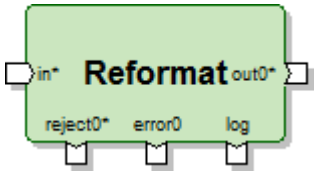
REFORMAT



Purpose

REFORMAT changes the format of records by dropping fields, or by using DML expressions to add fields, combine fields, or transform the data in the records.

REFORMAT performs an implicit reformat when you do not define a **reformat** function. For more information, see "Implicit reformat" in the *Graph Developer's Guide*.

Foldable

If component folding is enabled, the Co>Operating System folds this component by default. For more information, see "Component folding" in the *Graph Developer's Guide*.

Location in the Organizer

Transform folder

Training

The *Graph Development Basics* training module includes the following lessons about the REFORMAT component:

- *Modifying data*
- *Copying, combining, and splitting flows*
- *Graph design*
- *Redefining data*

NOTE: To open *Graph Development Basics*, choose **Help > Training > Graph Development Basics** from the GDE menu bar. If *Graph Development Basics* is not available at your site, contact Ab Initio Support for assistance.

REFORMAT topics

Alphabetical list of parameters for REFORMAT

Parameters for REFORMAT

count
(integer, required)

Sets the number of:

- **out** ports
- **reject** ports
- **error** ports
- **transform** parameters

Default is **1**.

transform*n*
(filename or string, optional)

The name of the file containing the types and transform functions, or a transform string. For more information, see "Transform package for REFORMAT".

select
(expression, optional)

Filter for records before reformatting.

error_group
(string, optional)

Name of the error group to which this component belongs. It sends its error output to the HANDLE ERRORS component with a matching **error_group** value.

log_group
(string, optional)

Name of the log group to which this component belongs. It sends its log output to the HANDLE LOGS component with a matching **log_group** value.

reject-threshold
(choice, required)

Specifies the component's tolerance for reject events.

For more information, see "Component tolerance for rejections" in the *Graph Developer's Guide*.

limit
(integer, required)

A number representing reject events.

When the **reject-threshold** parameter is set to **Use limit/ramp**, the component uses the values of the **ramp** and **limit** parameters in a formula to determine the component's tolerance for reject events.

Default is **0**.

For more information, see "Setting limit and ramp for reject events" in the *Graph Developer's Guide*.

ramp
(real, required)

Rate of toleration of reject events in the number of records processed.

When the **reject-threshold** parameter is set to **Use limit/ramp**, the component uses the values of the **ramp** and **limit** parameters in a formula to determine the component's tolerance for reject events.

Default is **0.0**.

For more information, see "Setting limit and ramp for reject events" in the *Graph Developer's Guide*.

output-index
(filename or string, optional)

Specifies either the name of a file containing a transform function, or a transform string. The component calls the specified transform function for each input record. The transform function should return an index value between **0** and the highest-numbered output port. REFORMAT uses this value to direct the input record to the output port that has the same number as the value, and executes the transform function, if any, associated with that port.

When you specify a value for this parameter, each input record goes to exactly one transform-output port pair. For example, suppose there are 100 input records and two output ports. Each output port receives between 0 and 100 records. According to the transform function you specify for **output-index**, the split can be 50/50, 60/40, 0/100, 99/1, or any other combination that adds up to 100.

If an index is out of range (less than zero or greater than the highest-numbered port), the component discards the input record. If the **output_index** function returns an error, the input record is rejected to the **reject0** port (where it is not distinguishable from a record rejected by the **transform0** function).

NOTE: If you specify a value for the **output-index** parameter, you cannot also specify the **output-indexes** parameter. Use **output-index** when you want to direct a single record to a single transform-output port; use **output-indexes** to direct a single record to multiple transform-output ports.

If you do not specify a value for either **output-index** or **output-indexes**, the component sends every input record to every transform-output port pair. For example, if the component has two output ports and there are no rejects, 100 input records results in 100 output records on each port for a total of 200 output records.

output-indexes
(filename or string, optional)

Specifies either the name of a file containing a transform function, or a transform string. The component calls the specified transform function for each input record. The transform function uses the value of the input record to direct that input record to particular transform-output ports.

The expected output of the transform function is a vector of numeric values. The component considers each element of this vector as an index into the output transforms and ports. The component directs the input record to the identified output ports and executes the transform functions, if any, associated with those ports.

If an index is out of range (less than zero or greater than the highest-numbered port), the component discards the input record. The component ignores duplicate **in** port numbers. If a port appears multiple times in the vector, the component uses the corresponding transform-port only once.

If the **output_indexes** function returns an error, the input record is rejected to the **reject0** port (where it is not distinguishable from a record rejected by the **transform0** function).

NOTE: If you specify a value for the **output-indexes** parameter, you cannot also specify the **output-index** parameter. Use **output-index** when you want to direct a single record to a single transform-output port; use **output-indexes** to direct a single record to multiple transform-output ports.

If you do not specify a value for either **output-indexes** or **output-index**, the component sends every input record to every transform-output port pair. For example, if the component has two output ports and there are no rejects, 100 input records results in 100 output records on each port for a total of 200 output records.

As an example of how the component uses the transform function specified in the **output-indexes** parameter, consider the following:

```
out :: output_indexes(in) =
begin
  out :1: if (in.kind == "a") [vector 0, 1, 2];
```

```
    out :2: if (in.kind == "b") [vector 2, 3, 4];
    out : : [vector 5];
  end;
```

When you specify this transform function for the **output-indexes** parameter, it directs the input record to the transform functions for:

- Ports 0, 1, and 2 if the field **in.kind** is **"a"**
- Ports 2, 3, and 4 if the field **in.kind** is **"b"**
- Port 5 if otherwise

logging
(boolean, optional)

Specifies whether the component logs certain events.

For more information, see "Using logging in components" in the *Graph Developer's Guide*.

log_input
(choice, optional)

Specifies how often the component sends an input record to its **log** port. The **logging** parameter must be set to **True** for this parameter to be available.

For example, if you select **100**, the component sends every 100th input record to its **log** port.

For more information, see "Using logging in components" in the *Graph Developer's Guide*.

log_output
(choice, optional)

Specifies how often the component sends an output record to its **log** port. The **logging** parameter must be set to **True** for this parameter to be available.

For example, if you select **100**, the component sends every 100th output record to its **log** port.

For more information, see "Using logging in components" in the *Graph Developer's Guide*.

log_reject
(choice, optional)

Specifies how often the component sends a reject record to its **log** port. The **logging** parameter must be set to **True** for this parameter to be available.

For example, if you select **100**, the component sends every 100th reject record to its **log** port.

For more information, see "Using logging in components" in the *Graph Developer's Guide*.

Transform package for REFORMAT

REFORMAT has an optional transform package whose functions transform fields or records, enable or disable record output, and support error and log handling.

The number of transforms (that is, the number of **transform***n* parameters) is determined by the value of the count parameter. Transforms are numbered from 0, as are ports. Therefore, the number *n* identifies the output, reject, and error ports that correspond to the transform. For example, **transform0** provides the error-handling functions for the port **out0**; errors and rejected records for this port are sent to **error0** and **reject0**.

In addition to the **reformat** function, the following functions are built into the package.

NOTE: If you define any of the following optional functions in a package, you must also define a **reformat** function.

output_at_event

This optional function allows you to perform additional processing when an input port receives a computepoint, checkpoint, end of unit of work, or shutdown event. It also allows for a record to be output as part of that additional processing, if you want. The function can be written to perform a specific behavior *without* writing a record to the **out***n* port, to write records to the **out***n* port, or a combination of both actions. The function is connected to the **out** port number that is associated with its containing package. Therefore, you can use it in different ways for different ports.

The syntax for the **output_at_event** transform function is as follows:

`out::output_at_event(`*event_info*`)`

To use this function, you must define the **event_info** argument. You can do this by including **~ab_home/include/event-info-type.dml** in your package. This file also defines integer constants that your DML code can test for and indicate the type of event that caused it to be evaluated. The **event-info-type.dml** file resembles this:

```
constant int COMPUTEPOINT = 0;
constant int CHECKPOINT = 1;
constant int END_OF_UOW = 2; // end of unit of work
constant int SHUTDOWN = 3;

type event_info_t = record
  int event_type;
end;
```

There are two basic uses for this function: one performs an action but does not generate output records, the other does generate output records. However, these uses are not mutually exclusive. You control whether the function writes to the output port by setting **out** to **NULL**; doing so disables writing to **out**.

The **output_at_event** function is called when an event is delivered, as follows:

- **COMPUTEPOINT** and **CHECKPOINT** are delivered upon a computepoint or checkpoint, respectively, arriving at the REFORMAT component's **in** port.
- **END_OF_UOW** is delivered to a REFORMAT component that is in a transactional group when the group's **BEGIN TRANSACTION** component commits a transaction, as specified by the value of its transaction_boundary parameter.
- **SHUTDOWN** is delivered after the last record has been processed, just before the graph shuts down.

Examples

The following transform example outputs an extra record containing status (**out.rec_type**), record count (**out.record_count**), and date and time information (**out.file_date**) when a **SHUTDOWN** event is received:

```
include "~ab_home/include/event-info-type.dml";

/* Count the number of records processed */
let int g_count = 0;

/*Reformat operation*/
out :: reformat(in)=
begin
  g_count = g_count + 1;
  out.rec_type :: "B";
  out.* :: in.*;
end;

/* This function is optional. */
/* Function to allow the output of an additional record at event */
out :: output_at_event(event_info)=
begin
  out.rec_type :: if( event_info.event_type == SHUTDOWN ) "T";
  out.record_count :: if( event_info.event_type == SHUTDOWN ) g_count;
  out.file_date :: if( event_info.event_type == SHUTDOWN ) today();
end;
```

NOTE: This use of the **output_at_event** function intentionally violates the REFORMAT component's principle of one output record for each input record because it creates additional output records.

The following transform example writes to a log when a **CHECKPOINT** event is received and sends NULL to **out**, which suppresses writing an additional output record.

```
include "~ab_home/include/event-info-type.dml";

constant string("\0") AI_SERIAL parameter;

let string("")[int] log_buffer = [vector ];

/*Reformat operation*/
out :: reformat(in)=
begin
  log_buffer = vector_append(log_buffer, (string("\0"))in.id + " " + in.description);
  out.* :: in.*;
end;
```

```
/* This function is optional. */
/* Function to allow the output of an additional record at event */
out :: output_at_event(event_info)=
begin
  if( event_info.event_type == CHECKPOINT )
  begin
    write_to_log_file(AI_SERIAL + "/desc_result.txt", string_representation(log_buffer));
    log_buffer = [vector ];
  end
  out :: NULL;  // no record is sent to the out port
end;
```

Error-handling functions

You can use a package to define error-handling and log-handling functions that customize how the component handles errors, log messages, or rejected input records. This component supports the **output_for_error**, **make_error**, **log_error**, and **final_log_output** functions. Note that each **transform***n* requires its own error-handling functions.

For more information, see "Functions for handling errors, logs, and rejects" in the *Graph Developer's Guide*.

Runtime behavior of REFORMAT

The *n* in **out***n* gives each **out** port a unique number. Each **out***n* port has a corresponding **reject***n* and **error***n* port. REFORMAT does the following:

1. REFORMAT reads a record from the **in** port.
2. If the **select** parameter has an expression specified, REFORMAT uses the expression to evaluate the input record:
   - If the expression evaluates to false (**0**), REFORMAT discards the input record and starts over with step 1.
   - If the expression produces NULL, REFORMAT writes a descriptive error message and stops execution of the graph.
   - If the expression evaluates to true (anything other than **0** or NULL), REFORMAT begins processing the input record.
3. If the **select** parameter does not have a value, REFORMAT begins processing the input record.
4. REFORMAT determines whether a transform function is specified in either the output-index or output-indexes parameter:
   - If neither **output-index** nor **output-indexes** has a value (the usual case when there is only one **out** port), REFORMAT sends the input record to every transform-out port pair, beginning with **out0** and progressing sequentially.
   - If **output-index** or **output-indexes** has a value, REFORMAT evaluates the specified index transform. If **output-indexes** is defined, it should return a vector of port index values. If **output-index** is defined, it should return a single port index value.

     REFORMAT uses one or more values from the index transform to determine the appropriate transform-output port pair or pairs for the input record. If the index transform returns more than one value, REFORMAT sends the record to each of the appropriate ports, starting with the lowest numbered port and progressing to the other ports sequentially.

     If the index transform returns an error, REFORMAT rejects the input record to the **reject0** port (where it is indistinguishable from records rejected by **transform 0**).

     NOTE: The value of **output-index** or **output-indexes** parameter does not affect the behavior of the output_at_event function, if defined. Rather, the function is called for every event received. The function's output port is identified by the output number that is associated with its containing transform package.
5. REFORMAT determines whether each **out***n* port has a transform function.
   - If an **out** port does not have a transform function, REFORMAT uses implicit reformat to process the input record. For more information, see "Implicit reformat".
   - If the input record is sent to more than one port, the order of the transform evaluation is sequential: it calls the transform function on each port in order, starting with the lowest numbered port. For example, if the record is to be sent to **port0** and **port2**, it is sent to **port0** first, and then to **port2**. The evaluation of the second transform can depend on the side-effects of the first transform, which means you could make successive calls to a function like next_in_sequence from sequential transforms for the same input record.

   If a transform function results in an error or returns NULL, REFORMAT writes the following:
   - An error message to the corresponding **error** port

- The current input record to the corresponding **reject** port

The component stops execution of the graph when the number of reject events exceeds the reject threshold. For more information, see "Setting limit and ramp for reject events" in the *Graph Developer's Guide*.

If the **reject** or **error** ports do not have flows attached to them, REFORMAT discards the record.

6. REFORMAT writes the record to the **out** port of each successful transform, and then begins processing the next input record.

NOTE: In addition to receiving records, REFORMAT also receives certain events on its **in** port. These events cause the **output_at_event** function, if it is defined, to be called. For more information, see "output_at_event".

Examples of using REFORMAT

See the *Cook>Book* topics that describe the following graphs:

- calculating-distances.mp
- difference.mp
- ht_process1.mp
- ht_validate2.mp
- make_difference_metadata.mp
- name-parsing.mp
- nearest-branch-quadtree.mp

In addition, run the following graphs in the **dml** sandbox. (See "Setting up the DML sandbox".)

- **overview.mp**
- **business_tasks/cleanse_data.mp**
- **business_tasks/get_month_end_date.mp**
- **business_tasks/process_summary.mp**
- **transforms/reformat/reformat.mp**
- **transforms/lookups/lookup.mp**
- **transforms/lookups/interval-lookup.mp**
- **transforms/lookups/regex-lookup.mp**

NOTE: Most of the DML predefined named type examples use a simple REFORMAT. See **/predefined_named_types** in the **dml** sandbox.

REFORMAT versus REDEFINE FORMAT

REFORMAT can change the record format of data records by dropping fields, or by using DML expressions to add fields, combine fields, or transform the data in the records.

REDEFINE FORMAT copies data records from its input to its output without changing the values in the data records. You use REDEFINE FORMAT to change or rename fields in a record format without changing the values in the records. In this way, REDEFINE FORMAT is similar to the DML built-in function reinterpret_as. Typically, REDEFINE FORMAT has different DML on its input and output ports, and enables the unmodified data to be interpreted in a different form.

Related topics

- HANDLE ERRORS (for details on handling error output)
- HANDLE LOGS (for details on handling log output)

Copyright © 1997-2018, Ab Initio, Confidential and Proprietary. All rights reserved.

Top of page