an element (D1) of a group array is selected. If the array GRP2 is passed as is, fillers are created for column A, columns C1 and C2, columns D2 and D3, and columns E and G. Column B, GRP1, and GRP2 are preserved for the same reasons as before.

The file layout is:

Table 155. File layout

Column	Picture Clause	Starting column	Ending column	Storage length
FILLER_1_1	PIC X(1).	1	1	1
B	PIC X(8).	2	9	8
GRP1	REDEFINES B.	2	9	8
FILLER_2_3	PIC X(2).	2	3	2

Table 155. File layout (continued)

Column	Picture Clause	Starting column	Ending column	Storage length
GRP2	OCCURS 2 TIMES.	4	9	6
D1	PIC X(1).			1
FILLER_1	PIC X(2).			2
FILLER_10_11	PIC X(2).	10	11	2

Select a column that is redefined by another column

Consider what happens when column E is selected, which is redefined by another column. Two fillers are created: one for columns A through D3, and another for column G. Since column F redefines column E, it is dropped, though it will be available for expansion (see "Expanding fillers" on page 687).

The file layout is:

Table 156. File layout 1

Column	Picture Clause	Starting column	Ending column	Storage length
FILLER_1_9	PIC X(9).	1	9	9
E	PIC X(1).	10	10	1
FILLER_10_11	PIC X(1).	11	11	1

Now suppose the REDEFINE column (F) is selected. In this case column E is preserved since it is redefined by column F. One filler is created for columns A through D3, and another for column G. The file layout is:

Table 157. File layout 2

Column	Picture Clause	Starting column	Ending column	Storage length
FILLER_1_9	PIC X(9).	1	9	9
E	PIC X(1).	10	10	1
F	PIC 9(1) REDEFINES E.	10	10	1
FILLER_11_11	PIC X(1).	11	11	1

Select multiple redefine columns

This example describes how fillers are created for multiple redefine columns. In this case the same column is being redefined multiple times. The source table contains a column and a group that redefine column A, as well as two columns that redefine the group that redefines column A:

```
05   A              PIC X(100).
05   B              PIC X(11) REDEFINES A.
05   GRP1           REDEFINES A.
     10   C1        PIC X(10).
     10   C2        PIC 9(1).
05   D              PIC X(1) REDEFINES GRP1.
05   E              PIC 9(1) REDEFINES GRP1.
05   F              PIC X(1).
```

If columns C2 and E are selected, four fillers are created: one for column B, one for column C1, one for column D, and one for column F. Since an element of GRP1 is selected and GRP1 redefines column A, both column A and GRP1 are preserved. The first three fillers have the same start offset because they redefine the same storage area.

The file layout is:

Table 158. File layout

Column	Picture Clause	Starting column	Ending column	Storage length
A	PIC X(100).	1	100	100
FILLER_1_11	PIC X(11) REDEFINES A.	1	11	11
GRP1	REDEFINES A.	1	11	11
FILLER_1_10	PIC X(10).	1	10	10
C2	PIC 9(1).	11	11	1
FILLER_1_1	PIC X(1). REDEFINES GRP1.	1	1	1
E	PIC 9(1) REDEFINES GRP1.	1	1	1
FILLER_101_101	PIC X(1).	101	101	1

Select multiple cascading redefine columns

This example shows filler creation for multiple redefine columns, except this time they are cascading redefines instead of redefines of the same column. Consider the following source table, where column B redefines column A, GRP1 redefines column B, column D redefines GRP1, and column E redefines column D:

```
05   A              PIC X(100).
05   B              PIC X(11) REDEFINES A.
05   GRP1           REDEFINES B.
     10   C1        PIC X(10).
     10   C2        PIC 9(1).
05   D              PIC X(2) REDEFINES GRP1.
05   E              PIC 9(1) REDEFINES D.
05   F              PIC X(1).
```

If columns C2 and E are selected, this time only two fillers are created: one for column C1 and one for column F. Column A, column B, GRP1, and column D are preserved because they are redefined by other columns.

The file layout is:

Table 159. File layout

Column	Picture Clause	Starting column	Ending column	Storage length
A	PIC X(100).	1	100	100
B	PIC X(11) REDEFINES A.	1	11	11
GRP1	REDEFINES B.	1	11	11
FILLER_1_10	PIC X(10).	1	10	10
C2	PIC 9(1).	11	11	1
D	PIC X(2). REDEFINES GRP1.	1	2	2
E	PIC 9(1) REDEFINES D.	1	1	1
FILLER_101_101	PIC X(1).	101	101	1

Select an OCCURS DEPENDING ON column

The final example shows how an ODO column is handled. Suppose the source table has the following structure:

```
05   A              PIC X(1).
05   B              PIC X(8).
05   GRP1           REDEFINES B.
     10   C1        PIC X(1).
     10   C2        PIC 9(1).
     10   GRP2      OCCURS 0 TO 2 TIMES DEPENDING ON C2.
          15   D1   PIC X(1).
          15   D2   PIC X(1).
          15   D3   PIC X(1).
05   E              PIC X(1).
05   F              PIC 9(1) REDEFINES E.
05   G              PIC X(1).
```

If column B is selected, four fillers are created. The file layout is:

Table 160. File layout

Column	Picture Clause	Starting column	Ending column	Storage length
FILLER_1_1	PIC X(1).	1	1	1
B	PIC X(8).	2	9	8
GRP1	REDEFINES B.	2	9	8
FILLER_2_2	PIC X(1).	2	2	1
C2	PIC 9(1).	3	3	1
GRP2	OCCURS 0 TO 2 TIMES DEPENDING ON C2.	4	9	6
FILLER_1	PIC X(3).			3
FILLER_2	PIC X(2).	10	11	2

Fillers are created for column A, column C1, columns D1 through D3, and columns E and G. GRP1 is preserved because it redefines column B. Since GRP2 (an ODO column) depends on column C2, column C2 is preserved. GRP2 is preserved because it is an ODO column.

Expanding fillers

After you select columns to load into a Complex Flat File stage, the selected columns and fillers appear on the Records tab on the Stage page. If you need to reselect any columns that are represented by fillers, it is not necessary to reload your table definition. An **Expand filler** option allows you to reselect any or all of the columns from a given filler.

To expand a filler, right-click the filler column in the columns tree and select **Expand filler** from the pop-up menu. The Expand Filler window opens. The contents of the given filler are displayed in the **Available columns** tree, allowing you to reselect the columns that you need.

Suppose the Records tab contains the following columns:

```
FILLER_1_1
B
GRP1
FILLER_2_9
FILLER_10_11
```

If you expand FILLER_2_9, the **Available columns** tree in the Expand Filler window displays the contents of FILLER_2_9:

```
C1
C2
GRP2(2)
    D1
    D2
    D3
```

If you select column C1, the Records tab changes to this:

```
FILLER_1_1
B
GRP1
C1
FILLER_3_9
FILLER_10_11
```

If you expand FILLER_3_9 and select column C2, the Records tab now changes to this:

```
FILLER_1_1
B
GRP1
C1
C2
FILLER_4_9
FILLER_10_11
```

If you continue to expand the fillers, eventually the Records tab will contain all of the original columns in the table:

```
A
B
GRP1
C1
C2
GRP2
D1
D2
D3
E
F
G
```

Product accessibility

You can get information about the accessibility status of IBM products.

The IBM InfoSphere Information Server product modules and user interfaces are not fully accessible. The installation program installs the following product modules and components:

- IBM InfoSphere Business Glossary
- IBM InfoSphere Business Glossary Anywhere
- IBM InfoSphere DataStage
- IBM InfoSphere FastTrack
- IBM InfoSphere Information Analyzer
- IBM InfoSphere Information Services Director
- IBM InfoSphere Metadata Workbench
- IBM InfoSphere QualityStage

For information about the accessibility status of IBM products, see the IBM product accessibility information at http://www.ibm.com/able/product_accessibility/index.html.

Accessible documentation

Accessible documentation for InfoSphere 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.

IBM and accessibility

See the IBM Human Ability and Accessibility Center for more information about the commitment that IBM has to accessibility.

Accessing product documentation

Documentation is provided in a variety of locations and formats, including in help that is opened directly from the product client interfaces, in a suite-wide information center, and in PDF file books.

The information center is installed as a common service with IBM InfoSphere Information Server. The information center contains help for most of the product interfaces, as well as complete documentation for all the product modules in the suite. You can open the information center from the installed product or from a Web browser.

Accessing the information center

You can use the following methods to open the installed information center.

- Click the **Help** link in the upper right of the client interface.

  Note: From IBM InfoSphere FastTrack and IBM InfoSphere Information Server Manager, the main Help item opens a local help system. Choose **Help > Open Info Center** to open the full suite information center.
- Press the F1 key. The F1 key typically opens the topic that describes the current context of the client interface.

  Note: The F1 key does not work in Web clients.
- Use a Web browser to access the installed information center even when you are not logged in to the product. Enter the following address in a Web browser: http://host_name:port_number/infocenter/topic/com.ibm.swg.im.iis.productization.iisinfsv.home.doc/ic-homepage.html. The host_name is the name of the services tier computer where the information center is installed, and port_number is the port number for InfoSphere Information Server. The default port number is 9080. For example, on a Microsoft® Windows® Server computer named iisdocs2, the Web address is in the following format: http://iisdocs2:9080/infocenter/topic/com.ibm.swg.im.iis.productization.iisinfsv.nav.doc/dochome/iisinfsrv_home.html.

A subset of the information center is also available on the IBM Web site and periodically refreshed at http://publib.boulder.ibm.com/infocenter/iisinfsv/v8r7/index.jsp.

Obtaining PDF and hardcopy documentation

- A subset of the PDF file books are available through the InfoSphere Information Server software installer and the distribution media. The other PDF file books are available online and can be accessed from this support document: https://www.ibm.com/support/docview.wss?uid=swg27008803&wv=1.
- 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 http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss.

Providing feedback about the documentation

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

Links to non-IBM Web sites

This information center may provide links or references to non-IBM Web sites and resources.

IBM makes no representations, warranties, or other commitments whatsoever about any non-IBM Web sites or third-party resources (including any Lenovo Web site) that may be referenced, accessible from, or linked to any IBM site. A link to a non-IBM Web site does not mean that IBM endorses the content or use of such Web site or its owner. In addition, IBM is not a party to or responsible for any transactions you may enter into with third parties, even if you learn of such parties (or use a link to such parties) from an IBM site. Accordingly, you acknowledge and agree that IBM is not responsible for the availability of such external sites or resources, and is not responsible or liable for any content, services, products or other materials on or available from those sites or resources.

When you access a non-IBM Web site, even one that may contain the IBM-logo, please understand that it is independent from IBM, and that IBM does not control the content on that Web site. It is up to you to take precautions to protect yourself from viruses, worms, trojan horses, and other potentially destructive programs, and to protect your information as you deem appropriate.

Notices and trademarks

This information was developed for products and services offered in the U.S.A.

Notices

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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample 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, the IBM logo, and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:

Adobe is a registered trademark 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 and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The United States Postal Service owns the following trademarks: CASS, CASS Certified, DPV, LACS^{Link}, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACS^{Link} licensee of the United States Postal Service.

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

Contacting IBM

You can contact IBM for customer support, software services, product information, and general information. You also can provide feedback to IBM about products and documentation.

The following table lists resources for customer support, software services, training, and product and solutions information.

Table 161. IBM resources

Resource	Description and location
IBM Support Portal	You can customize support information by choosing the products and the topics that interest you at www.ibm.com/support/entry/portal/Software/Information_Management/InfoSphere_Information_Server
Software services	You can find information about software, IT, and business consulting services, on the solutions site at www.ibm.com/businesssolutions/
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/
Training and certification	You can learn about technical training and education services designed for individuals, companies, and public organizations to acquire, maintain, and optimize their IT skills at http://www.ibm.com/software/sw-training/
IBM representatives	You can contact an IBM representative to learn about solutions at www.ibm.com/connect/ibm/us/en/

Providing feedback

The following table describes how to provide feedback to IBM about products and product documentation.

Table 162. Providing feedback to IBM

Type of feedback	Action
Product feedback	You can provide general product feedback through the Consumability Survey at www.ibm.com/software/data/info/consumability-survey
Documentation feedback	To comment on the information center, click the Feedback link on the top right side of any topic in the information center. You can also send 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

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Y

Printed in USA

SC19-3459-00

Spine information:

IBM InfoSphere DataStage and QualityStage **Version 8 Release 7** **Parallel Job Developer's Guide**