



# Data Integrator: Lookups and Joins



© 2010 Dimensional Insight, Inc.

## **Data Integrator**

*TR-INLJ-042010-01*

# DI Training Series

## Data Integrator

### Lookups and Joins

#### **Description**

This is a supplemental seminar in the DI Training Series. This class is designed specifically to explore the uses and options of the Data Integrator Lookup and Join Process Objects.

#### **Who should attend**

This seminar is primarily intended for the IS professional as well as those developers responsible for the creation and maintenance of data within The Diver Solution suite.

#### **Prerequisites**

Students are expected to be familiar with the basic syntax of Data Integrator.

For more information about training, call (920) 436-8299, or visit our web site at <http://www.dimins.com>. On-site training can be arranged.

Revision: 1.0

Part Number: *TR-INLJ-042010-01*

Product Version: 2.2

©Copyright 2010. Dimensional Insight Inc. All rights reserved. Duplication of this course material is strictly prohibited without the expressed written consent of Dimensional Insight Inc. All other products and brand names are trademarks and registered trademarks of their respective companies.



## Data Integrator Process Objects

- Data Integrator contains a rich set of Process Objects. Three among them, are used to join data flows:
  - Concat
  - Lookup
  - Join



## Data Integrator Process Objects

- Which Process Object to use?
- Determine by:
  - Knowing your options
  - Running test scripts against known data and comparing results
  - Applying to the “real” data



This session will do this – present the options and allow you to compare results.

Let's start with Concat...



# Concat

The Concat Process Object will concatenate multiple Input flows together; the Output column set is the union of all the Input columns. If a column is requested in the Output flow that does not appear in all the Input flows, the value of that Output column will be null in the appropriate rows.

Data Flow A		Data Flow B	
ProdID	Sales	Product	Name
1	100	1	Product A
3	100	2	Product B
4	100	3	Product C
6	100	4	Product D
		5	Product E

A CONCAT of Data Flows A and B			
ProdID	Sales	Product	Name
1	100	null	null
3	100	null	null
4	100	null	null
6	100	null	null
null	null	1	Product A
null	null	2	Product B
null	null	3	Product C
null	null	4	Product D
null	null	5	Product E



## Concat

- Appends one or more flows to another
- The resulting number of rows is always the sum of the input
- Rows with like key values are not merged
- Fields with like column names are aligned
- Lookups and Joins merge rows by linking key values establishing a new relationship between fields



© 2010. Dimensional Insight, Inc.

In a Concat, one could say the relationship between the fields is already established – it's either there or not.

Lookup and Join can create new relationships.

So let's take a look at those... starting with the Lookup Process Object.



## Lookup Process Object

- Merges two Input flows to form one output flow
- All rows in the Input flow are kept
- Order of the data flows (left/right) matters
- Compared to SQL, Lookup join types are "outer joins"
- Do not apply SQL Join terminology to the Data Integrator Lookup join types!



## Lookup Process Object

- The two flows are the Input flow (left side) and the Lookup flow (right side)
- A single or multiple columns from each flow are used as the key to match rows between the flows
- For each row in the Input flow, the value of the match column(s) is used to find a row in the Lookup flow
- The match is NOT case sensitive unless specified
- The data in the Lookup flow must fit into memory
- Five types: inner, outer, multiply, update, outer\_update



### Process Object – LOOKUP

The Lookup Process Object combines two Input flows together in a simple relational join to form a single Output flow. The two flows are the Input flow and the Lookup flow. A single or multiple columns from each flow are used to match rows between the flows. For each row in the Input flow, the value of the match column(s) is used to find a row in the Lookup flow. If no row is found, empty strings are inserted for those columns. The search is case-insensitive unless the **case\_sensitive** attribute is set to “true”. The Lookup flow must fit in memory.



## Lookup Process Object

```
object 'PROC' " ____ " {  
  process_type      = "Lookup",  
  join_type        = "____",  
  inputs           = {"____", "____"},  
  joins            = {"____", "____"},  
  multijoins       = {  
    { "____", "____" },  
    { "____", "____" },  
  },  
  case_sensitive   = "____",  
  update_null_value = "____",  
};
```



### Process Object – LOOKUP (continued)

#### **outer\_update**

Similar to the **update** join type, except that any non-matched rows in the Lookup flow will be added to the Output, with empty strings inserted for columns that only appear in the input flow.

#### **update\_null\_value**

This attribute defines the null value used for the **update** and **outer\_update** join types. If a Lookup column value matches this string, then its value is considered to be null, and does not affect the Output. This attribute defaults to "" (empty string).

#### **join\_type**

Defines whether or not the Lookup will be an **inner** join, **outer** join, **multiply**, **update**, or **outer\_update**.

#### **inputs**

Defines the Input flow (left side) and Lookup flow (right side).

#### **joins**

Used for a single column match. Defines the columns that must match for the Lookup. The first column name is a column in the Input flow; the second column name is a column in the Lookup flow.

#### **multijoins**

Used for a multicolumn match. Each element of the array is a two-element array (pair) of column names. The first element of the pair is a column name in the data (first) input. The second element of the pair is a corresponding lookup (second) input.

#### **case\_sensitive**

Controls whether or not comparisons are case sensitive or case insensitive. Valid entries are "true" or "false". If the attribute is not given, comparisons will be case insensitive.

## // sample\_lookup\_01.int

```
object 'INPT' "input-flow" {
  input_type = "Filein",
  file_type = "column_headers",
  filename = "../data/lookup_input_01.dat",
};

object 'INPT' "lookup-flow" {
  input_type = "Filein",
  file_type = "column_headers",
  filename = "../data/lookup_lookup_01.dat",
};

object 'PROC' "lookup_inner_proc" {
  process_type="Lookup",
  join_type = "INNER",
  inputs = {"input-flow", "lookup-flow"},
  joins = {"Cust ID", "Cust ID"},
};

object 'OUTP' "out" {
  output_type = "Fileout",
  file_type = "column_headers",
  input = "lookup_inner_proc",
  filename = "../temp/lookup_file_01.tmp",
};
```

## Input Flow

Invoice	Cust ID	Product	Amount
000001	100002	Coke	1,000
000002	100003	Coke	2,000
000003	100004	Pepsi	2,200
000003	100004	Coke	3,000
000004	100005	Coke	6,000

## Lookup Flow

Cust ID	Customer	Location
100001	ABC Company	Boston
100002	Bertuccis	New York
100002	Bertuccis	Miami
100004	Peters Pizza	San Diego
100005	Pizzeria Uno	Chicago

## Output File

Invoice	Cust ID	Product	Amount	Customer	Location
000001	100002	Coke	1,000	Bertuccis	New York
000002	100003	Coke	2,000		
000003	100004	Pepsi	2,200	Peters Pizza	San Diego
000003	100004	Coke	3,000	Peters Pizza	San Diego
000004	100005	Coke	6,000	Pizzeria Uno	Chicago

```
C:\DI_Training\Programs>integ sample_lookup_01.int
DI-Atlantis Data Integrator v2.1 (32)
Copyright (C) 1991-2004 by Dimensional Insight, Inc

Sun May 22 13:19:12 2005 Starting task sample_lookup_01...
Reading ../data/lookup_input_01.dat...
Reading ../data/lookup_lookup_01.dat...
Warning: Duplicate value "100002" found in join column on row 2 for Lookup object "lookup_inner_proc"
5 records written to file ../temp/lookup_file_01.tmp.
Sun May 22 13:19:13 2005 Finished task sample_lookup_01. (elapsed time 0:00:01)

C:\DI_Training\Programs>
```

## Lookup – Example 1

The **sample\_lookup\_01.int** script contains an **INNER** lookup that joins the Input flow with the Lookup flow based on the column **Cust ID**. The row count in the Output file (5) matches the row count in the Input flow. There are two rows in the Lookup flow for **Cust ID 100002**. One has a **Location** of *New York* and the second has a **Location** of *Miami*. As a result, the following warning message is displayed when the script is executed:

“Warning: Duplicate value "100002" found in join column on row 2 for Lookup object "lookup\_inner\_proc".”

The **Location** column from the first occurrence of **Cust ID 100002** in the Lookup flow (*New York*) will be added to the records in the Output flow.

Invoice 000002 has a **Cust ID** of 100003 but this **Cust ID** is NOT in the Lookup flow. As a result, this row will contain blank columns for **Customer** and **Location** in the Output file.

The Lookup flow contains a **Cust ID** (100001) that is not in the Input flow. Since this is an **inner** lookup, **Cust ID 100001** will NOT appear in the Output file.



## Lookup Process Object – INNER

- Most common join type (default)
- Merge code or IDs in a transaction flow with their user-friendly descriptors

```
// sample_lookup_02.int

object 'INPT' "input-flow" {
  input_type = "Filein",
  file_type = "column_headers",
  filename = "../data/lookup_input_01.dat",
};

object 'INPT' "lookup-flow" {
  input_type = "Filein",
  file_type = "column_headers",
  filename = "../data/lookup_lookup_01.dat",
  aliases = {"Cust ID=Cust LU"},
};

object 'PROC' "lookup_outer_proc" {
  process_type="Lookup",
  join_type = "OUTER",
  inputs = {"input-flow", "lookup-flow"},
  joins = {"Cust ID", "Cust LU"},
};

object 'OUTP' "out" {
  output_type = "Fileout",
  file_type = "column_headers",
  input = "lookup_outer_proc",
  filename = "../temp/lookup_file_02.tmp",
};
```

**Input Flow**

Invoice	Cust ID	Product	Amount
000001	100002	Coke	1,000
000002	100003	Coke	2,000
000003	100004	Pepsi	2,200
000003	100004	Coke	3,000
000004	100005	Coke	6,000

**Lookup Flow**

Cust ID	Customer	Location
100001	ABC Company	Boston
100002	Bertuccis	New York
100002	Bertuccis	Miami
100004	Peters Pizza	San Diego
100005	Pizzeria Uno	Chicago

**Output File**

Invoice	Cust ID	Product	Amount	Cust LU	Customer	Location
000001	100002	Coke	1,000	100002	Bertuccis	New York
000002	100003	Coke	2,000			
000003	100004	Pepsi	2,200	100004	Peters Pizza	San Diego
000003	100004	Coke	3,000	100004	Peters Pizza	San Diego
000004	100005	Coke	6,000	100005	Pizzeria Uno	Chicago
				100001	ABC Company	Boston

## Lookup – Example 2

The `sample_lookup_02.int` script contains the script from the previous example with the following changes. The `join_type` attribute has been changed to **OUTER**, the **Cust ID** column in the Lookup flow has been aliased to **Cust LU**, and the `joins` column for the Lookup flow has been changes to **Cust LU**.

Notice how the Output file has changed. Since the **Cust ID** column in the Lookup flow was aliased to **Cust LU**, it now appears in the Output file.

An extra record has also been written to the Output file. By definition, an **OUTER** lookup will add any unmatched rows in the Lookup flow to the Output file. As we remember from Example 1, the Lookup flow contains a row for **Cust ID 100001** that is not in the Input flow. A row is added to the Output file for this row in the Lookup flow with the **Invoice** and **Cust ID** columns set to blank (see last row).

When you use an **OUTER** lookup, it is recommended that the column names used in the `joins` or `multijoins` attributes NOT be identical. In our example, we used the `aliases` attribute to rename the **Cust ID** column in the Lookup flow, so it does not match the column name in the Input flow. If column names are identical, rows that exist in the Lookup flow and do not exist in the Input flow will have blank values in the columns that have identical names. The graphic below displays how the Output appears if we do NOT alias the column. Notice how the **Cust ID** column in the last row is blank and we can no longer identify it as being associated with **Cust ID 100001**.

Invoice	Cust ID	Customer	Location
000001	100002	Bertuccis	New York
000002	100002	Bertuccis	New York
000003	100003		
000004	100004	Peters Pizza	San Diego
000005	100005	Pizzeria Uno	Chicago
		ABC Company	Boston



## Lookup Process Object – OUTER

- Assures that the Output file has at least one record for all key values
- For a transaction flow, can show the absence of a transaction against a master file of all possible transactions

```
// sample_lookup_04.int

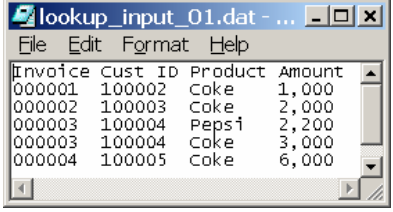
object 'INPT' "input-flow" {
  input_type = "Filein",
  file_type = "column_headers",
  filename = "../data/lookup_input_01.dat",
};

object 'INPT' "lookup-flow" {
  input_type = "Filein",
  file_type = "column_headers",
  filename = "../data/lookup_lookup_01.dat",
  aliases = {"Cust ID=Cust LU"},
};

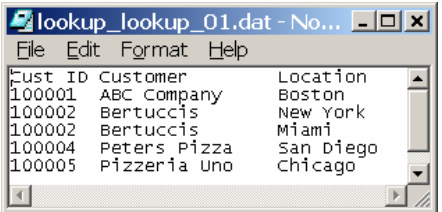
object 'PROC' "lookup_multiply_proc" {
  process_type="Lookup",
  join_type = "MULTIPLY",
  inputs = {"input-flow", "lookup-flow"},
};

object 'OUTP' "out" {
  output_type = "Fileout",
  file_type = "column_headers",
  input = "lookup_multiply_proc",
  filename = "../temp/lookup_file_04.tmp",
};
```

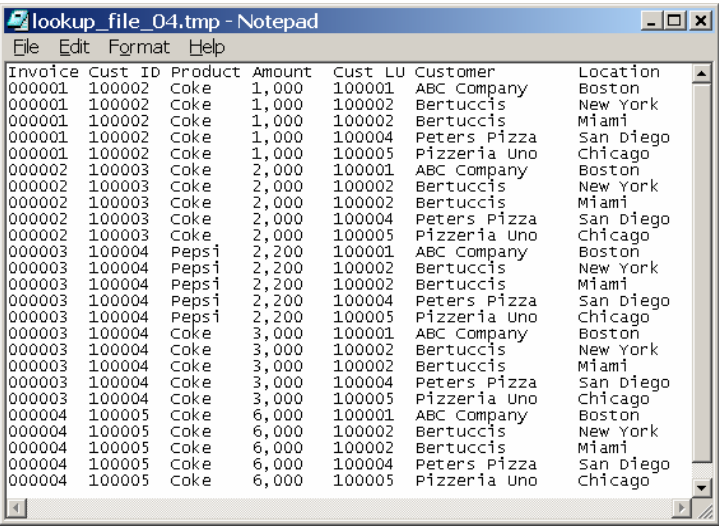
**Input Flow**



**Lookup Flow**



**Output File**



### Lookup – Example 4

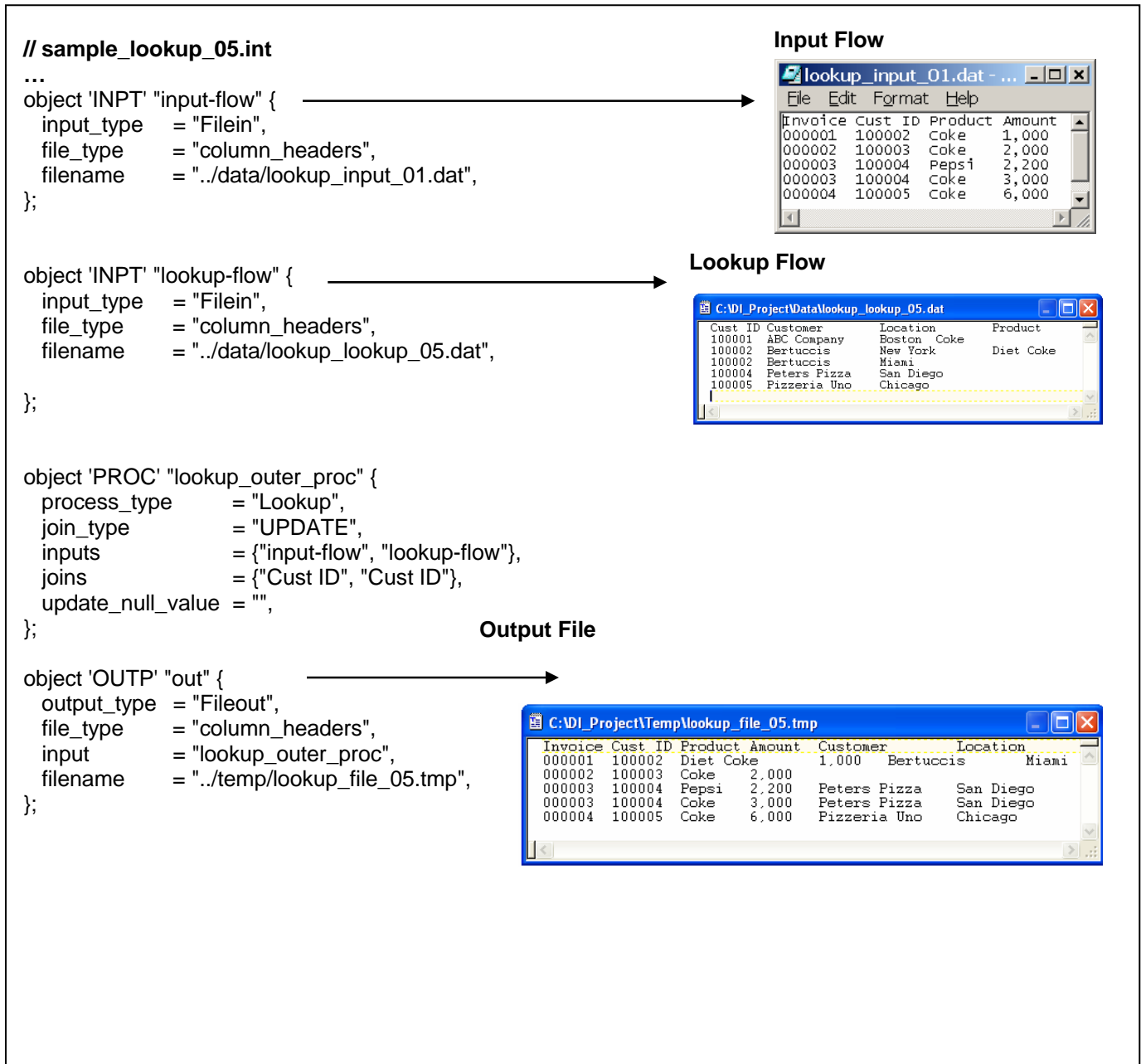
The **sample\_lookup\_04.int** script contains the script from Example 2 with the following changes. The **join\_type** attribute has been changed to **MULTIPLY** and the **joins** attribute has been removed.

The Output file now contains a Cartesian product of the Input and Lookup flows. For each of the five records in the Input flow, five records are written to the Output flow (one for each of the five records in the Lookup flow). As a result, 25 rows are written to the Output file. Notice how the **Amounts** for each of our **Invoice** lines is now counted five times. Be careful when you use the **multiply** join type!



## Lookup Process Object – MULTIPLY

- Produces an Output file of all possible combinations
- Sometimes used to join a single record as a new column in a data flow



### Lookup – Example 5

The **sample\_lookup\_05.int** script contains the script from Example 1, an **INNER** lookup, with the following change: the **join\_type** attribute has been changed to **UPDATE**. (The **join\_type** attribute **update\_null\_value** is displayed with its default for illustration only.)

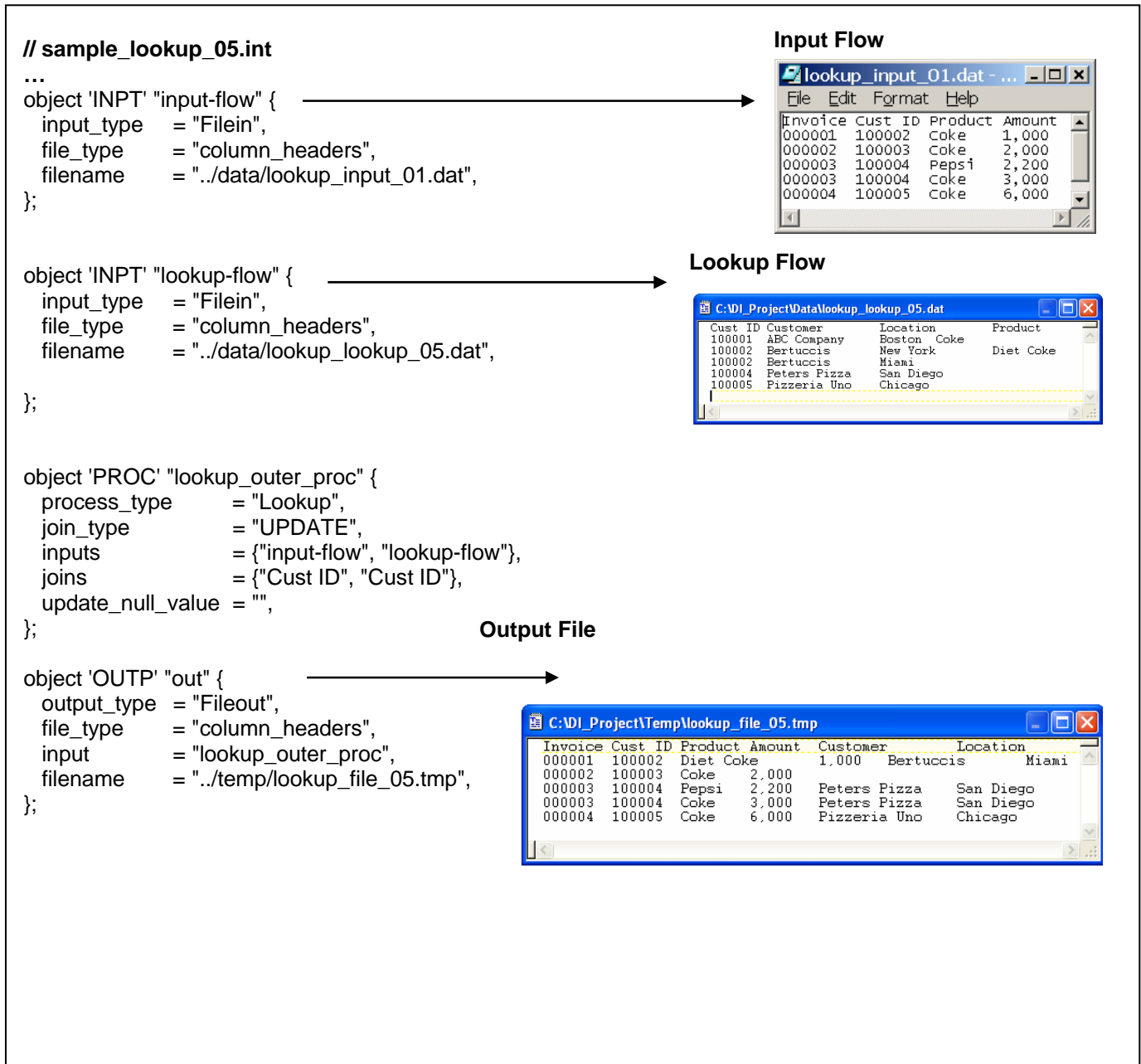
The Output file now contains an **update** of the Input flow. For each of the five records in the Input flow, a record is written to the Output flow. For the common column Product, the value from the Lookup flow is used when it is not null. *Diet Coke* has replaced *Coke* for **Cust ID** 100002. As a result, original Input flow data is changed in the Output file. Note also that, for duplicate rows in the Lookup, as before, the first encountered value is kept. So *Diet Coke* is from one row and *Miami* from another. Be careful when you use the **update** join type!

On the other hand, use of the **outer\_update** join type can simplify the Lookup Outer join as described in Example 3. With the **outer\_update** join type there is no need to alias the join column in the Lookup flow, nor to include a calculation to merge the columns. Outer\_update will append any non-matched Lookup rows to the data flow and retain the value from the Lookup file. The file **sample\_lookup\_05b.int** is an example.



## Lookup Process Object – UPDATE

- Can correct errors in an existing file where the Lookup file becomes a list of changes based on a key field.
- Provides a means to perform incremental updates.



### Lookup – Example 5

The **sample\_lookup\_05.int** script contains the script from Example 1, an **INNER** lookup, with the following change: the **join\_type** attribute has been changed to **UPDATE**. (The **join\_type** attribute **update\_null\_value** is displayed with its default for illustration only.)

The Output file now contains an **update** of the Input flow. For each of the five records in the Input flow, a record is written to the Output flow. For the common column Product, the value from the Lookup flow is used when it is not null. *Diet Coke* has replaced *Coke* for **Cust ID** 100002. As a result, original Input flow data is changed in the Output file. Note also that, for duplicate rows in the Lookup, as before, the first encountered value is kept. So *Diet Coke* is from one row and *Miami* from another. Be careful when you use the **update** join type!

On the other hand, use of the **outer\_update** join type can simplify the Lookup Outer join as described in Example 3. With the **outer\_update** join type there is no need to alias the join column in the Lookup flow, nor to include a calculation to merge the columns. Outer\_update will append any non-matched Lookup rows to the data flow and retain the value from the Lookup file. The file **sample\_lookup\_05b.int** is an example.



## Lookup – OUTER UPDATE

- Can be used to replace nulls in the Input flow
- Simplifies the Outer Join - no post-join Calc Process required

Data Flow A (Input)			Data Flow B (Lookup)	
ProdID	Sales	Name	ProdID	Name
1	100	Product A	1	null
3	100	Product B	2	Product B
4	100	Product C	3	Product C
6	100	Product F	4	Product D
			5	Product E

An Outer Update Lookup of Data Flows A and B - Joins: ProdId

ProdID	Sales	Name
1	100	Product A
3	100	Product C
4	100	Product D
6	100	Product F
2	null	Product B
5	null	Product E

**Note:** In this example the join columns are named the same in both data flows.



## Join Process Object

- SQL-style joins  
(inner, outer, left outer, right outer, merge)
- Similar to “Lookup”
  - Joins 2 data flows (Input/left and Lookup/right)
  - Similar object attributes but...
- Different than “Lookup”
  - Does not require one of the data flows to fit into memory
  - Flows must be sorted on the join columns prior to joining
  - Cartesian products possible



### Process Object – JOIN

The Join Process Object can be used to perform the standard SQL **inner**, **outer**, **left outer** and **right outer** joins. It combines a “left” Input flow with a “right” Input flow, using one or more columns to identify matching rows. The Join object will NOT read one of the Input flows into memory BUT the two Input flows **MUST** be sorted prior to the Join Process, or your results may not be as expected! Integrator will warn a flow is unsorted but continue to process.

#### Inner

With an “inner” join, only rows that have matching join columns in both Input flows are returned in the Output flow. If a flow has duplicate rows with the same join column values, then a Cartesian product of the matching rows is returned in the Output flow.

#### Outer

With an “outer” join, all rows from both Input flows are returned in the Output flow. If a row in one flow does not have a match in the other flow, then it is matched with a row of blank values. As with the inner join, a Cartesian product is returned for duplicate rows when present in both flows.

#### Left Outer

With a “left outer” join, all rows from the first (left) flow are returned. If the values of the join columns in the left flow do not match the values of the join columns for any row in the right flow, they are matched with a row of blank values for right flow columns. If the right flow has duplicate rows with the same join column values, then a Cartesian product of the matching rows is returned in the Output flow. Similar to Lookup Inner join.

#### Right Outer

With a “right outer” join, all rows from the second (right) flow are returned. If the values of the join columns for a row in the right flow do not match the values of the join columns for any row in the left flow, they are matched with a row of blank values for left flow columns. If the left flow has duplicate rows with the same join column values, then a Cartesian product of the matching rows is returned in the Output flow.

#### Merge

All rows from both flows are returned in the sorted order of the join column(s). This is the same as using the Concat object on the two Input flows, except that the rows are interleaved based on the sort instead of one Input flow appearing after the other. Useful when you want to maintain the sort order of the rows.



## Join Process Object

```
object 'PROC' "____" {
  process_type = "Join",
  join_type    = "inner", "outer", "left outer", "right outer",
               or "merge",
  inputs       = { "____", "____" },
  joins        = { "____", "____" },
  multijoins   = {
    { "____", "____" },
    { "____", "____" }
  },
  case_sensitive = "true" or "false",
  locale         = "true" or "false",
  numeric_columns = { "____", "____", "____" },
  autosort       = "true", "false", "left" or "right",
};
```



### Process Object – JOIN (continued)

#### **join\_type**

Defines whether or not the join will be an "inner" join, "outer" join, "left outer" join, "right outer" join, or "merge".

#### **inputs**

Defines the "left" flow and "right" flow.

#### **joins**

Used for a single column match. Defines the columns that must match for the join. The first column name is a column in the "left" flow; the second column name is a column in the "right" flow.

#### **multijoins**

Used for a multicolumn match. Each element of the array is a two-element array (pair) of column names. The first element of the pair is a column name in the "left" flow. The second element of the pair is a corresponding "right" flow.

#### **case\_sensitive**

Controls whether or not comparisons are case sensitive or case insensitive. Valid entries are "true" or "false". If the attribute is not given, comparisons will be case insensitive.

#### **numeric\_columns**

Identifies join columns that are sorted numerically as opposed to alphabetically (Latin 1 char set).

#### **autosort**

Automatically sorts one or both of the Input flows creating an anonymous Sort object with sort columns matching the appropriate join columns. The Sort object uses the default values for Sort Attributes such as temp\_directory and sort\_size, so if these need to be set, you will have to use an explicit Sort object. If the value is "true" then both flows are sorted; if the value is "left" then the left flow is sorted; if the value is "right" then the right flow is sorted. The default value is "false".

#### **locale**

Controls whether or not the sort comparisons are done using the locale collation sequence, to reflect international sort sequences. If this attribute is "true", then the comparisons are done using the current locale (as defined by the system). This attribute defaults to "false", and overrides the **case\_sensitive** attribute.

```
// sample_join_01.int

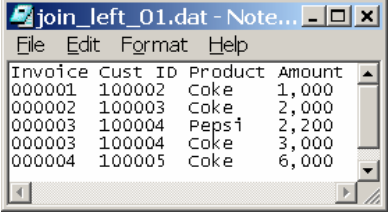
object 'INPT' "left-flow" {
  input_type   = "Filein",
  file_type    = "column_headers",
  filename     = "../data/join_left_01.dat",
};

object 'INPT' "right-flow" {
  input_type   = "Filein",
  file_type    = "column_headers",
  filename     = "../data/join_right_01.dat",
};

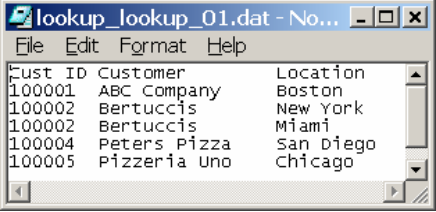
object 'PROC' "join_inner_proc" {
  process_type = "join",
  join_type    = "INNER",
  autosort    = "TRUE",
  inputs      = {"left-flow", "right-flow"},
  joins       = {"Cust ID", "Cust ID"},
};

object 'OUTP' "out" {
  output_type  = "Fileout",
  file_type    = "column_headers",
  input        = "join_inner_proc",
  filename     = "../temp/join_file_01.tmp",
};
```

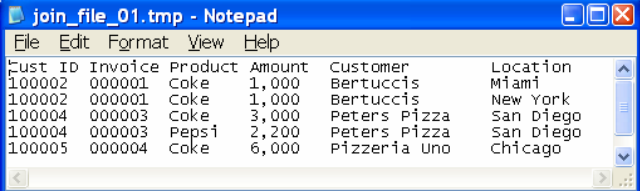
**Left Flow**



**Right Flow**



**Output File**



### Join – Example 1

The `sample_join_01.int` script contains an **inner** join that joins the “left” flow with the “right” flow based on the column **Cust ID**. The resulting file contains five rows.

**With an "inner" join, only rows that have matching join columns in both Input flows are returned in the Output flow.**

The row for **Invoice** 000002 in the “left” flow is not in the Output file because its **Cust ID** (100003) is not in the “right” flow. The row for **Cust ID** 100001 is not in the Output flow because there are no rows for this **Cust ID** in the “left” flow.

**With an "inner" join, if a flow has duplicate rows with the same join column values, then a Cartesian product of the matching rows is returned in the Output flow.**

**Invoice** 000001, which occurs once in the “left” flow, now appears twice in the Output file because its **Cust ID** 100002 appears twice in the “right” flow. The **Amount** from this invoice (\$1,000) will now be double-counted in our Model. OUCH!

```
// sample_join_02.int
```

```
object 'INPT' "left-flow" {  
  input_type      = "Filein",  
  file_type       = "column_headers",  
  filename        = "../data/join_left_01.dat",  
};
```

Left Flow

Invoice	Cust ID	Product	Amount
000001	100002	Coke	1,000
000002	100003	Coke	2,000
000003	100004	Pepsi	2,200
000003	100004	Coke	3,000
000004	100005	Coke	6,000

```
object 'INPT' "right-flow" {  
  input_type      = "Filein",  
  file_type       = "column_headers",  
  filename        = "../data/join_right_01.dat",  
};
```

Right Flow

Cust ID	Customer	Location
100001	ABC Company	Boston
100002	Bertuccis	New York
100002	Bertuccis	Miami
100004	Peters Pizza	San Diego
100005	Pizzeria Uno	Chicago

```
object 'PROC' "join_inner_proc" {  
  process_type    = "join",  
  join_type       = "OUTER",  
  autosort        = "TRUE",  
  inputs          = {"left-flow", "right-flow"},  
  joins           = {"Cust ID", "Cust ID"},  
};
```

```
object 'OUTP' "out" {  
  output_type     = "Fileout",  
  file_type       = "column_headers",  
  input           = "join_inner_proc",  
  filename        = "../temp/join_file_02.tmp",  
};
```

Output File

Cust ID	Invoice	Product	Amount	Customer	Location
100002	000001	Coke	1,000	ABC Company	Boston
100002	000001	Coke	1,000	Bertuccis	New York
100003	000002	Coke	2,000	Bertuccis	Miami
100004	000003	Coke	2,000	Peters Pizza	San Diego
100004	000003	Pepsi	2,200	Peters Pizza	San Diego
100005	000004	Coke	6,000	Pizzeria Uno	Chicago

## Join – Example 2

In the **sample\_join\_02.int** script above, we have changed the join type in the previous example to an **outer** join. The resulting Output file contains seven rows.

**With an “outer” join, all rows from both Input flows are returned in the Output flow. If a row in one flow does not have a match in the other flow, then it is matched with a row of blank values.**

The row for **Invoice 000002** in the “left” flow and the row for **Cust ID 100001** in the “right” flow now appear in the Output file. Notice how the other columns in these rows have been set to blanks and we don’t see the **Cust ID** on the row for **Cust ID 100001**. We would have to alias the **Cust ID** column in the “right” flow in order to make it appear.

**As with the inner join, a Cartesian product is returned for duplicate rows when present in both flows. Invoice 000001**, which occurs once in the “left” flow, still appears twice in the Output file because its **Cust ID 100002** appears twice in the “right” flow. The **Amount** from this invoice (\$1,000) will still be double-counted in our Model. OUCH!



## Join Process Object

- Inner
  - Join type not available in Lookup object
- Outer
  - Compared to a Lookup Outer, Output rows are sorted

```
// sample_join_03.int

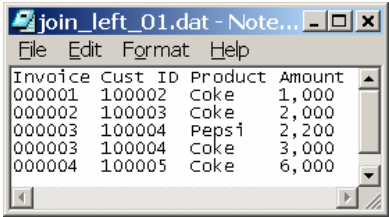
object 'INPT' "left-flow" {
  input_type      = "Filein",
  file_type       = "column_headers",
  filename        = "../data/join_left_01.dat",
};

object 'INPT' "right-flow" {
  input_type      = "Filein",
  file_type       = "column_headers",
  filename        = "../data/join_right_01.dat",
};

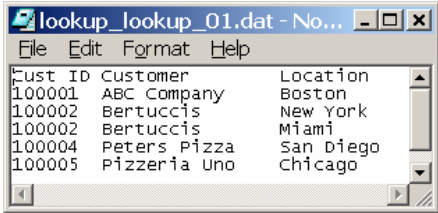
object 'PROC' "join_inner_proc" {
  process_type    = "join",
  join_type       = "LEFT OUTER",
  autosort        = "TRUE",
  inputs          = {"left-flow", "right-flow"},
  joins           = {"Cust ID", "Cust ID"},
};

object 'OUTP' "out" {
  output_type     = "Fileout",
  file_type       = "column_headers",
  input           = "join_inner_proc",
  filename        = "../temp/join_file_03.tmp",
};
```

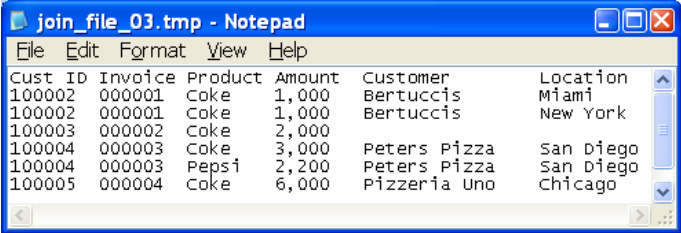
**Left Flow**



**Right Flow**



**Output File**



### Join – Example 3

In the `sample_join_03.int` script above, we have changed the join type in the previous example to a **left outer** join. The resulting Output file contains six rows.

**With a “left outer” join, all rows from the first (left) flow are returned. If the values of the join columns for a row in the left flow do not match the values of the join columns for any row in the right flow, they are matched with a row of blank values for right flow columns.**

The row for **Invoice** 000002 in the “left” flow is included in the Output file, but the row for **Cust ID** 100001 in the “right” flow is NOT.

**If the “right” flow has duplicate rows with the same join column values, then a Cartesian product of the matching rows is returned in the Output flow.**

**Invoice** 000001, which occurs once in the “left” flow, still appears twice in the Output file because its **Cust ID** 100002 appears twice in the “right” flow. The **Amount** from this invoice (\$1,000) will still be double-counted in our Model. OUCH!

```
// sample_join_04.int
```

```
object 'INPT' "left-flow" {  
  input_type      = "Filein",  
  file_type       = "column_headers",  
  filename        = "../data/join_left_01.dat",  
};
```

Left Flow

Invoice	Cust ID	Product	Amount
000001	100002	Coke	1,000
000002	100003	Coke	2,000
000003	100004	Pepsi	2,200
000004	100004	Coke	3,000
000004	100005	Coke	6,000

```
object 'INPT' "right-flow" {  
  input_type      = "Filein",  
  file_type       = "column_headers",  
  filename        = "../data/join_right_01.dat",  
};
```

Right Flow

Cust ID	Customer	Location
100001	ABC Company	Boston
100002	Bertuccis	New York
100002	Bertuccis	Miami
100004	Peters Pizza	San Diego
100005	Pizzeria Uno	Chicago

```
object 'PROC' "join_inner_proc" {  
  process_type    = "join",  
  join_type       = "RIGHT OUTER",  
  autosort        = "TRUE",  
  inputs          = {"left-flow", "right-flow"},  
  joins           = {"Cust ID", "Cust ID"},  
};
```

```
object 'OUTP' "out" {  
  output_type     = "Fileout",  
  file_type       = "column_headers",  
  input           = "join_inner_proc",  
  filename        = "../temp/join_file_04.tmp",  
};
```

Output File

Cust ID	Invoice	Product	Amount	Customer	Location
100002	000001	Coke	1,000	ABC Company	Boston
100002	000001	Coke	1,000	Bertuccis	Miami
100004	000003	Coke	3,000	Peters Pizza	San Diego
100004	000003	Pepsi	2,200	Peters Pizza	San Diego
100005	000004	Coke	6,000	Pizzeria Uno	Chicago

#### Join – Example 4

In the **sample\_join\_04.int** script above, we have changed the join type in the previous example to a **right outer** join. The resulting Output file contains six rows.

**With a “right outer” join, all rows from the second (right) flow are returned. If the values of the join columns for a row in the right flow do not match the values of the join columns for any row in the left flow, they are matched with a row of blank values for left flow columns.**

The row for **Cust ID 100001** in the “right” flow is included in the Output file, but the row for **Invoice 000002** in the “left” flow is NOT.

**If the “left” flow has duplicate rows with the same join column values, then a Cartesian product of the matching rows is returned in the Output flow.**

**Invoice 000001**, which occurs once in the “left” flow, still appears twice in the Output file because its **Cust ID 100002** appears twice in the “right” flow. The **Amount** from this invoice (\$1,000) will still be double-counted in our Model. OUCH!



## Join Process Object

- Left Outer
  - Equivalent to the Lookup Inner join type, except that the Cartesian product is possible
- Right Outer
  - Be careful - if there are columns in both flows with the same name, the left values are kept!

## // sample\_join\_05.int

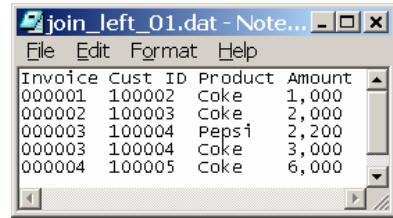
```
object 'INPT' "left-flow" {
  input_type   = "Filein",
  file_type    = "column_headers",
  filename     = "../data/join_left_01.dat",
};

object 'INPT' "right-flow" {
  input_type   = "Filein",
  file_type    = "column_headers",
  filename     = "../data/join_right_01.dat",
};

object 'PROC' "join_inner_proc" {
  process_type = "join",
  join_type    = "MERGE",
  autosort    = "true",
  inputs      = {"left-flow", "right-flow"},
  joins       = {"Cust ID", "Cust ID"},
};

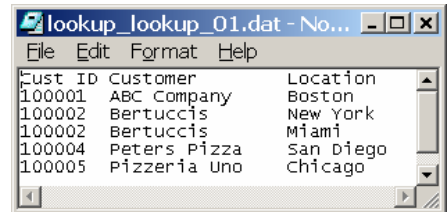
object 'OUTP' "out" {
  output_type  = "Fileout",
  file_type    = "column_headers",
  input        = "join_inner_proc",
  filename     = "../temp/join_file_05.tmp",
};
```

### Left Flow



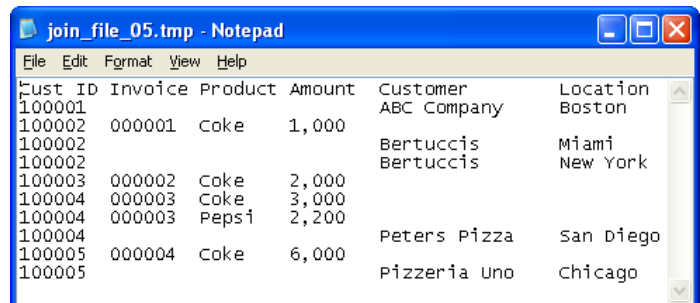
Invoice	Cust ID	Product	Amount
000001	100002	Coke	1,000
000002	100003	Coke	2,000
000003	100004	Pepsi	2,200
000003	100004	Coke	3,000
000004	100005	Coke	6,000

### Right Flow



Cust ID	Customer	Location
100001	ABC Company	Boston
100002	Bertuccis	New York
100002	Bertuccis	Miami
100004	Peters Pizza	San Diego
100005	Pizzeria Uno	Chicago

### Output File



Cust ID	Invoice	Product	Amount	Customer	Location
100001	000001	Coke	1,000	ABC Company	Boston
100002				Bertuccis	Miami
100002				Bertuccis	New York
100003	000002	Coke	2,000		
100004	000003	Coke	3,000		
100004	000003	Pepsi	2,200		
100004				Peters Pizza	San Diego
100005	000004	Coke	6,000		
100005				Pizzeria Uno	Chicago

## Join – Example 5

In the **sample\_join\_05.int** script above, we have changed the join type in the previous example to a **merge** join. The resulting Output file contains 10 rows.

**With a “merge” join, all rows from the both the left and right flow are returned. The Records have maintained their sort order by Cust ID.**

From the Output file we see that if either the left or the right flow does not have data for a particular **Cust ID**, then a blank value is returned for that row under any columns that don't match in either flow.



## Lookup versus Join

### When to use Lookup

- Lookup file will fit into memory
- Fast join
- One-to-one relationship between Input and Lookup flow
- You only want one row returned from the right-hand flow
- Take advantage of Update

### When to use Join

- Lookup file is large (won't fit in memory)
- One-to-many relationship between Input and Lookup flow
- Remember to sort both Input flows before joining
- Take advantage of Merge



A Lookup Process Object can join one Lookup file record to many identical values in the Input flow. The Join object will create additional rows as needed to join multiple Lookup values for each Input value.

While a Join object can be used in a "one-to-many" condition as well as the "many-to-many" condition, as long as the Lookup file fits into memory, the Lookup object provides a faster join in the "one-to-many" case, as sorting of the flows is not required.

For both Process Objects, if a common column name exists, the "A" or Input flow values take precedence over the "B" or Lookup flow values, even in the case of Outer joins. This may potentially result in "Null" values in the Output column. To retain "B" values, the column name must be different.



# Data Integrator: Lookups and Joins



© 2010 Dimensional Insight, Inc.