Ab Initio

Introduction

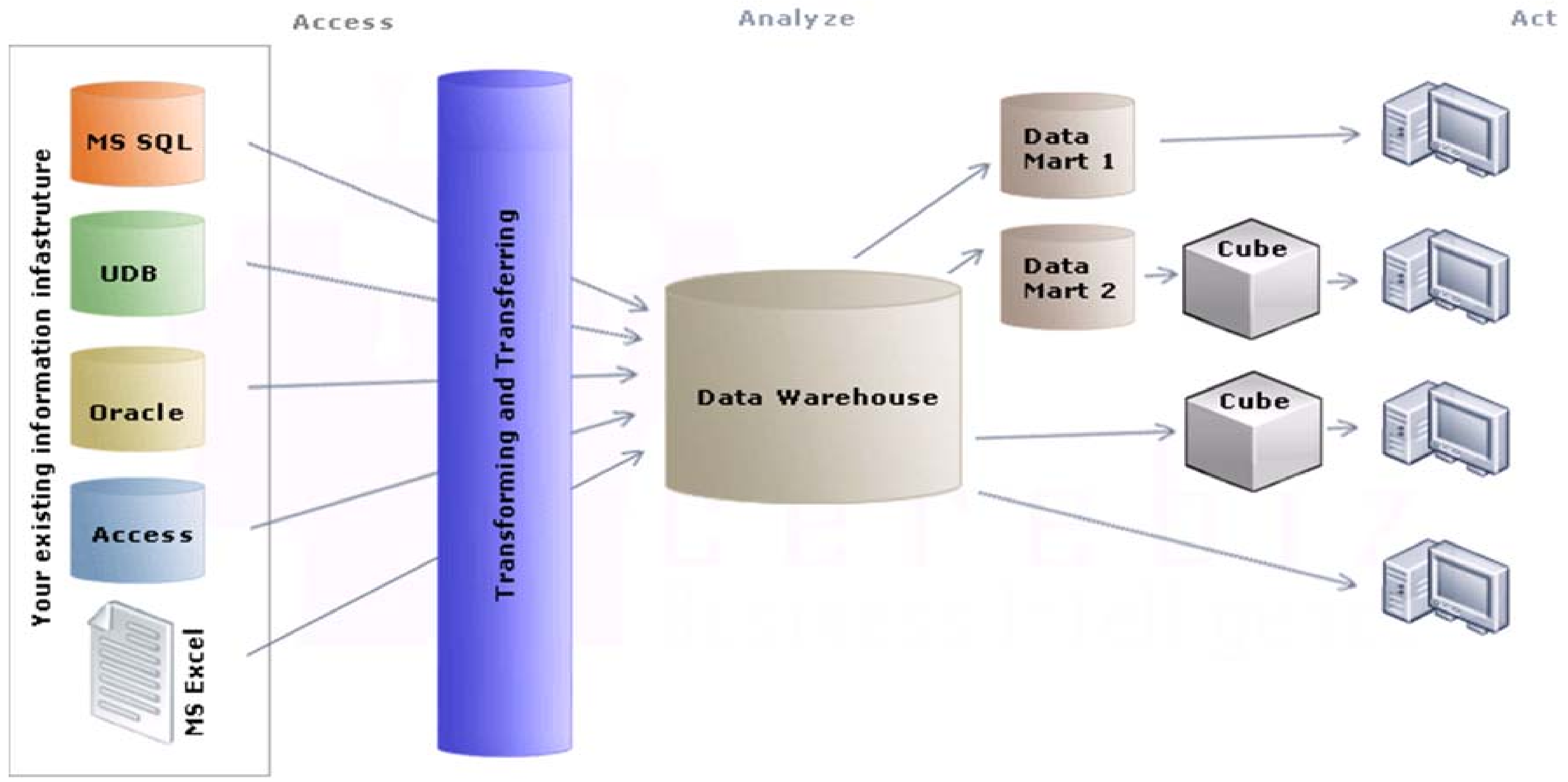
Agenda

- What is Data warehousing?
- Why Data warehousing?
- ETL process
- Various ETL tools
- Introduction about Ab Initio
- why Ab Initio
- How Unix involved with Ab Initio
- GDE window
- EME Repository
- Sandboxes- User and Standard Sandbox
- Ab Initio - Components
- Creation of simple graphs

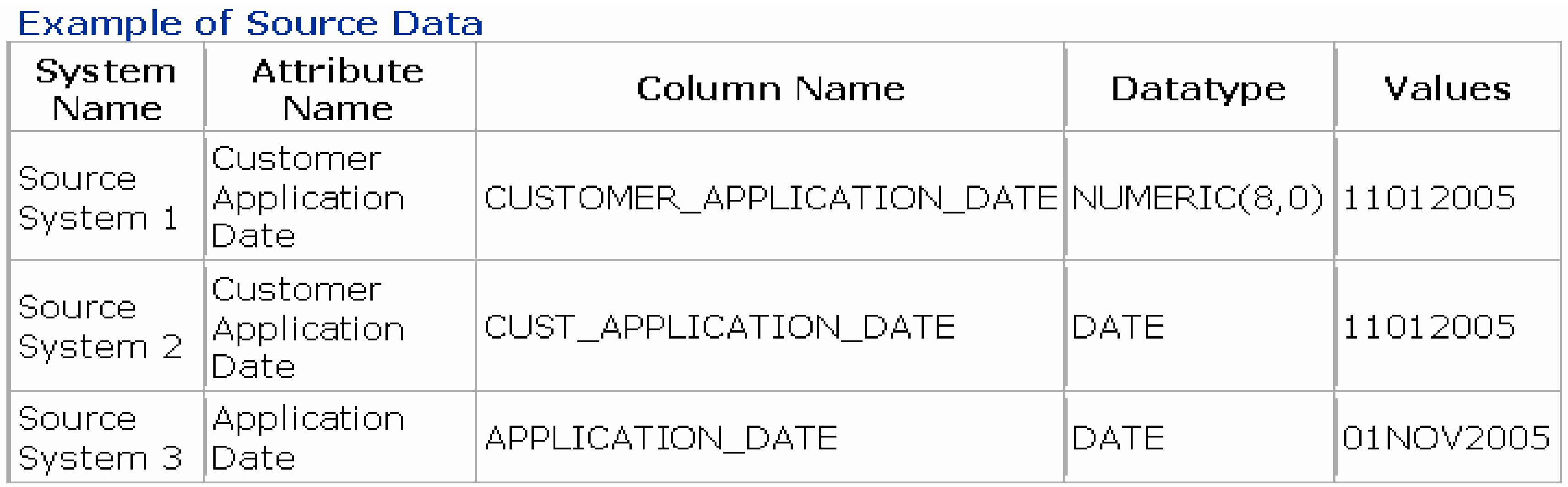
Data warehousing
and
ETL Process

Data Warehouse

Data Warehouse is a collection of ‘logical’ Data Marts, each of which is designed for a particular line of business i.e. Sales, Marketing(designed to favor/facilitate data analysis and reporting).

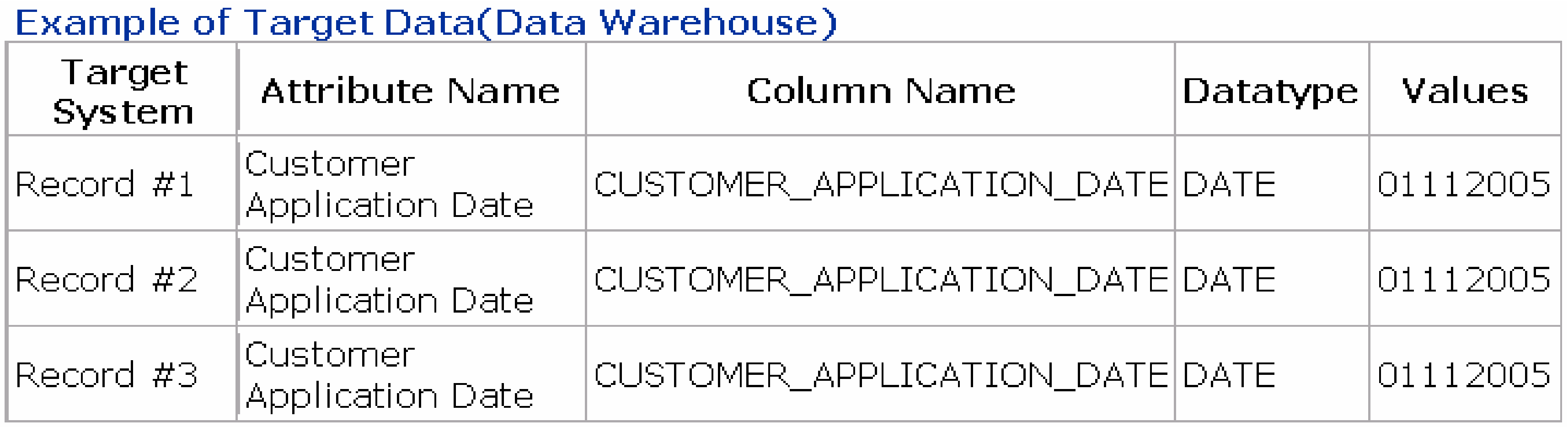


Why Data warehousing ?



Example of Source Data

System Name	Attribute Name	Column Name	Datatype	Values
Source System 1	Customer Application Date	CUSTOMER_APPLICATION_DATE	NUMERIC(8,0)	11012005
Source System 2	Customer Application Date	CUST_APPLICATION_DATE	DATE	11012005
Source System 3	Application Date	APPLICATION_DATE	DATE	01NOV2005



Example of Target Data(Data Warehouse)

Target System	Attribute Name	Column Name	Datatype	Values
Record #1	Customer Application Date	CUSTOMER_APPLICATION_DATE	DATE	01112005
Record #2	Customer Application Date	CUSTOMER_APPLICATION_DATE	DATE	01112005
Record #3	Customer Application Date	CUSTOMER_APPLICATION_DATE	DATE	01112005

ETL process

- Data is first stored temporarily in a ‘Staging Table/Area’ and is called ‘Staging Data’ i.e. Data queued for processing.
- The processing tool reads the ‘Staged Data’, performs qualitative processing, filtering, cleansing (As required for the OLAP i.e. reporting/analysis) and finally loads/writes them into Data Warehouse.
- All these data flow (both inward and outward) and data processing activities (Extraction from Source System – Transformation of data by cleansing/filtering – Loading into Data Warehouse) are performed using an ETL tool i.e. Ab Initio, Informatica etc.
- This entire process is said to be as “ETL process”.

Extract, Transformation, Load (ETL) functionalities

<u>Extract:</u>

The first phase of an ETL process is to extract the data from the source systems. Each separate system may also use a different data organization/ format. Common data source formats are relational databases, and flat files, but other source formats exist. Extraction converts the data into records and columns.

<u>Transform:</u> The transform phase applies a series of rules or functions to the extracted data.

Examples:
 Derive a new calculated value (e.g. sale_amount = qty * unit_price)
 Summarize multiple rows of data (e.g. total sales for each region)

<u>Load:</u>
The load phase loads the data into the data warehouse. Depending on the requirements of the organization, this process ranges widely.
Simple-Overwrite old data with new.
More complex systems ->Maintenance of history and audit trail of all changes to the data

Various Popular ETL Tools

Tool Name	Company Name
Informatica	Informatica Corporation
DT/studio	Embarcadero technologies
Datastage	IBM
Abinitio	Abinitio Software corporation
Talend	Talend corporation
Pentaho	Pentaho corporation
Datajunction	Pervasive Software
Oracle warehouse builder	Oracle Corporation
Microsoft SQl Server Integration	Microsoft

Introduction-Ab-Initio

Ab Initio

- Data processing tool from Ab Initio software corporation (http://www.abinitio.com)
- Latin for “from the beginning”
- Designed to support largest and most complex business applications
- Graphical, intuitive, and “fits the way your business works”.

Focus:

Moving Data -

Move small and large volumes of data in an efficient manner.

Deal with the complexity associated with business data.

High Performance

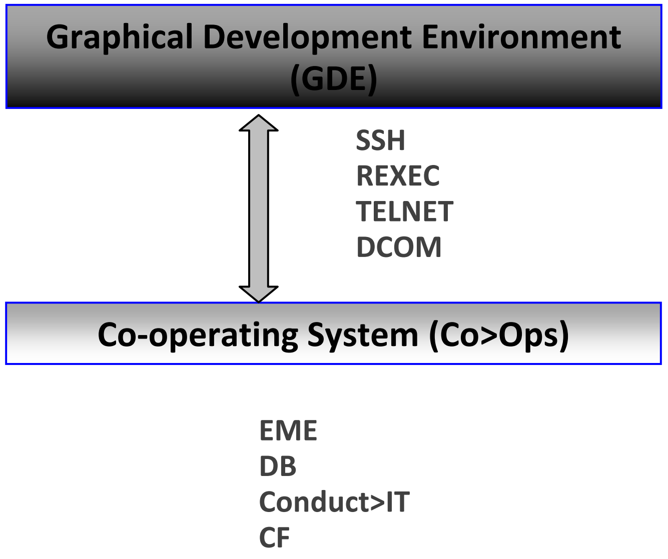
Scalable Solutions

Better productivity

Usage:

- Data Warehousing
- Batch Processing
- Data Movement
- Data Transformation

Product Constituents


Graphical Development Environment (GDE)
SSH
REXEC
TELNET
DCOM
Co-operating System (Co>Ops)
EME
DB
Conduct>IT
CF

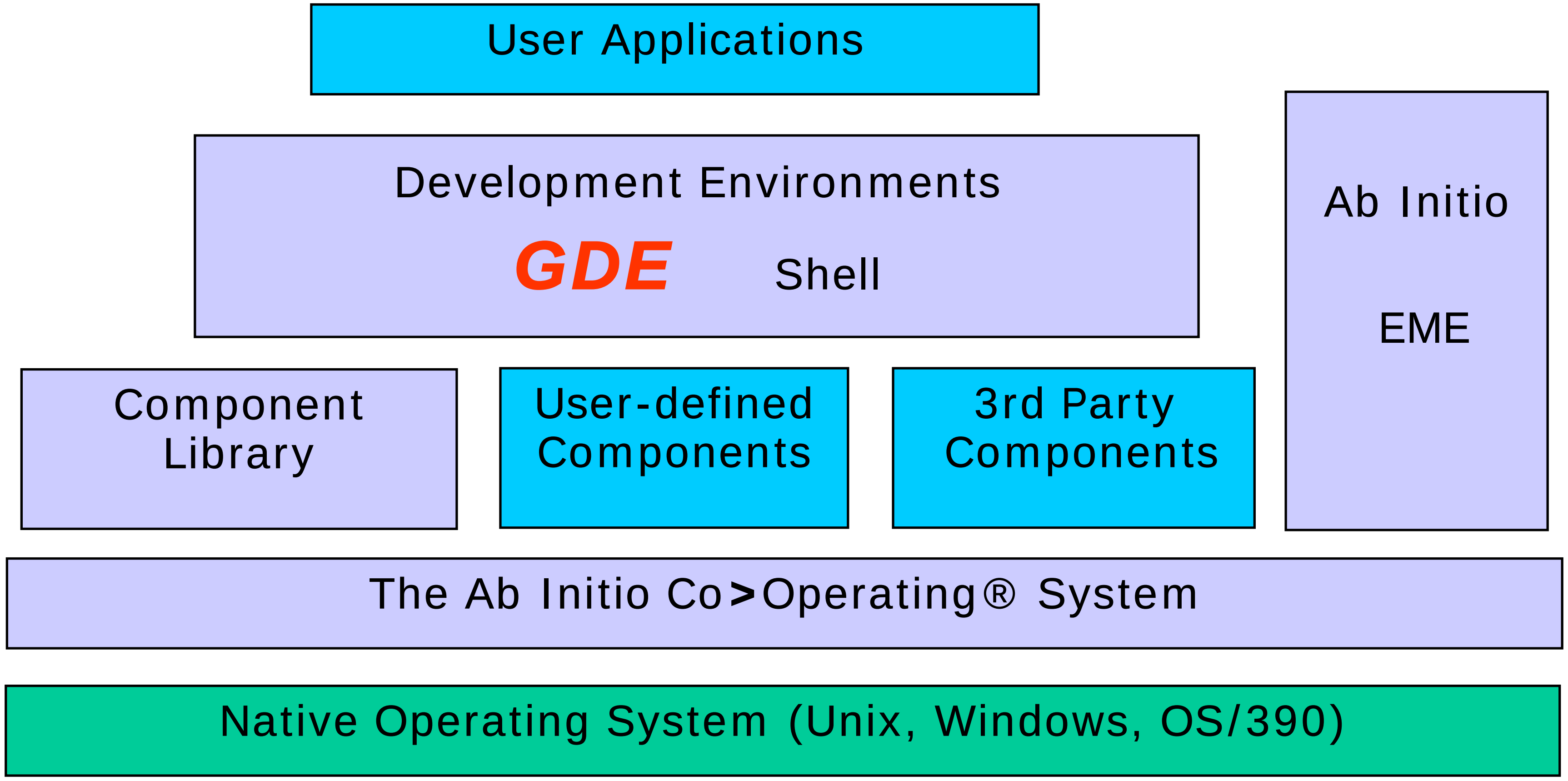
Product	Functionality
GDE	User Interface for creating Graphs and Plans in Ab Initio
Data Profiler	Ab Initio Tool for Data Profiling
Co>Ops	Server Component for running deployed Ab Initio programs
EME	Ab Initio Technical Repository – Part of Co>Ops Install
Database	Ab Initio Server Database Components
Conduct>IT	Ab Initio Server Component for running Ab Initio Plans
Continuous Flow	Ab Initio Server Components for running CF programs

- All server components are installed by default

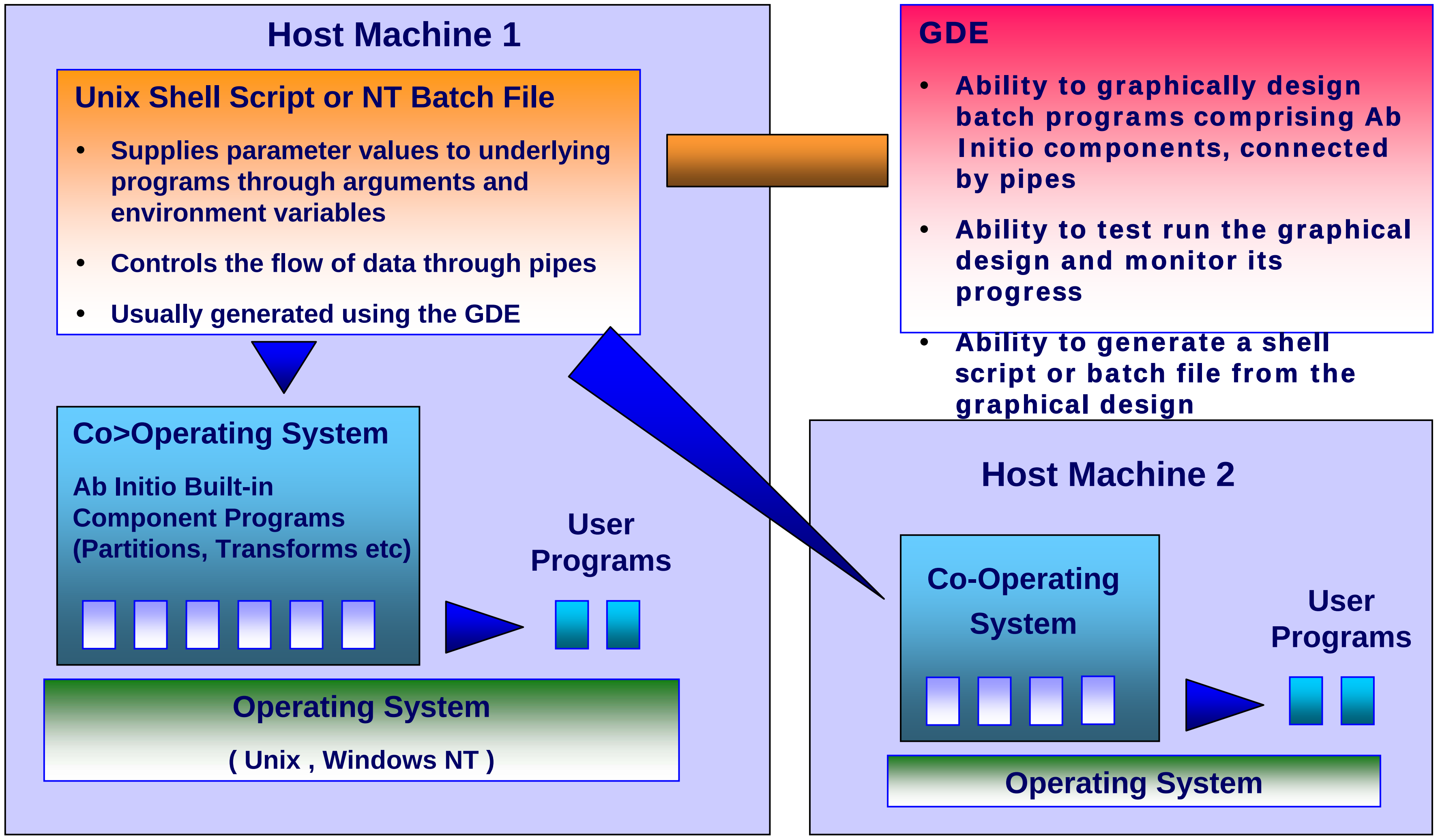
- **“AB_HOME”** refers to installation location of Ab Initio
- Various Connectors and Plugins installed in “**AB_HOME/Connectors** & **AB_HOME/plugins**” location

All binaries and library files available in “**AB_HOME/bin** & **AB_HOME/lib**” respectively

Ab Initio Product Architecture



Product Architecture



Host Machine 1

Unix Shell Script or NT Batch File

- Supplies parameter values to underlying programs through arguments and environment variables
- Controls the flow of data through pipes
- Usually generated using the GDE

Co>Operating System

Ab Initio Built-in Component Programs (Partitions, Transforms etc)

User Programs

Operating System

(Unix , Windows NT)

GDE

- Ability to graphically design batch programs comprising Ab Initio components, connected by pipes
- Ability to test run the graphical design and monitor its progress
- Ability to generate a shell script or batch file from the graphical design

Host Machine 2

Co-Operating System

User Programs

Operating System

Co>Operating System and GDE

- Co>Operating System
  - Layered on the top of the operating system.
  - Unites a network of computing resources – CPUs, storage disks, programs, datasets into a data-processing system with scalable performance.
- GDE
  - can talk to the Co-operating system using several protocols like Telnet, Rexec and FTP
  - It is GUI for building applications in Ab Initio

The Graph Model

Graph

- **is the logical modular unit of an application.**
- **consists of several components that forms the building blocks of an Ab Initio application**
- **Start Script (Host Setup) - Local to the Graph**
- **End Script - Local to the Graph**

Component

- **is a program that does a specific type of job and can be controlled by its parameter settings. Ex: Join, Re-format etc**

Component Organizer

- **Groups all components under different categories.**

Setup Command

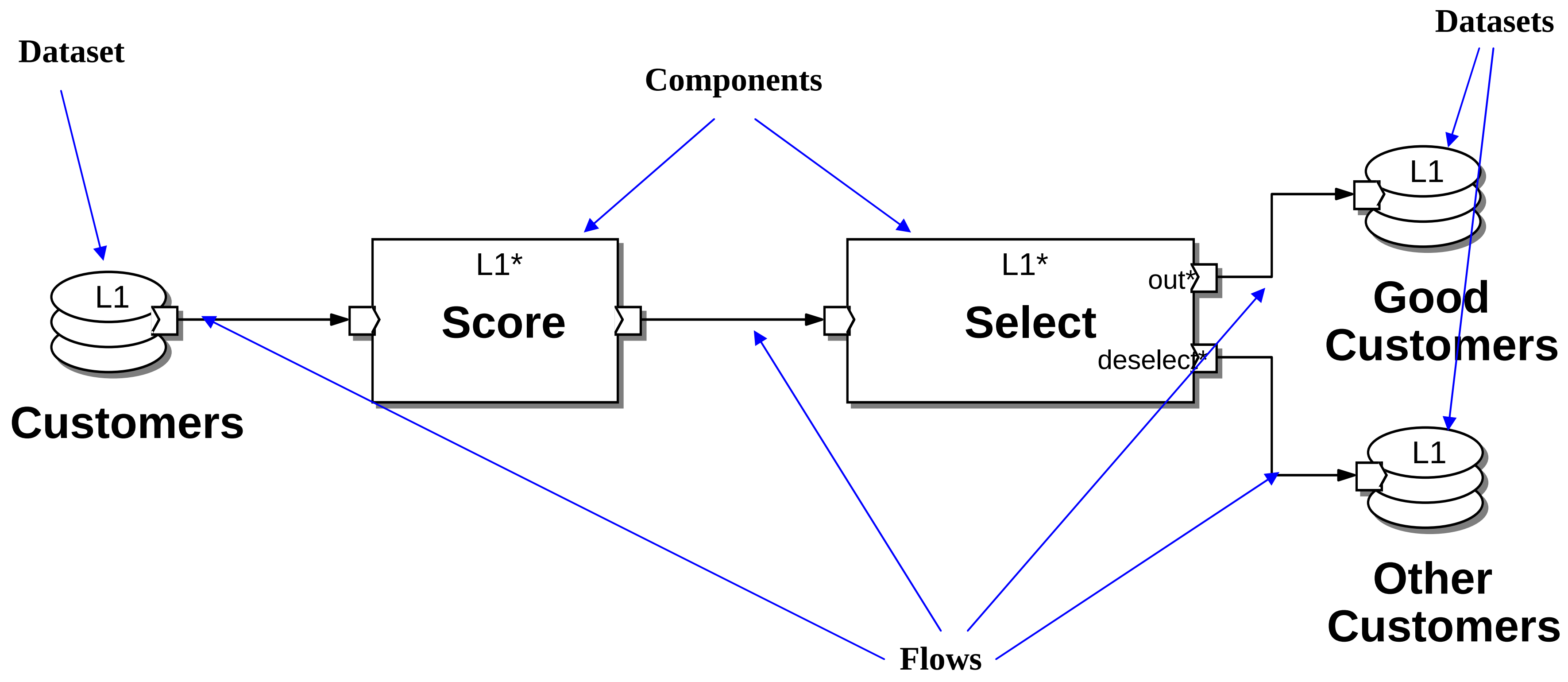
- **Ab Initio Host (AIH) file**
- **Builds up the environment to run an Ab Initio application.**

Parts of typical graph

- Files
- Formats
- Components
- Flows
- Layouts
- Building with *mp job*
- Building with *mp run*

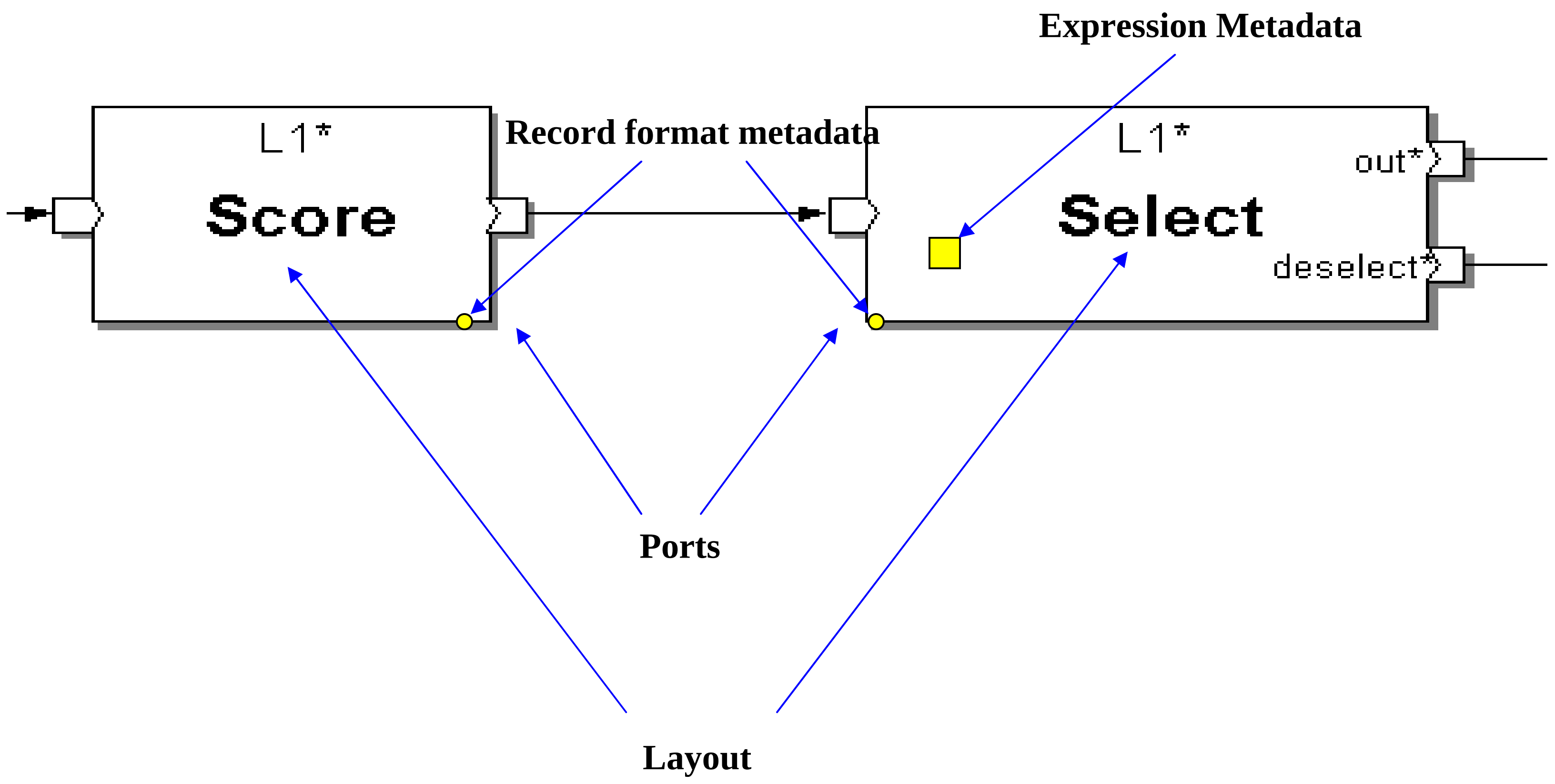
The Graph Model: Naming the pieces

A Sample Graph …



The Graph Model: A closer look

A Sample Graph …



Runtime Environment

- A graph, after development, is deployed to the back-end server as a Unix shell script or Windows NT batch file.
- This becomes the executable to run at the back-end with the help of the Co-operating system.
- The execution can be done from the GDE itself or manually from the back-end
- Ab Initio runtime environment is different from the development environment.

Unix and Abinitio

- Unix serves as backend for Ab-initio.
- All the graphs/Jobs in Ab-initio can be accessible through Unix(backend)
- Putty connectivity

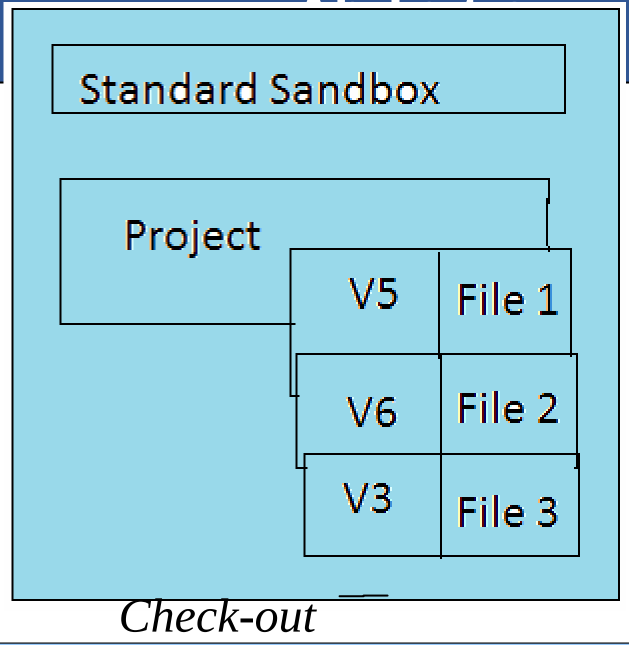
Environment – Quick Overview:

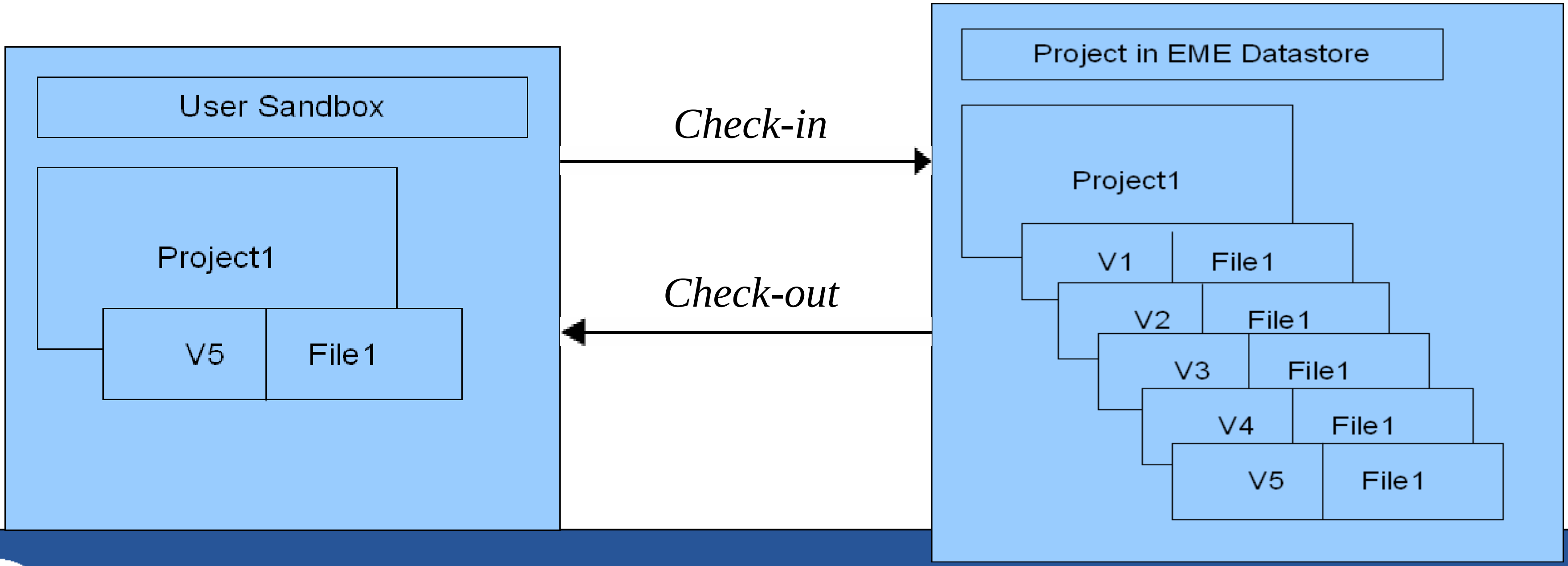
- $AI_RUN,$AI_BIN—run directory, .ksh scripts
  $AI_PLAN, $AI_SERIAL_<LOG/ERROR/SUMMARY>
- $AI_DML—record format files
- $AI_XFR—transform files
- $AI_MP—graphs
- $AI_DB—database config files
- $AI_SERIAL - serial source data, other serial data
- $AI_MFS - Ab Initio multifile directory – in training will also contain partition directories (more about this later!)
- $AI_LOG - A location to place logging files, etc

Sandboxes and EME

• Sandboxes are work areas used to develop, test or run code associated with a given project. Only one version of the code can be held within the sandbox at any time.

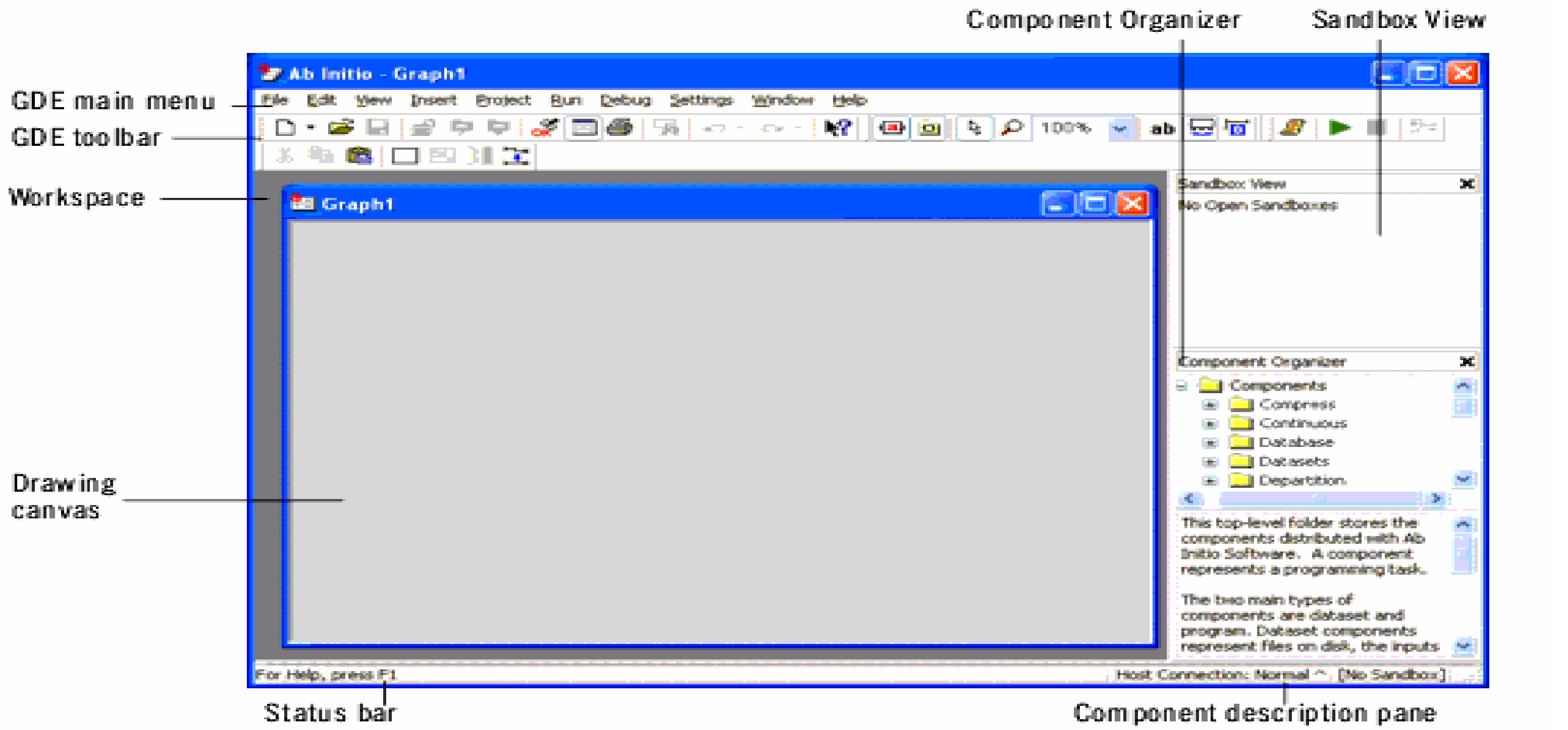
• The EME Datastore contains all versions of the code that have been checked into it.





The GDE window

The GDE window is made up of several panes and toolbars, as shown in the following figure:



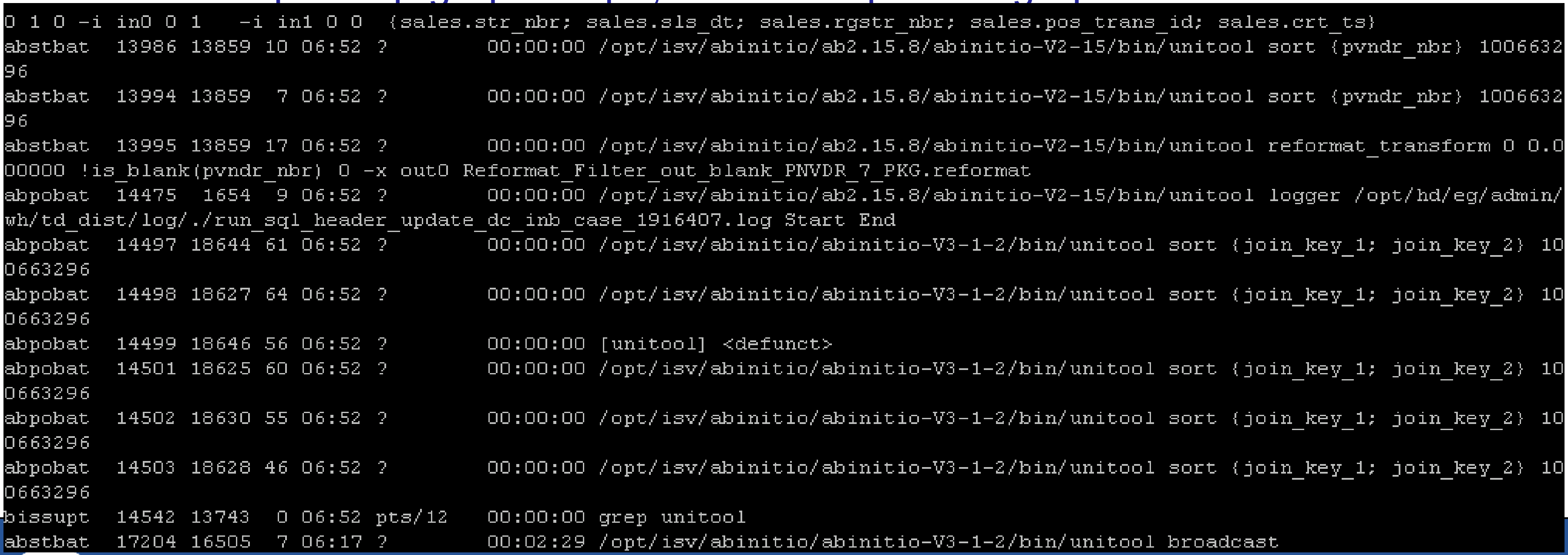
This top-level folder stores the components distributed with Ab Initio Software. A component represents a programming task.

The two main types of components are dataset and program. Dataset components represent files on disk, the inputs

Abinitio Environment - Job run

How a job runs

- The execution of an Ab Initio graph is a job.
- To run a job, need to invoke a shell script that the GDE generates from a graph.
- The script process initiates job processes that control the execution of the programs represented by the graph.
- Graph->mp/graph1.mp ; Shell script->run/graph1.ksh



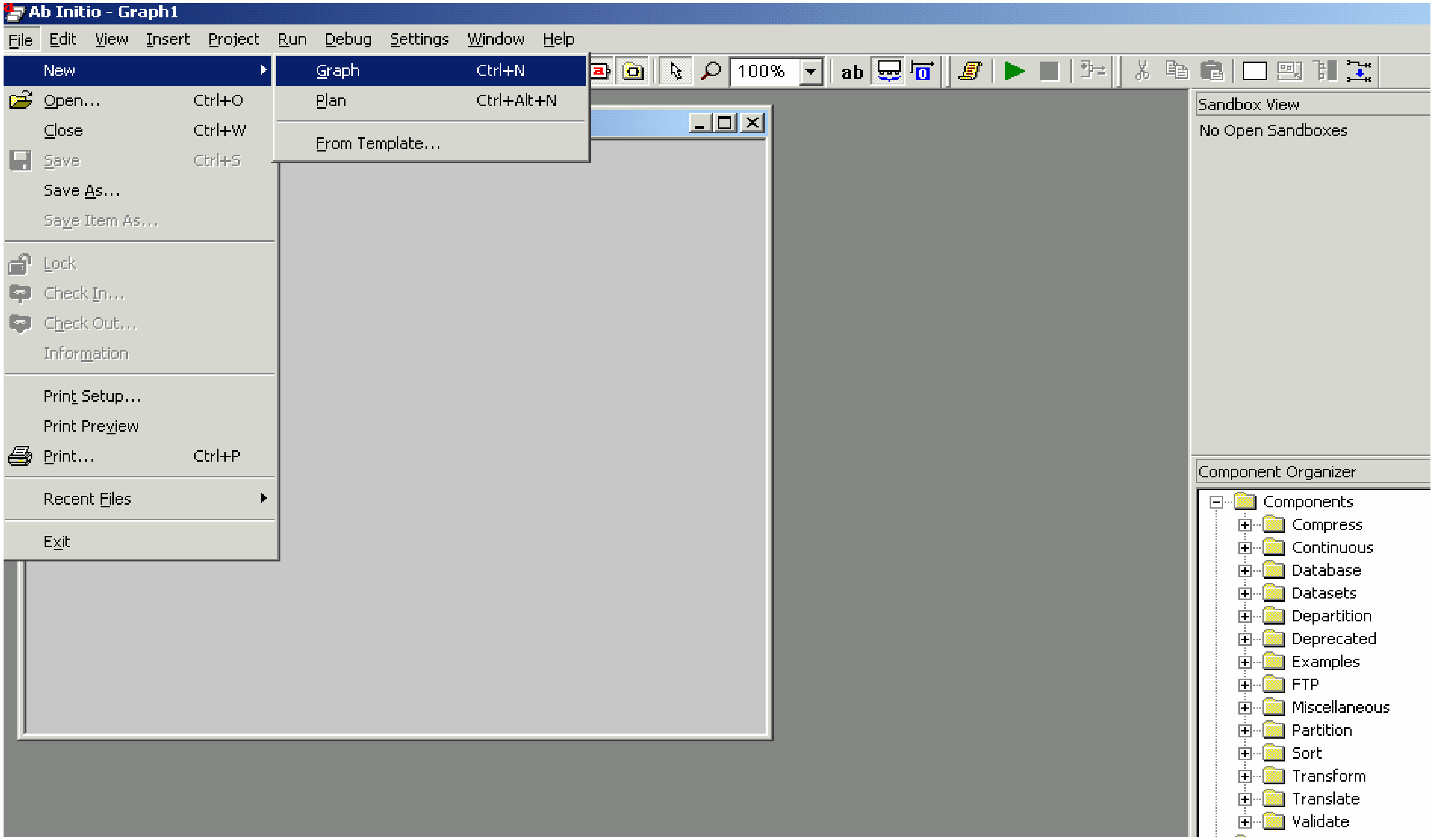
```
0 1 0 -i in0 0 1   -i in1 0 0  {sales.str_nbr; sales.sls_dt; sales.rgstr_nbr; sales.pos_trans_id; sales.crt_ts}
abstbat  13986 13859 10 06:52 ?        00:00:00 /opt/isv/abinitio/ab2.15.8/abinitio-V2-15/bin/unitool sort {pvndr_nbr} 1006632
96
abstbat  13994 13859  7 06:52 ?        00:00:00 /opt/isv/abinitio/ab2.15.8/abinitio-V2-15/bin/unitool sort {pvndr_nbr} 1006632
96
abstbat  13995 13859 17 06:52 ?        00:00:00 /opt/isv/abinitio/ab2.15.8/abinitio-V2-15/bin/unitool reformat_transform 0 0.0
00000 !is_blank(pvndr_nbr) 0 -x out0 Reformat_Filter_out_blank_PNVDR_7_PKG.reformat
abpobat  14475  1654  9 06:52 ?        00:00:00 /opt/isv/abinitio/ab2.15.8/abinitio-V2-15/bin/unitool logger /opt/hd/eg/admin/
wh/td_dist/log/./run_sql_header_update_dc_inb_case_1916407.log Start End
abpobat  14497 18644 61 06:52 ?        00:00:00 /opt/isv/abinitio/abinitio-V3-1-2/bin/unitool sort {join_key_1; join_key_2} 10
0663296
abpobat  14498 18627 64 06:52 ?        00:00:00 /opt/isv/abinitio/abinitio-V3-1-2/bin/unitool sort {join_key_1; join_key_2} 10
0663296
abpobat  14499 18646 56 06:52 ?        00:00:00 [unitool] <defunct>
abpobat  14501 18625 60 06:52 ?        00:00:00 /opt/isv/abinitio/abinitio-V3-1-2/bin/unitool sort {join_key_1; join_key_2} 10
0663296
abpobat  14502 18630 55 06:52 ?        00:00:00 /opt/isv/abinitio/abinitio-V3-1-2/bin/unitool sort {join_key_1; join_key_2} 10
0663296
abpobat  14503 18628 46 06:52 ?        00:00:00 /opt/isv/abinitio/abinitio-V3-1-2/bin/unitool sort {join_key_1; join_key_2} 10
0663296
bissupt  14542 13743  0 06:52 pts/12   00:00:00 grep unitool
abstbat  17204 16505  7 06:17 ?        00:02:29 /opt/isv/abinitio/abinitio-V3-1-2/bin/unitool broadcast
```

Abinitio Environment - Job run

You can invoke the script in two ways:

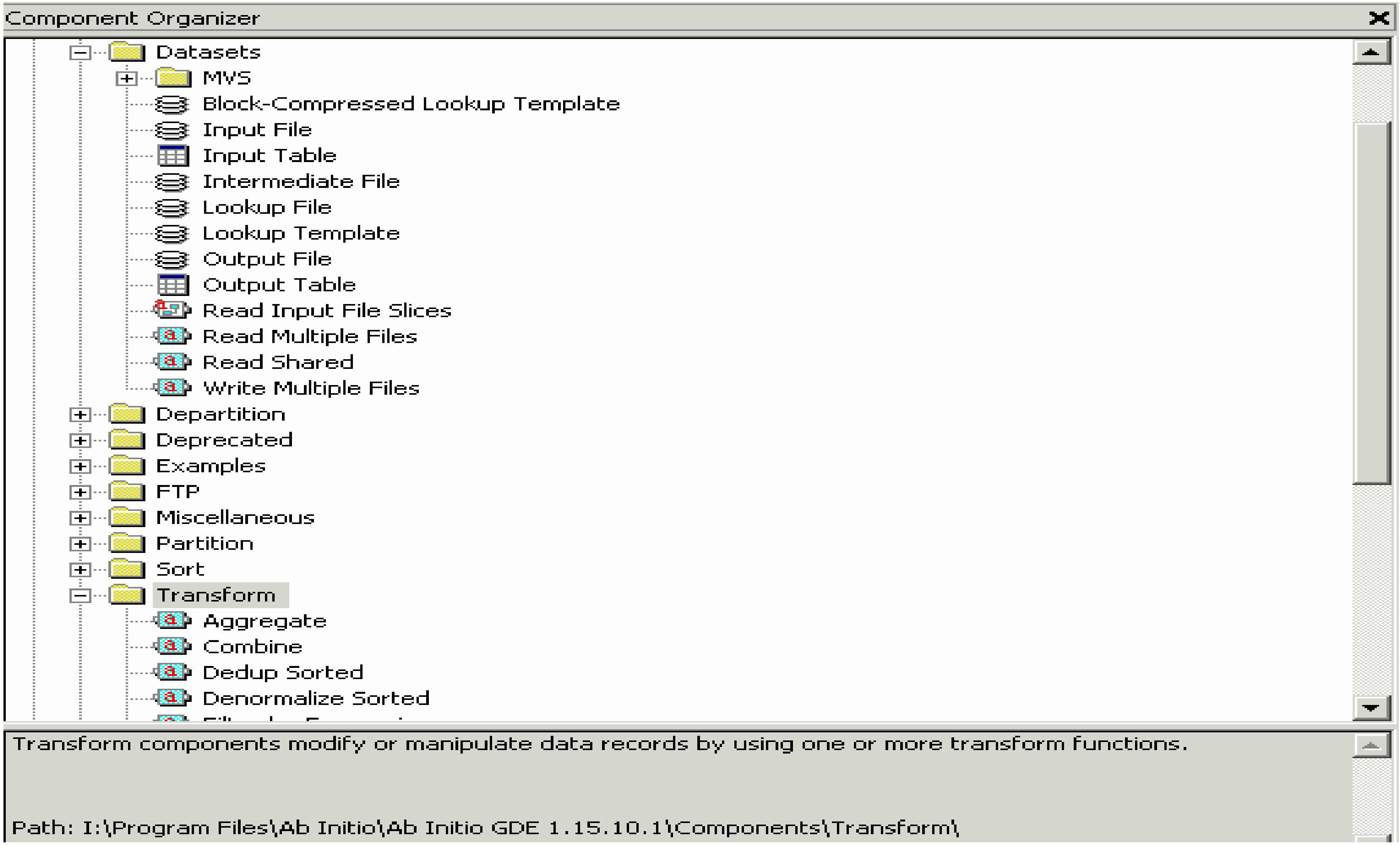
- ❑ From the GDE
- ❑ From a command line
- ❑ To invoke the script from the GDE, click the Run button or choose Run > Start from the GDE menu bar.
- ❑ To invoke the script through command line,
- ❑ For bin script: ksh scriptname.ksh in bin path.
- ❑ To run a graph from backend: $AI_RUN Graphname.ksh parameters(if needed) in run path.

Creation of a Graph



Components - Overview

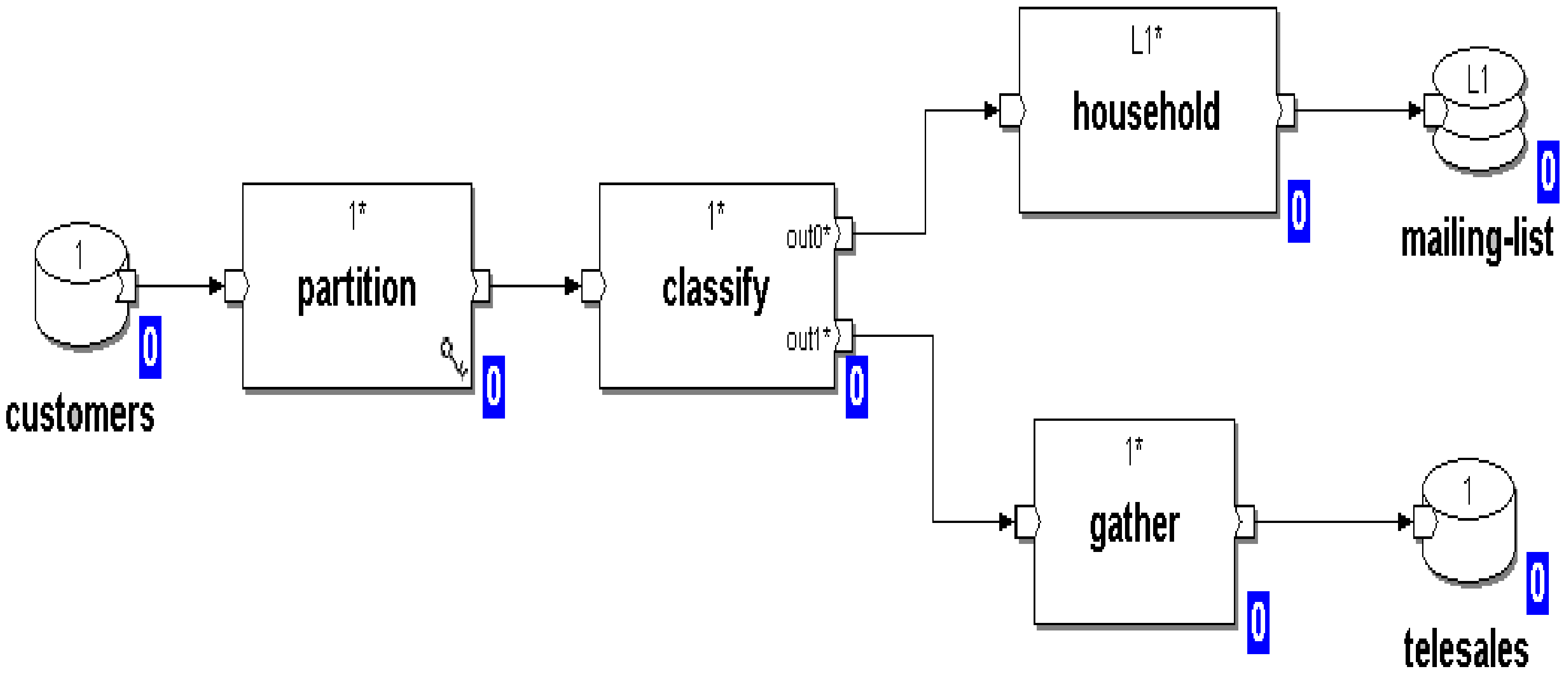
Component Organizer



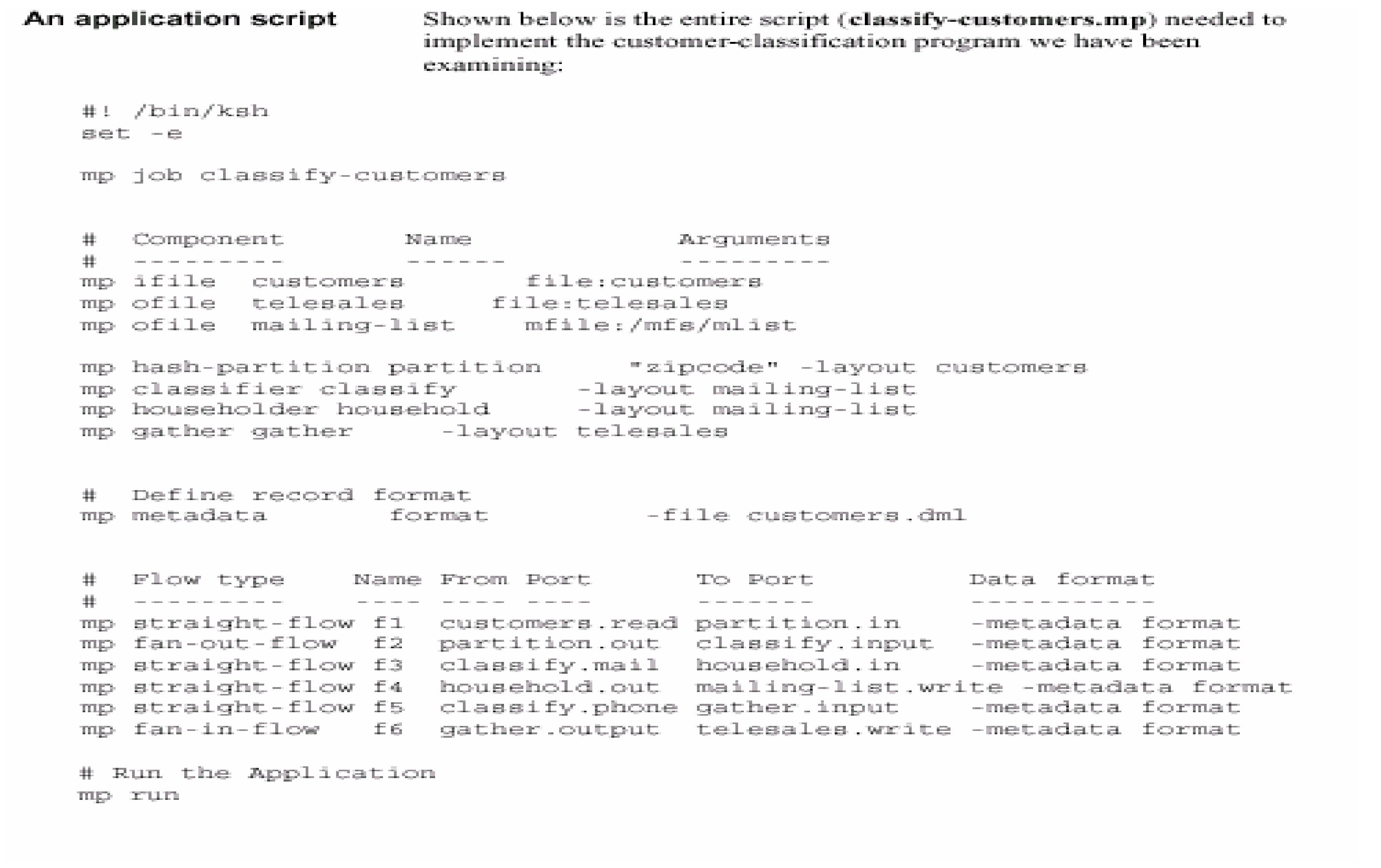
Transform components modify or manipulate data records by using one or more transform functions.

Path: I:\Program Files\Ab Initio\Ab Initio GDE 1.15.10.1\Components\Transform\

A sample graph



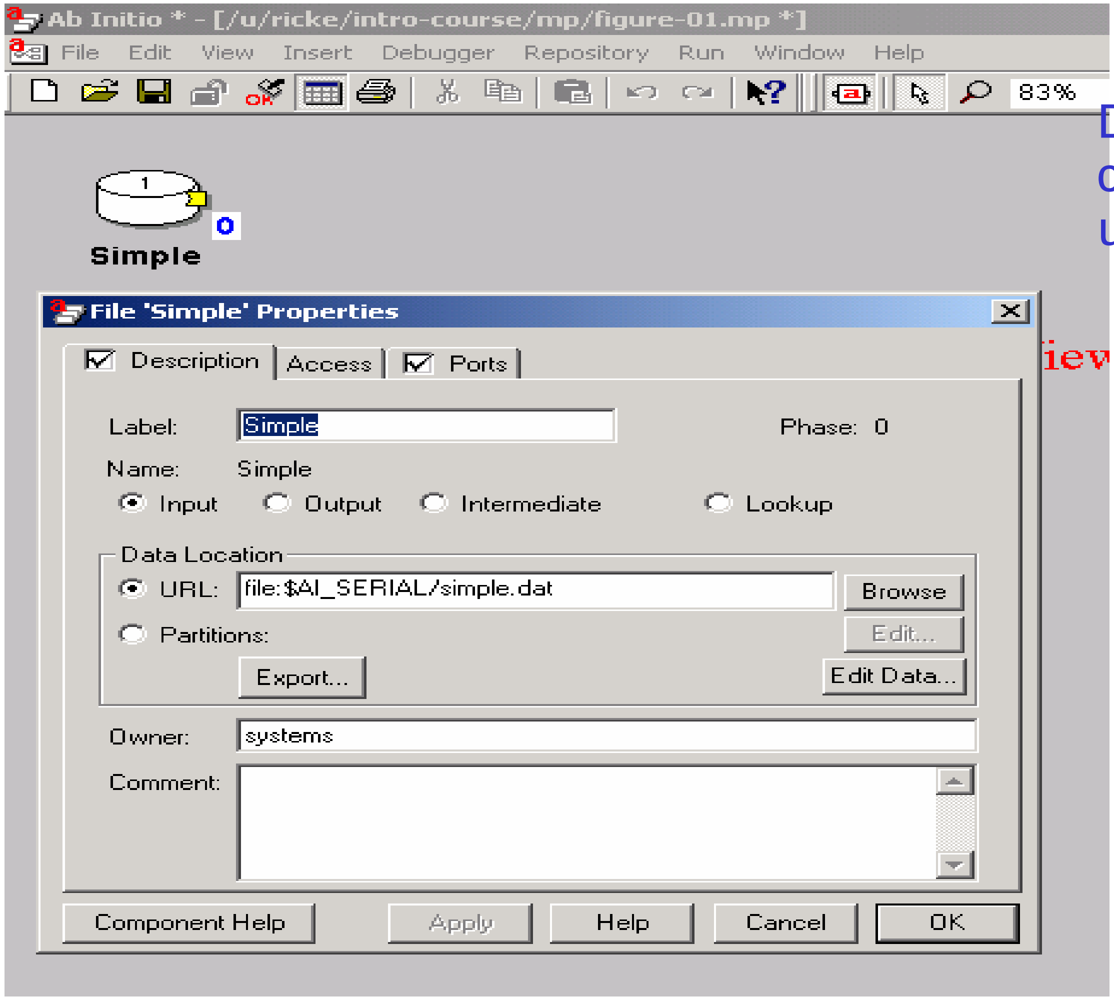
A sample korn shell script



An application script Shown below is the entire script (**classify-customers.mp**) needed to implement the customer-classification program we have been examining:

```
#! /bin/ksh
set -e

mp job classify-customers


#   Component        Name              Arguments
#   ---------        ------            ---------
mp ifile   customers       file:customers
mp ofile   telesales     file:telesales
mp ofile   mailing-list    mfile:/mfs/mlist

mp hash-partition partition     "zipcode" -layout customers
mp classifier classify        -layout mailing-list
mp householder household      -layout mailing-list
mp gather gather      -layout telesales


#   Define record format
mp metadata        format           -file customers.dml


#   Flow type     Name From Port       To Port           Data format
#   ---------     ---- ---- ----       -------           -----------
mp straight-flow f1   customers.read partition.in      -metadata format
mp fan-out-flow  f2   partition.out  classify.input    -metadata format
mp straight-flow f3   classify.mail  household.in      -metadata format
mp straight-flow f4   household.out  mailing-list.write -metadata format
mp straight-flow f5   classify.phone gather.input      -metadata format
mp fan-in-flow   f6   gather.output  telesales.write -metadata format

# Run the Application
mp run
```

Dataset Component Properties


Ab Initio * - [/u/ricke/intro-course/mp/figure-01.mp *]
File Edit View Insert Debugger Repository Run Window Help
83%
Simple
File 'Simple' Properties
Description Access Ports
Browse
Edit...
Export...
Edit Data...
Component Help Apply Help Cancel OK

Label: Simple Phase: 0

Name: Simple

[✓ Input] Output Intermediate Lookup

Data Location

[✓ URL:] file:$AI_SERIAL/simple.dat

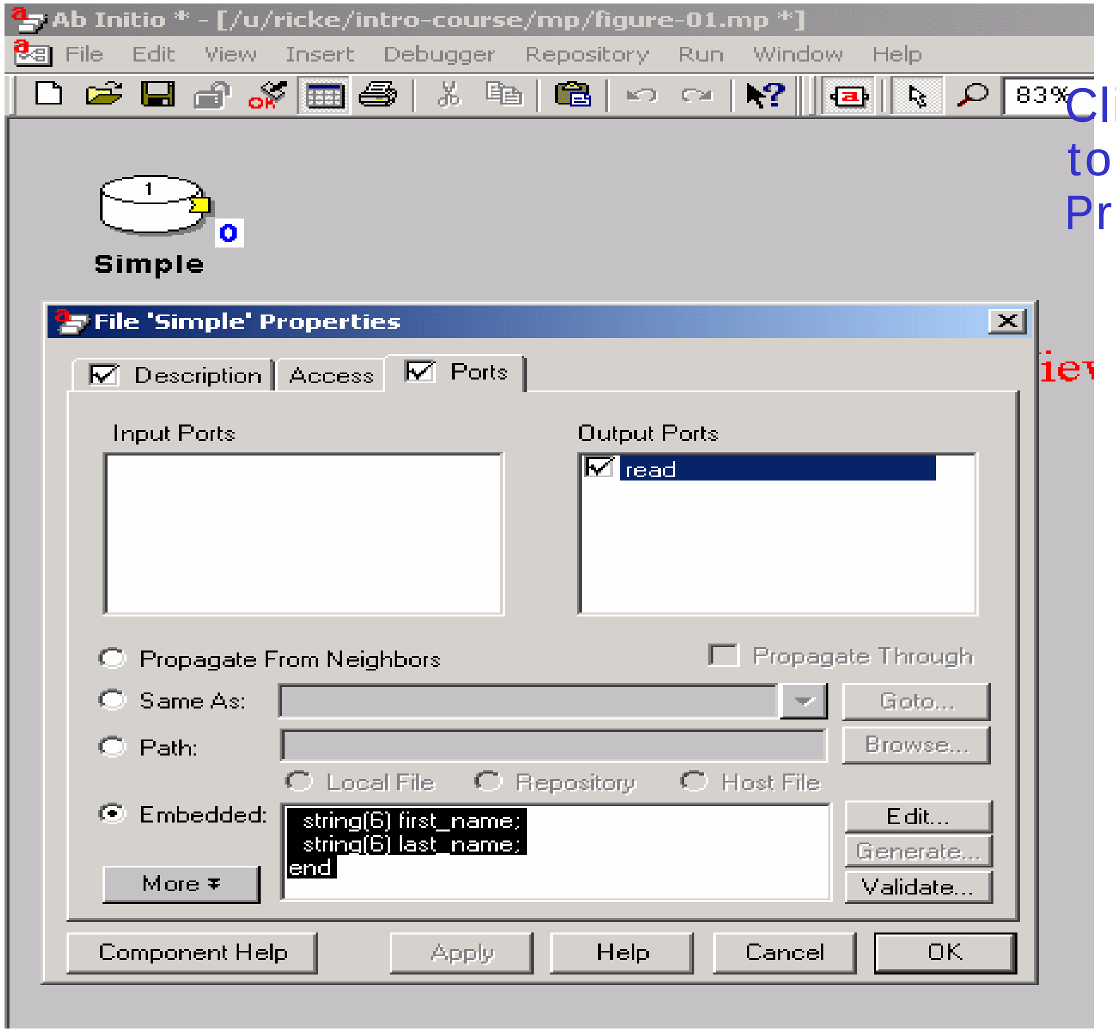
Partitions:

Owner: systems

Comment:

Double click on a component to bring up its Properties Page

Viewing Port Properties


Ab Initio * - [/u/ricke/intro-course/mp/figure-01.mp *]
File Edit View Insert Debugger Repository Run Window Help
83%
Simple
File 'Simple' Properties
Description
Access
Ports
Component Help
Apply
Help
Cancel
OK

Input Ports

Output Ports
- [✓] read

- Propagate From Neighbors
- [ ] Propagate Through
- Same As:
- Path:
- Local File / Repository / Host File
- [✓ Embedded:]

```
 string(6) first_name;
 string(6) last_name;
end
```

Click on the Ports Tab to view the 'Port(s)' Properties

DMLs and XFRs

- **DML**
  - **Ab Initio stores metadata in the form of record formats.**
  - **Metadata can be embedded within a component or can be stored external to the graph in a file with a “.dml” extension.**
- **XFR**
  - **Data can be transformed with the help of transform functions.**
  - **Transform functions can be embedded within a component or can be stored external to the graph in a file with a “.xfr” extension.**

Data Metadata Language or DML

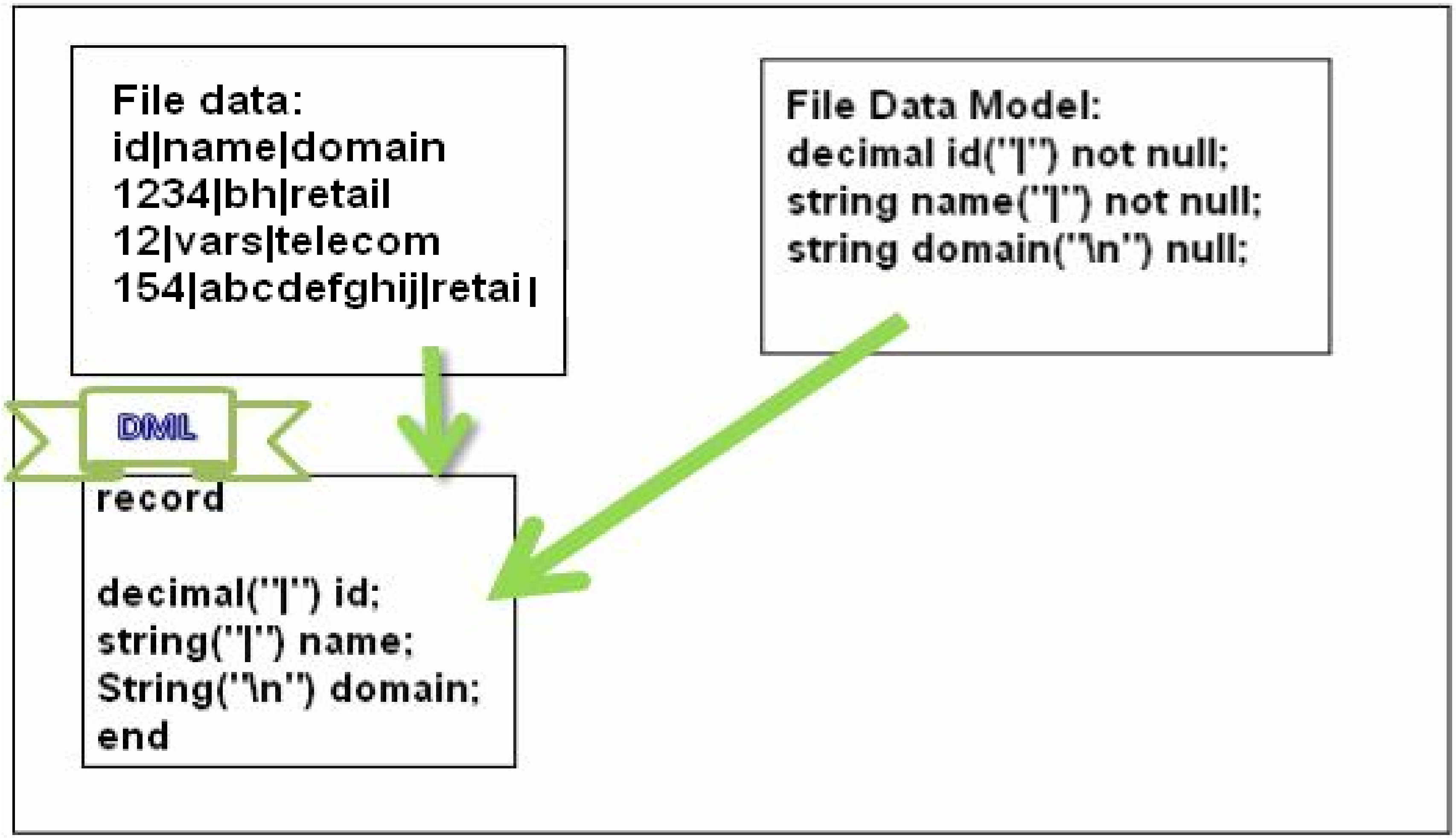
DML Syntax

- Record types begin with *record* and end with *end*
- Fields are declared: *data_type*(*len*) *field_name*;
- Field names consist of letters(a…z,A…Z),digits(0…9) and underscores(_) and are *Case sensitive*
- Keywords/Reserved words are *record*, *end*, *date*….

Some of the Data Types available

- String
- Decimal
- Integer
    - Storing Data in binary form
- Date and Datetime
- EBCDIC and ASCII records
- Null in Ab Initio - Non-existence of column values.

In Text view(special symbol as delimiter)

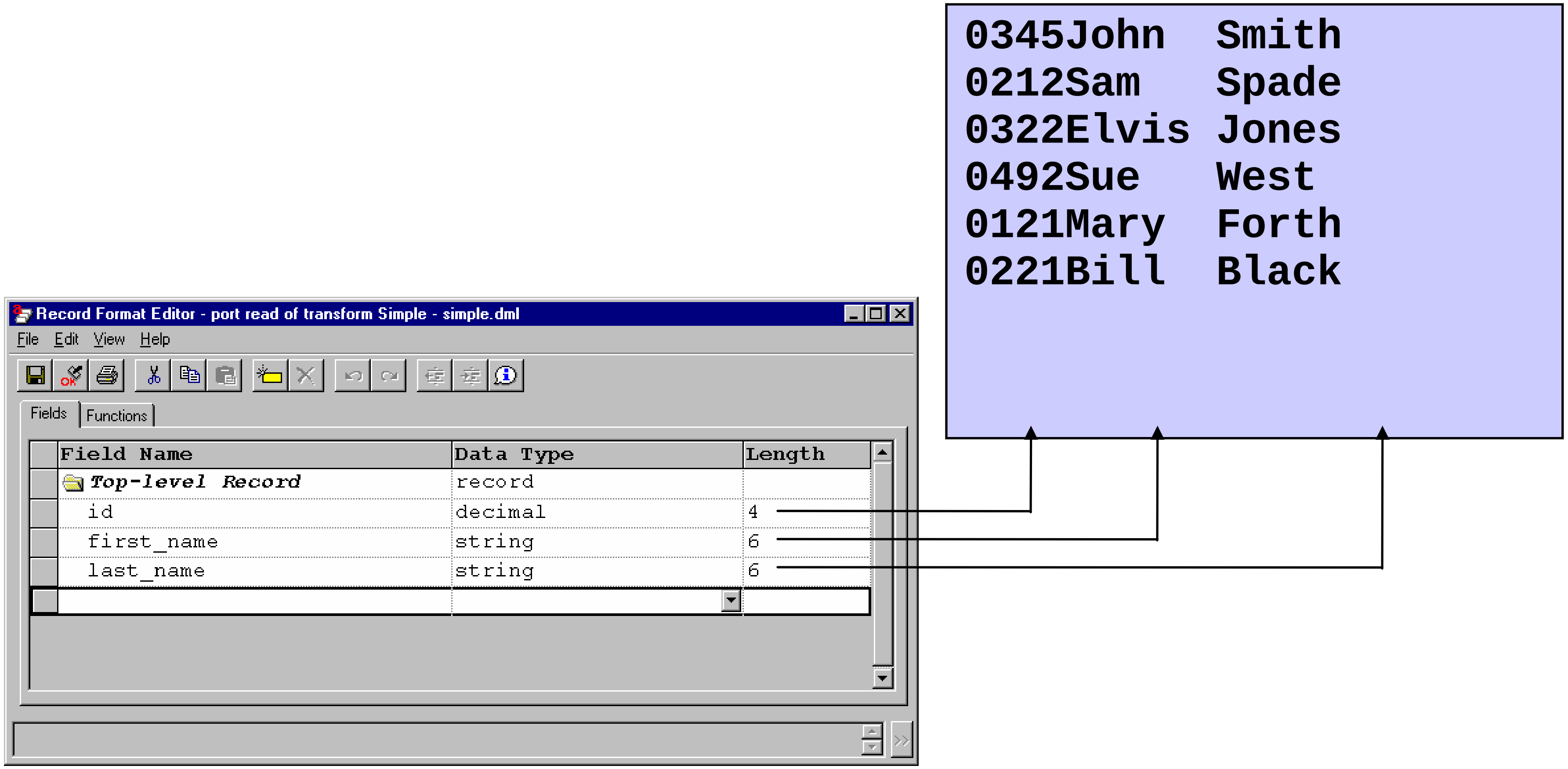

File data:
```
id|name|domain
1234|bh|retail
12|vars|telecom
154|abcdefghij|retail
```

File Data Model:
```
decimal id("|") not null;
string name("|") not null;
string domain("\n") null;
```

```
record

decimal("|") id;
string("|") name;
String("\n") domain;
end
```

Record format - In Graphical form(grid view)

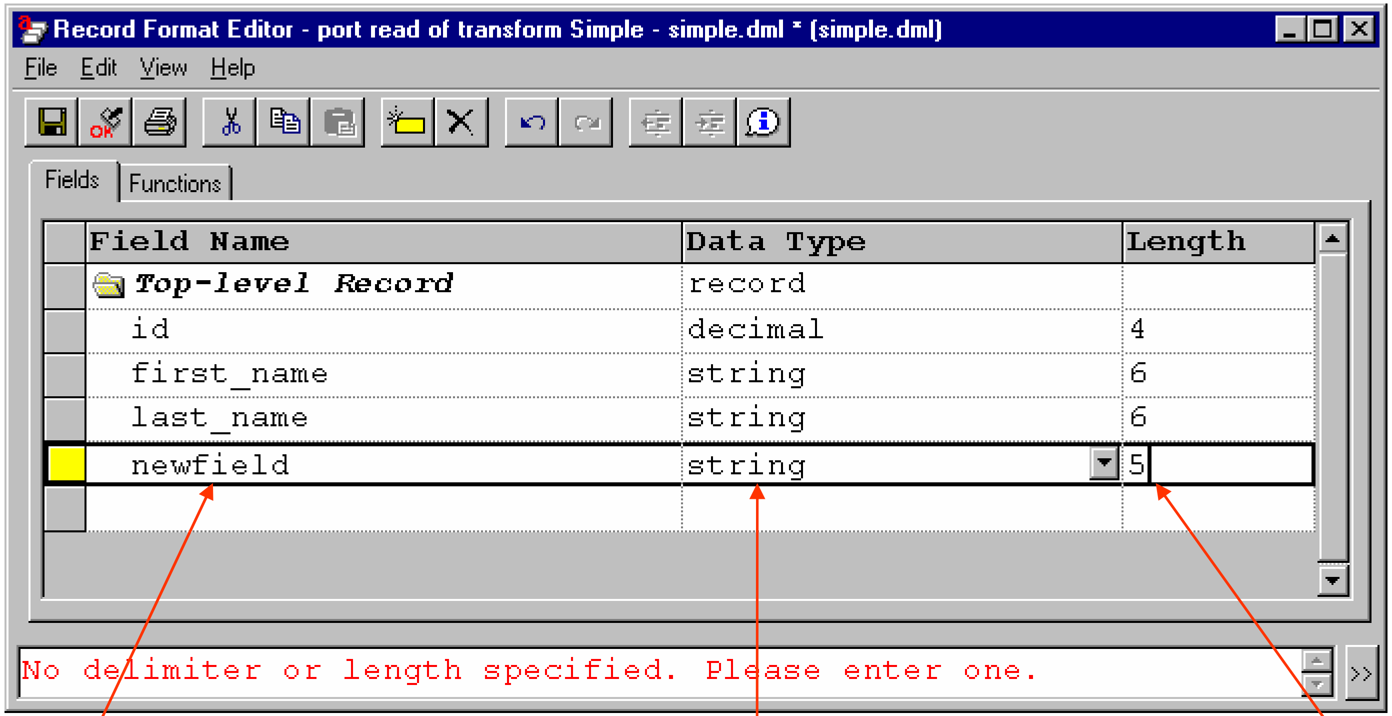
DML format created for a data


Record Format Editor - port read of transform Simple - simple.dml
File Edit View Help
Fields Functions

Field Name	Data Type	Length
Top-level Record	record	
id	decimal	4
first_name	string	6
last_name	string	6

```
0345John  Smith
0212Sam   Spade
0322Elvis Jones
0492Sue   West
0121Mary  Forth
0221Bill  Black
```

DML creation


Record Format Editor - port read of transform Simple - simple.dml * (simple.dml)
File Edit View Help
Fields Functions

Field Name	Data Type	Length
Top-level Record	record	
id	decimal	4
first_name	string	6
last_name	string	6
newfield	string	5

```
No delimiter or length specified. Please enter one.
```

Field name

Field type

Field length

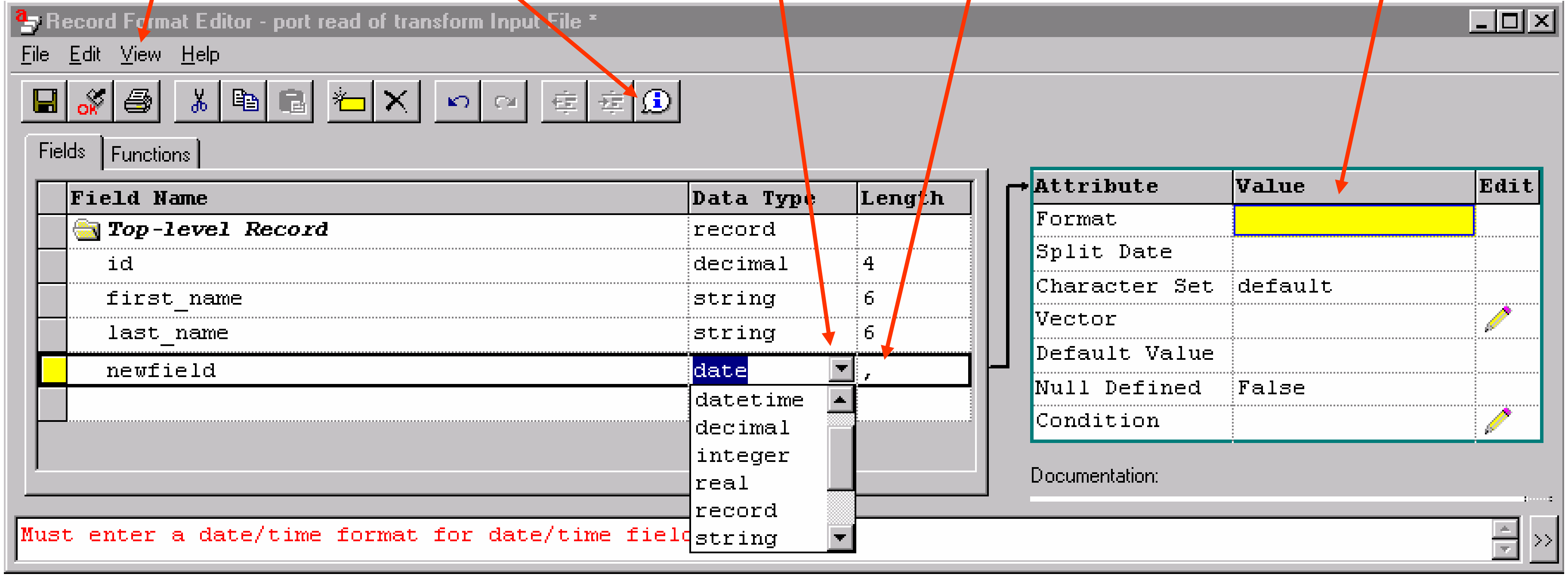
More Record Format Editing

View… Attributes.

Length can be delimiter string

Field Type drop-down

Date format goes here



	Field Name	Data Type	Length
	Top-level Record	record	
	id	decimal	4
	first_name	string	6
	last_name	string	6
🟨	newfield	date	,

Attribute	Value	Edit
Format	🟨	
Split Date		
Character Set	default	
Vector		✎
Default Value		
Null Defined	False	
Condition		✎

Must enter a date/time format for date/time field

Auto DML creation in Table component

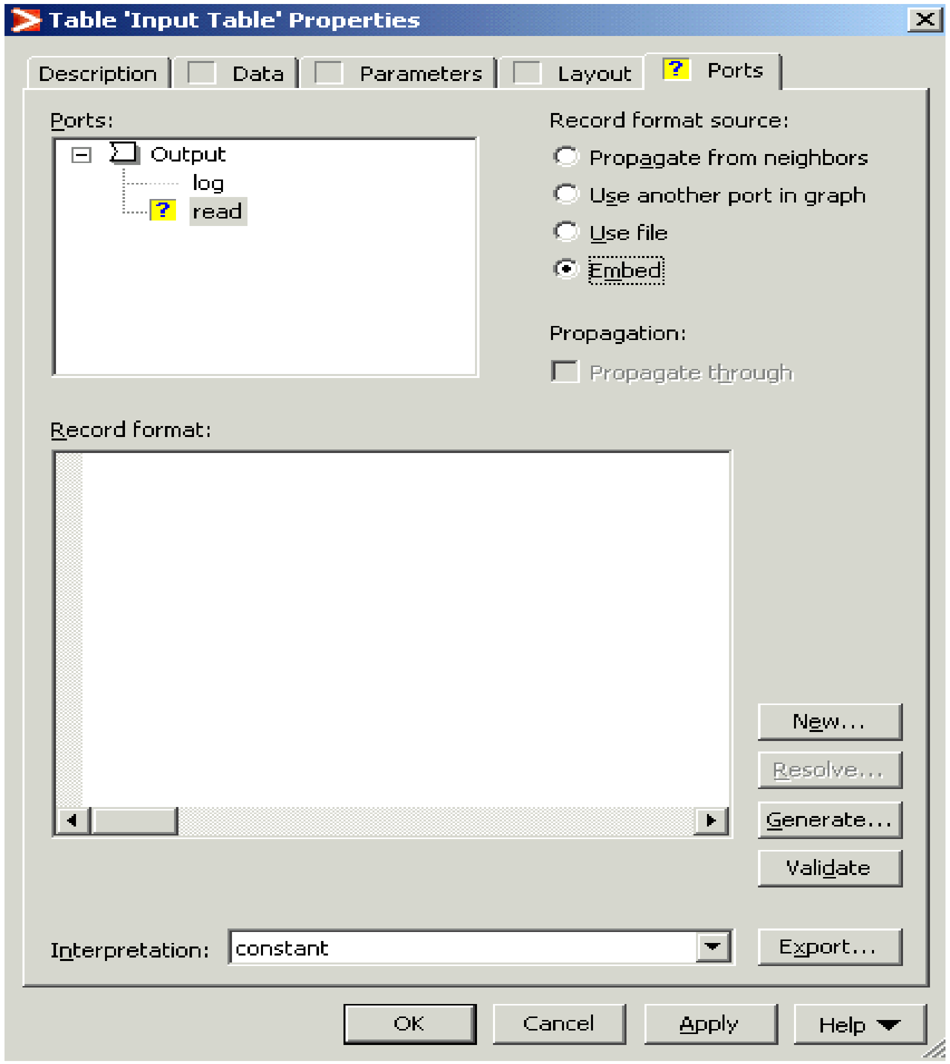


Table 'Input Table' Properties

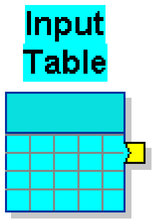
Ports:
- Output
  - log
  - read

Record format source:
- Propagate from neighbors
- Use another port in graph
- Use file
- [✓ Embed]

Propagation:
- Propagate through

Record format:

Interpretation: constant



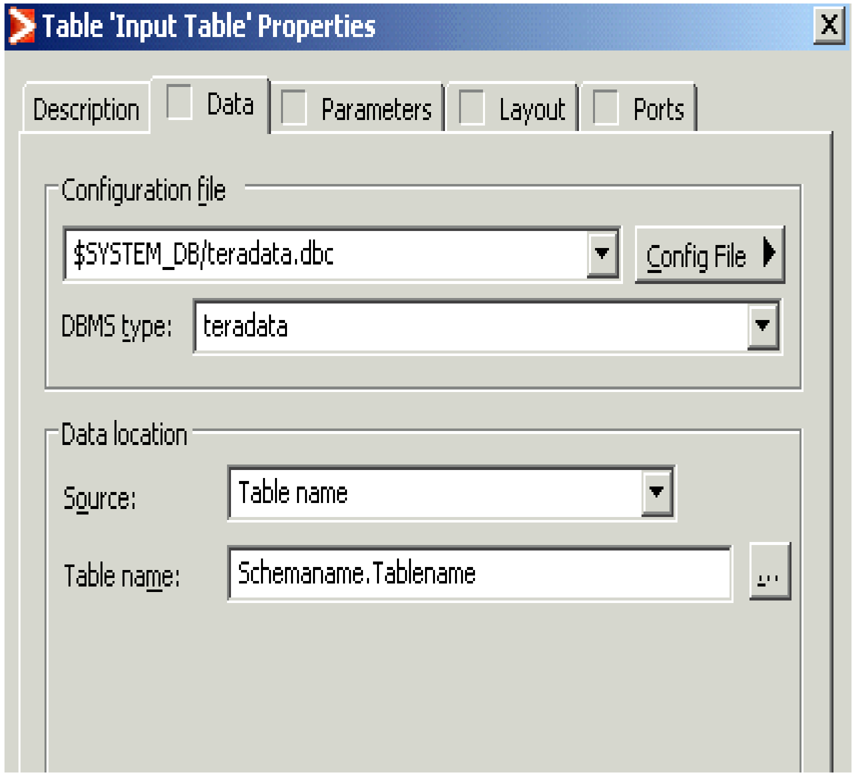


Table 'Input Table' Properties

Configuration file

$SYSTEM_DB/teradata.dbc

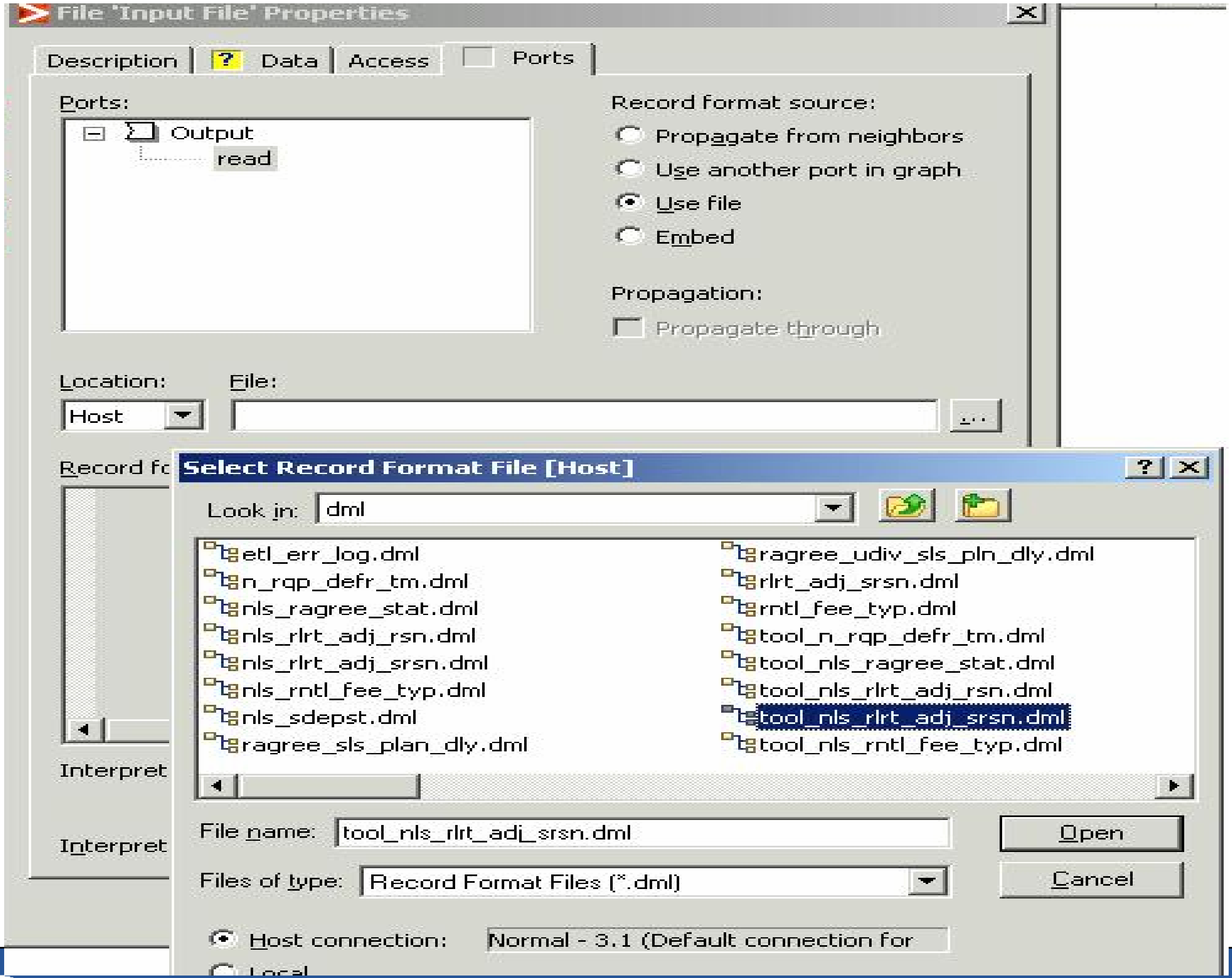
DBMS type: teradata

Data location

Source: Table name

Table name: Schemaname.Tablename

DML creation – Use file option in dataset



File 'Input File' Properties

Description | Data | Access | Ports

Ports:
Output
read

Record format source:
( ) Propagate from neighbors
( ) Use another port in graph
[✓ Use file]
( ) Embed

Propagation:
[ ] Propagate through

Location: Host
File:

Record fo

Interpret

Interpret

Select Record Format File [Host]

Look in: dml

etl_err_log.dml
n_rqp_defr_tm.dml
nls_ragree_stat.dml
nls_rlrt_adj_rsn.dml
nls_rlrt_adj_srsn.dml
nls_rntl_fee_typ.dml
nls_sdepst.dml
ragree_sls_plan_dly.dml
ragree_udiv_sls_pln_dly.dml
rlrt_adj_srsn.dml
rntl_fee_typ.dml
tool_n_rqp_defr_tm.dml
tool_nls_ragree_stat.dml
tool_nls_rlrt_adj_rsn.dml
tool_nls_rlrt_adj_srsn.dml
tool_nls_rntl_fee_typ.dml

File name: tool_nls_rlrt_adj_srsn.dml
Files of type: Record Format Files (*.dml)

Open | Cancel

[✓ Host connection:] Normal - 3.1 (Default connection for
( ) Local

Transform Functions : XFRs

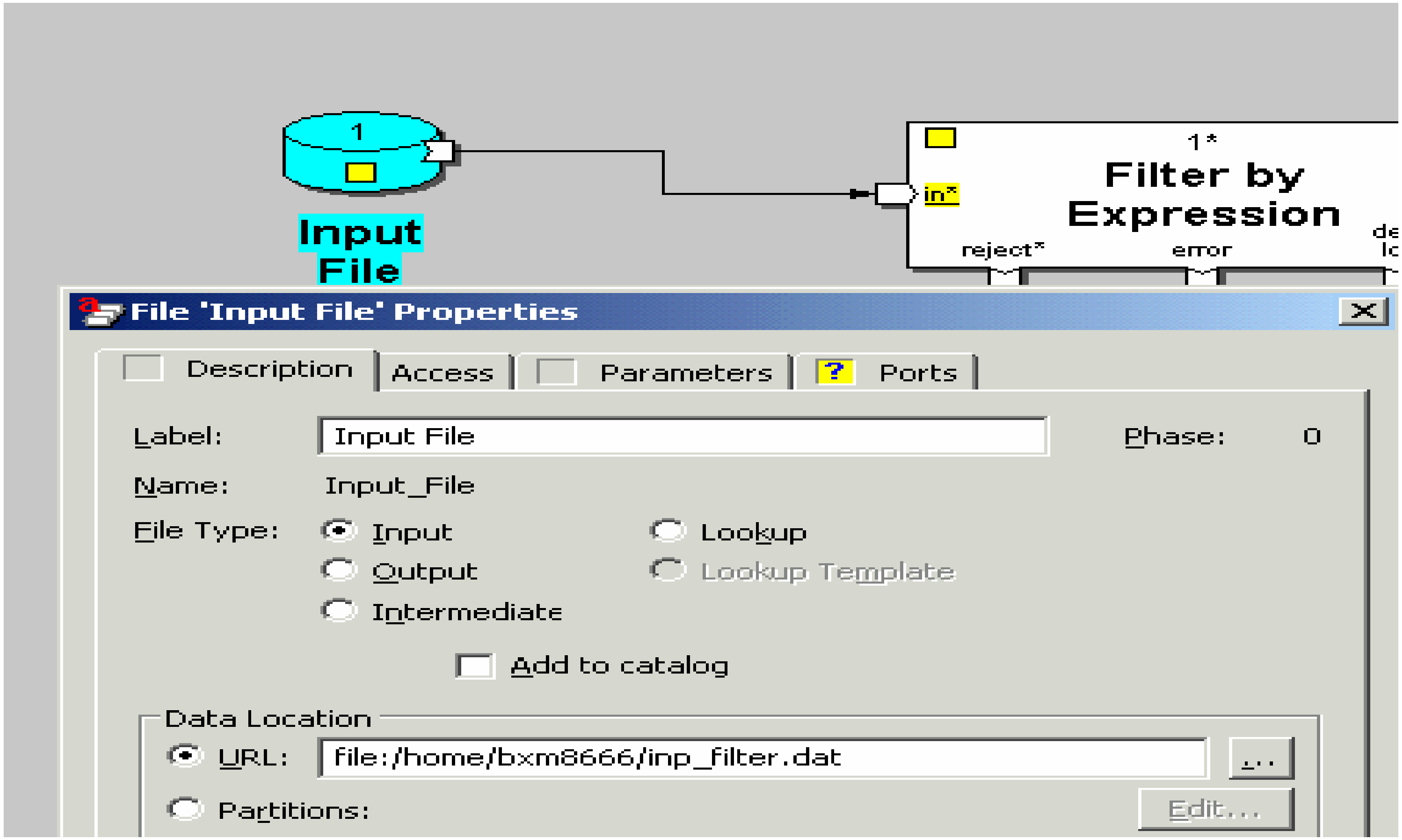
- User-defined function producing one or more output from one or more input
- Associated with transform components
- Rules that computes expression from input values and local variable and assigns the result to output objects
- Syntax
  - Functions :

```
output-records : : function-name (input-records) =
begin
assignments
End;
```

Assignments :

Direct Mapping without any transformation: out.* :: in.*

Input file settings



Label: Input File

Phase: 0

Name: Input_File

File Type: [✓ Input] Output Intermediate Lookup Lookup Template

Add to catalog

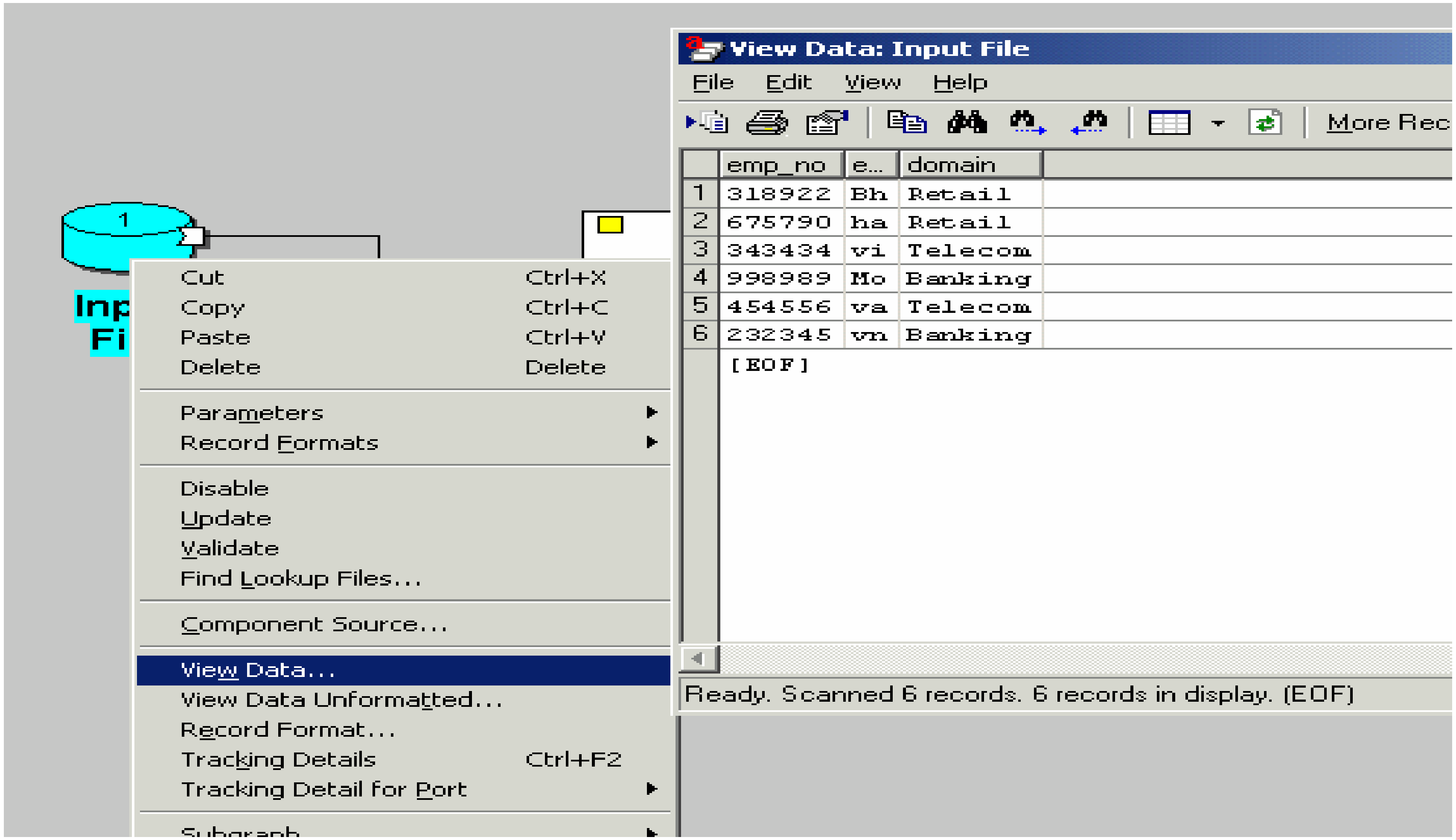
Data Location

[✓ URL:] file:/home/bxm8666/inp_filter.dat

Partitions:

Edit...

Input Data - Record View

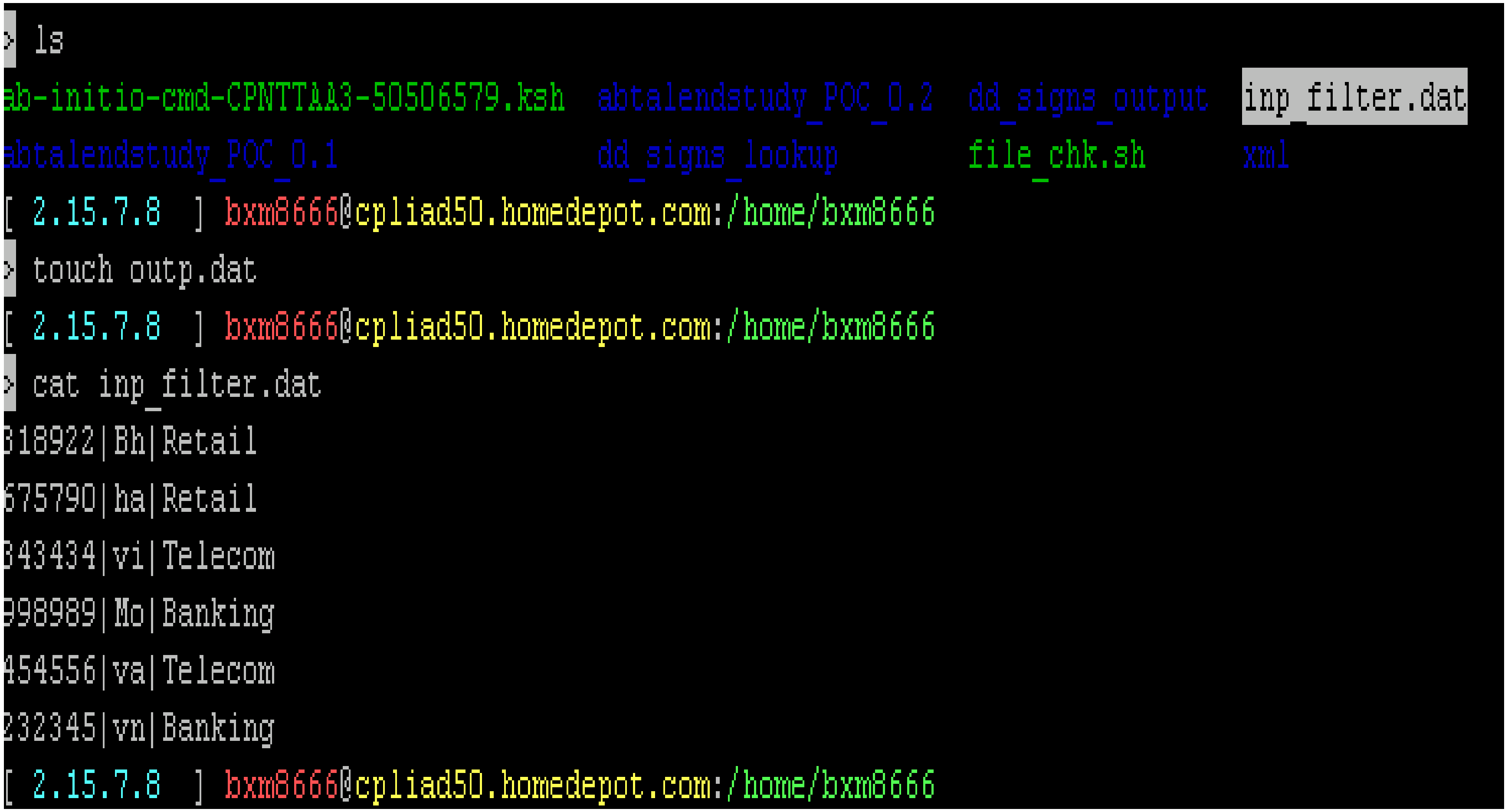


	emp_no	e...	domain
1	318922	Bh	Retail
2	675790	ha	Retail
3	343434	vi	Telecom
4	998989	Mo	Banking
5	454556	va	Telecom
6	232345	vn	Banking

```
[EOF]
```

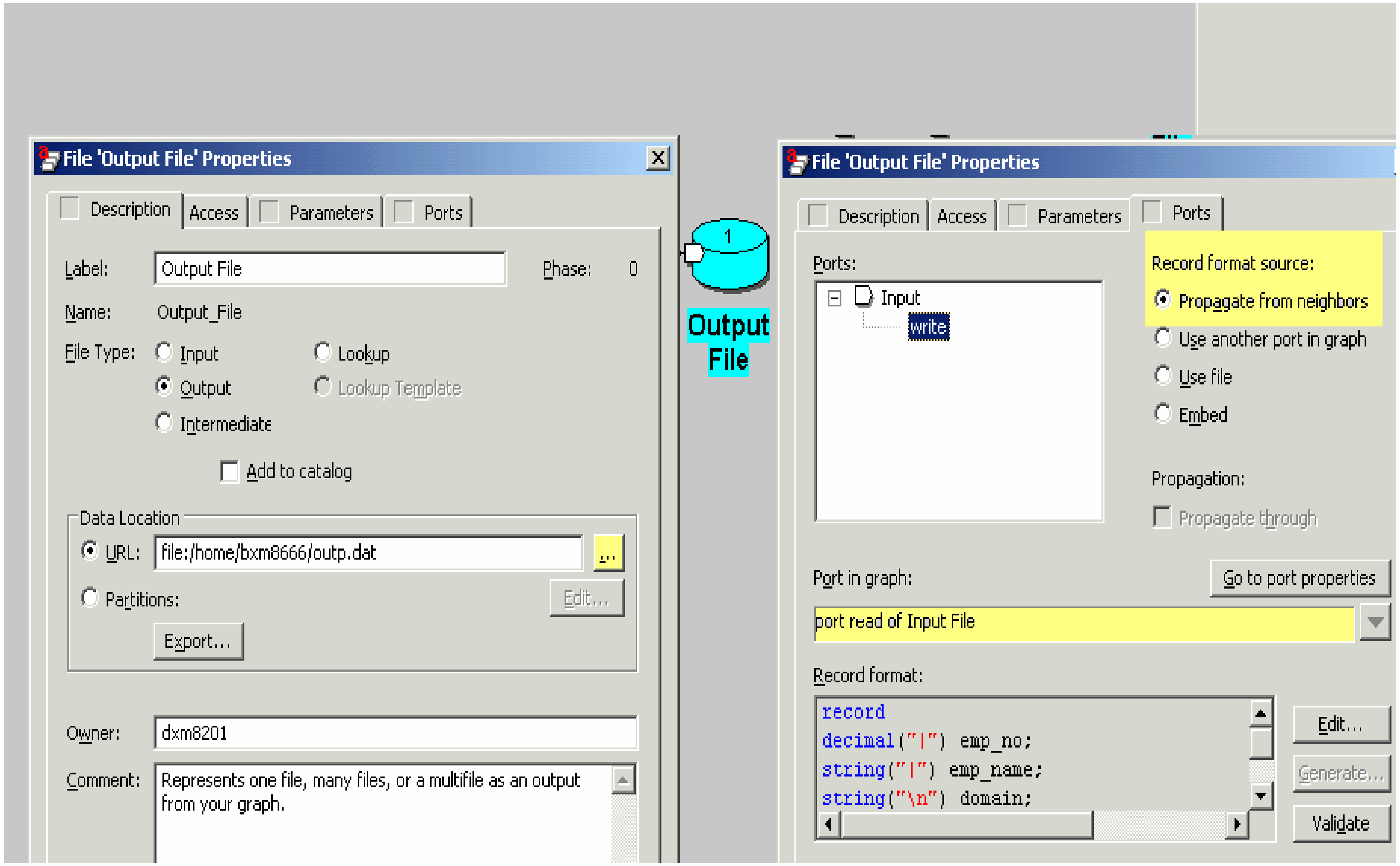
Ready. Scanned 6 records. 6 records in display. (EOF)

Input file View – Back end



```
> ls
ab-initio-cmd-CPNTTAA3-50506579.ksh  abtalendstudy_POC_0.2  dd_signs_output  inp_filter.dat
abtalendstudy_POC_0.1                dd_signs_lookup        file_chk.sh      xml
[ 2.15.7.8  ] bxm8666@cpliad50.homedepot.com:/home/bxm8666
> touch outp.dat
[ 2.15.7.8  ] bxm8666@cpliad50.homedepot.com:/home/bxm8666
> cat inp_filter.dat
318922|Bh|Retail
675790|ha|Retail
343434|vi|Telecom
998989|Mo|Banking
454556|va|Telecom
232345|vn|Banking
[ 2.15.7.8  ] bxm8666@cpliad50.homedepot.com:/home/bxm8666
```

Output – Settings(Propagating from input)



File 'Output File' Properties

Description | Access | Parameters | Ports

Label: Output File
Phase: 0
Name: Output_File
File Type: Input / [✓ Output] / Intermediate / Lookup / Lookup Template
Add to catalog

Data Location
[✓ URL:] file:/home/bxm8666/outp.dat
Partitions:
Edit...
Export...

Owner: dxm8201
Comment: Represents one file, many files, or a multifile as an output from your graph.

File 'Output File' Properties

Description | Access | Parameters | Ports

Ports:
Input
write

Record format source:
[✓ Propagate from neighbors]
Use another port in graph
Use file
Embed

Propagation:
Propagate through

Port in graph: port read of Input File
Go to port properties

Record format:

```
record
decimal("|") emp_no;
string("|") emp_name;
string("\n") domain;
```

Edit...
Generate...
Validate

Lookup File

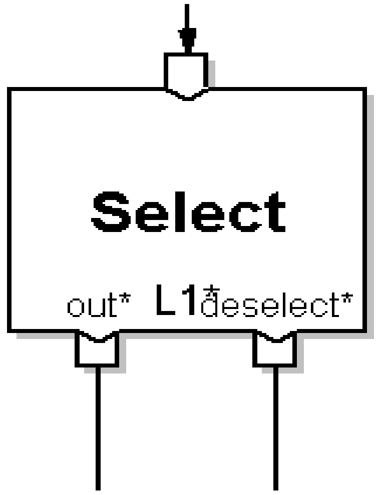
- Serial or Multifiles
- Held in main memory
- Searching and Retrieval is key-based and faster as compared to files stored on disks
- associates key values with corresponding data values to index records and retrieve them
- Lookup parameters
  - Key
  - Record Format

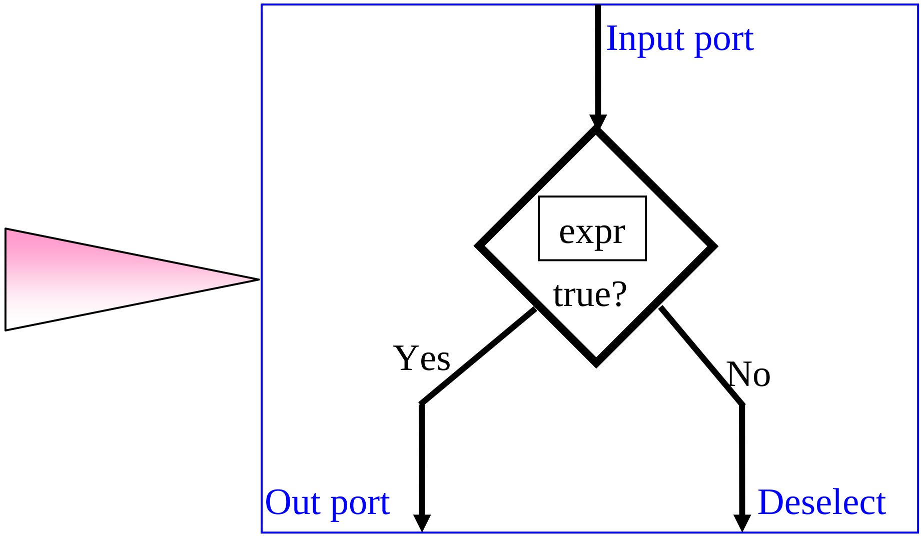
Basic Components

- ❑ Filter by Expression
- ❑ Reformat
- ❑ Redefine Format
- ❑ Sort
- ❑ Join
- ❑ Replicate
- ❑ Dedup
- ❑ Rollup

Filter by Expression

- Reads record from *input* port
- Evaluate the *select_expr*
- If result is true, record written to *out* port
- If result is false, record written to *deselect* port

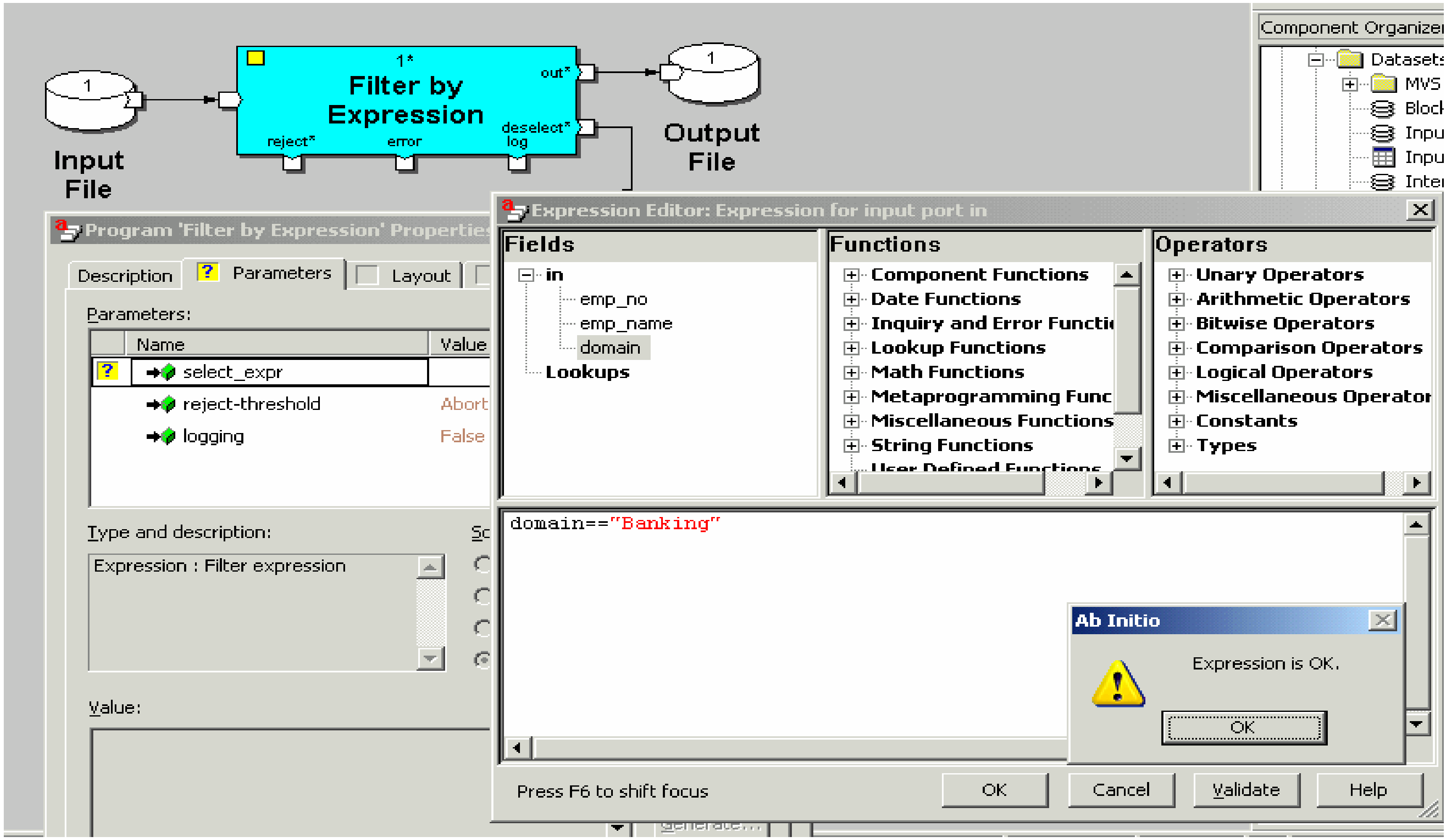




Diagnostic Ports

- REJECT
  - Input records that caused error
- ERROR
  - Associated error message
- LOG
  - Logging records

Filter by Expression



Parameters:

Name	Value
select_expr	
reject-threshold	Abort
logging	False

Type and description:

Expression : Filter expression

Value:

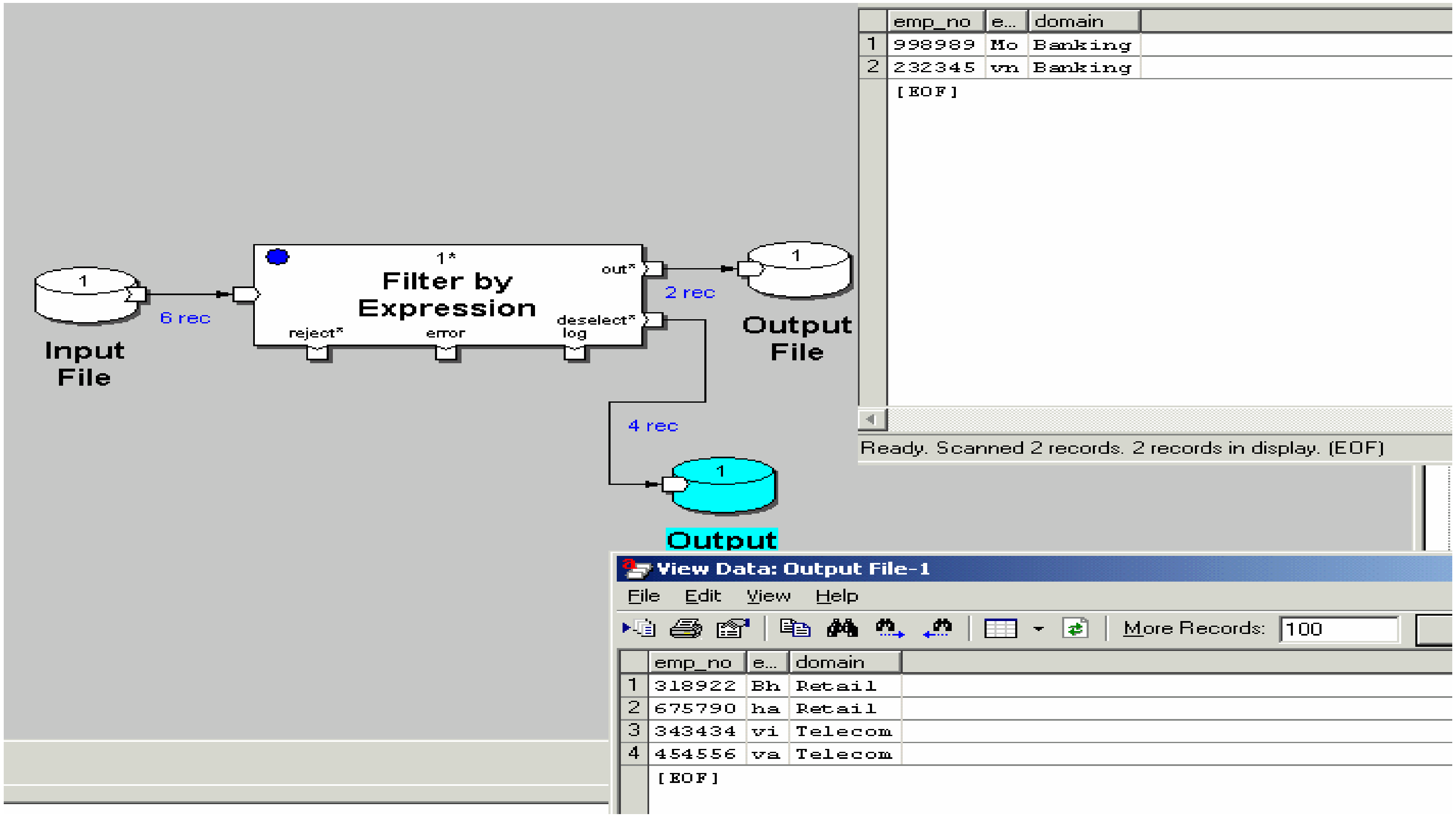
```
domain=="Banking"
```

Ab Initio

Expression is OK.

OK

Filtered output



	emp_no	e...	domain
1	998989	Mo	Banking
2	232345	vn	Banking

[EOF]

Ready. Scanned 2 records. 2 records in display. (EOF)

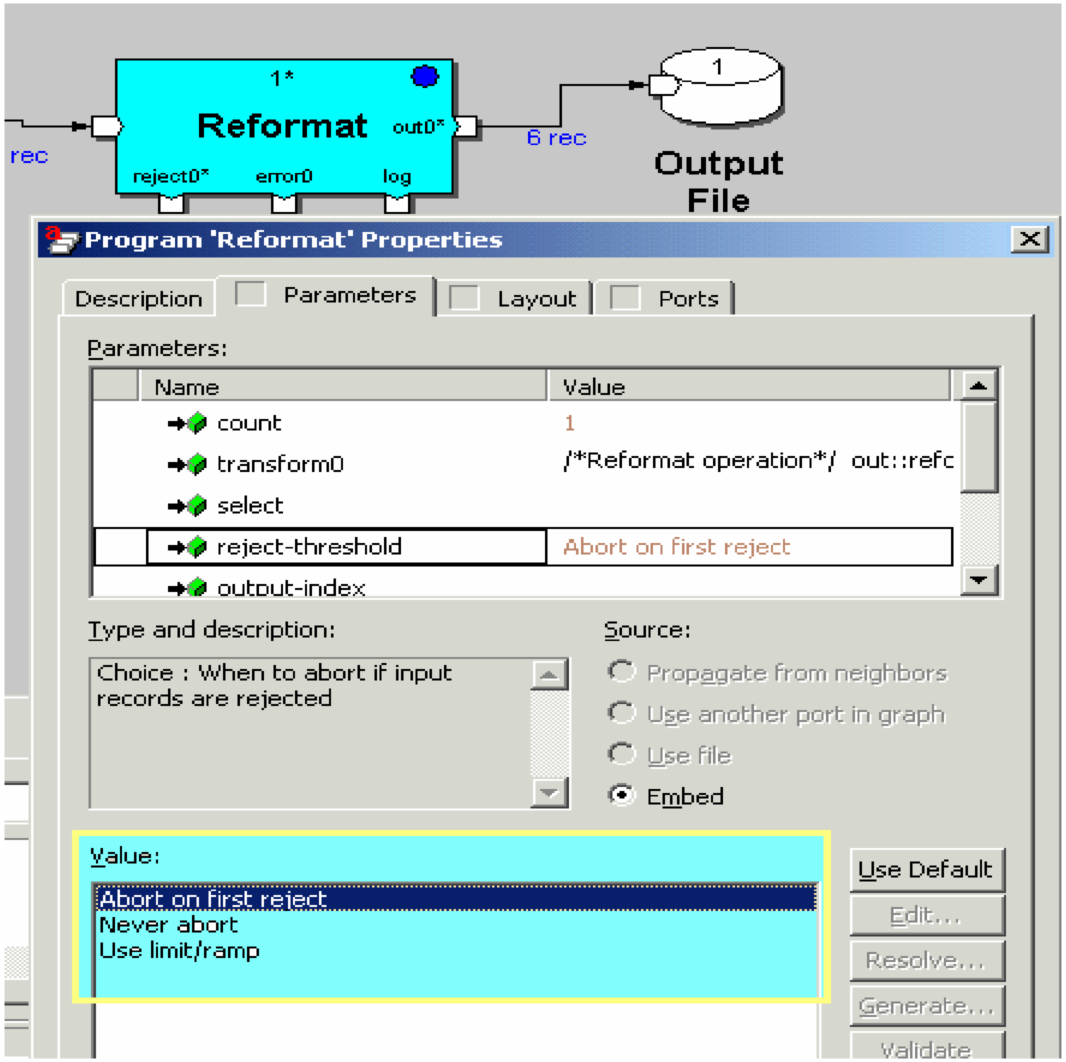
	emp_no	e...	domain
1	318922	Bh	Retail
2	675790	ha	Retail
3	343434	vi	Telecom
4	454556	va	Telecom

[EOF]

Reformat

1. Reads record from *input* port
2. Record passes as argument to *transform function or xfr*
3. Records written to *out* ports, if the function returns a success status
4. Records written to *reject* ports, if the function returns a failure status
5. Parameters of Reformat Component
   - Count
   - Transform (Xfr) Function
   - Reject-Threshold
     - Abort
     - Never Abort
     - Use Limit & Ramp
       - Limit – Number of errors to tolerate
       - Ramp – Scale of errors to tolerate per Input

Reformat-reject threshold



Program 'Reformat' Properties

Description | Parameters | Layout | Ports

Parameters:

Name	Value
count	1
transform0	/*Reformat operation*/ out::refc
select	
reject-threshold	Abort on first reject
output-index	

Type and description:

Choice : When to abort if input records are rejected

Source:

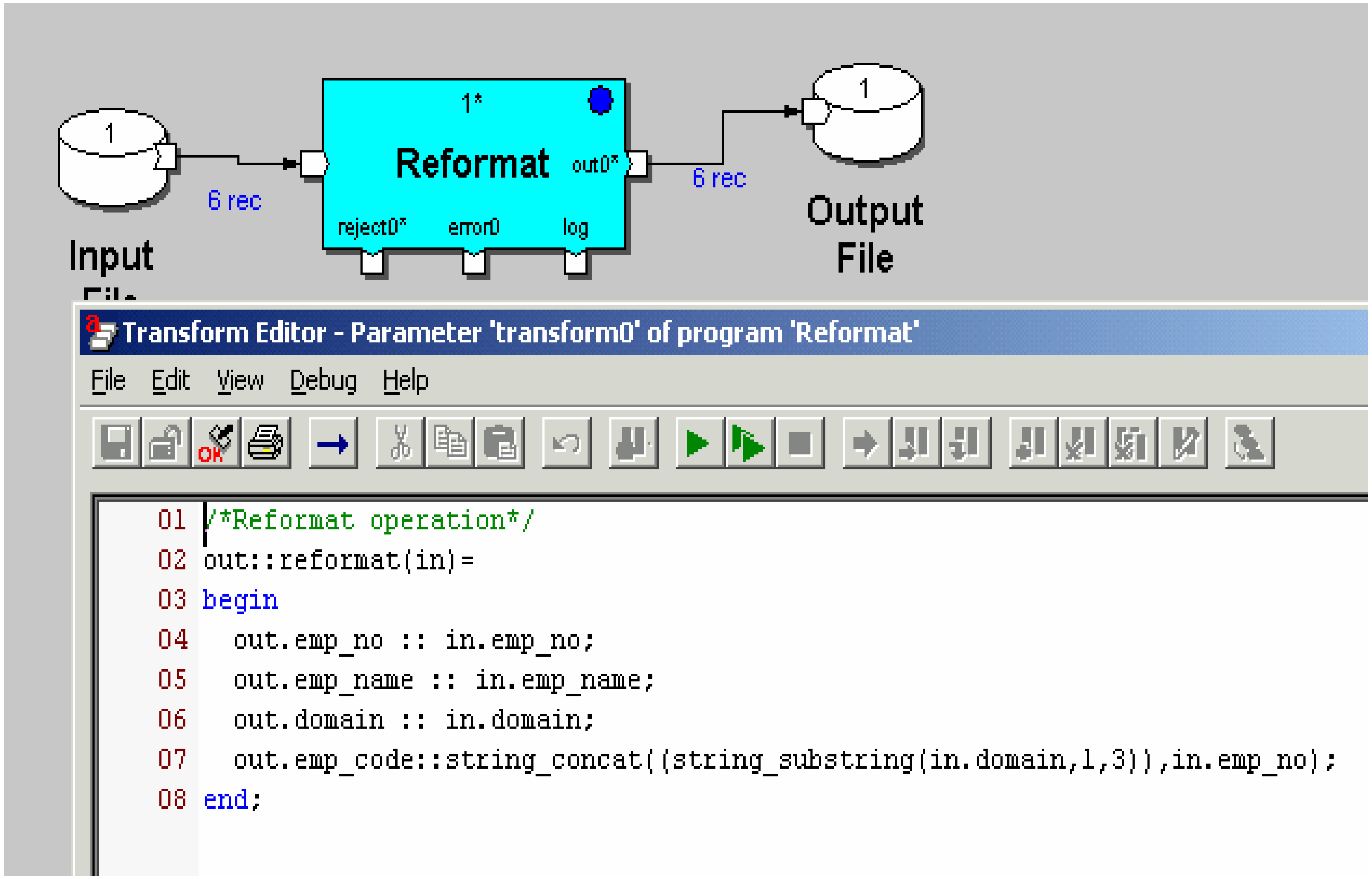
( ) Propagate from neighbors
( ) Use another port in graph
( ) Use file
(●) Embed

Value:

Abort on first reject
Never abort
Use limit/ramp

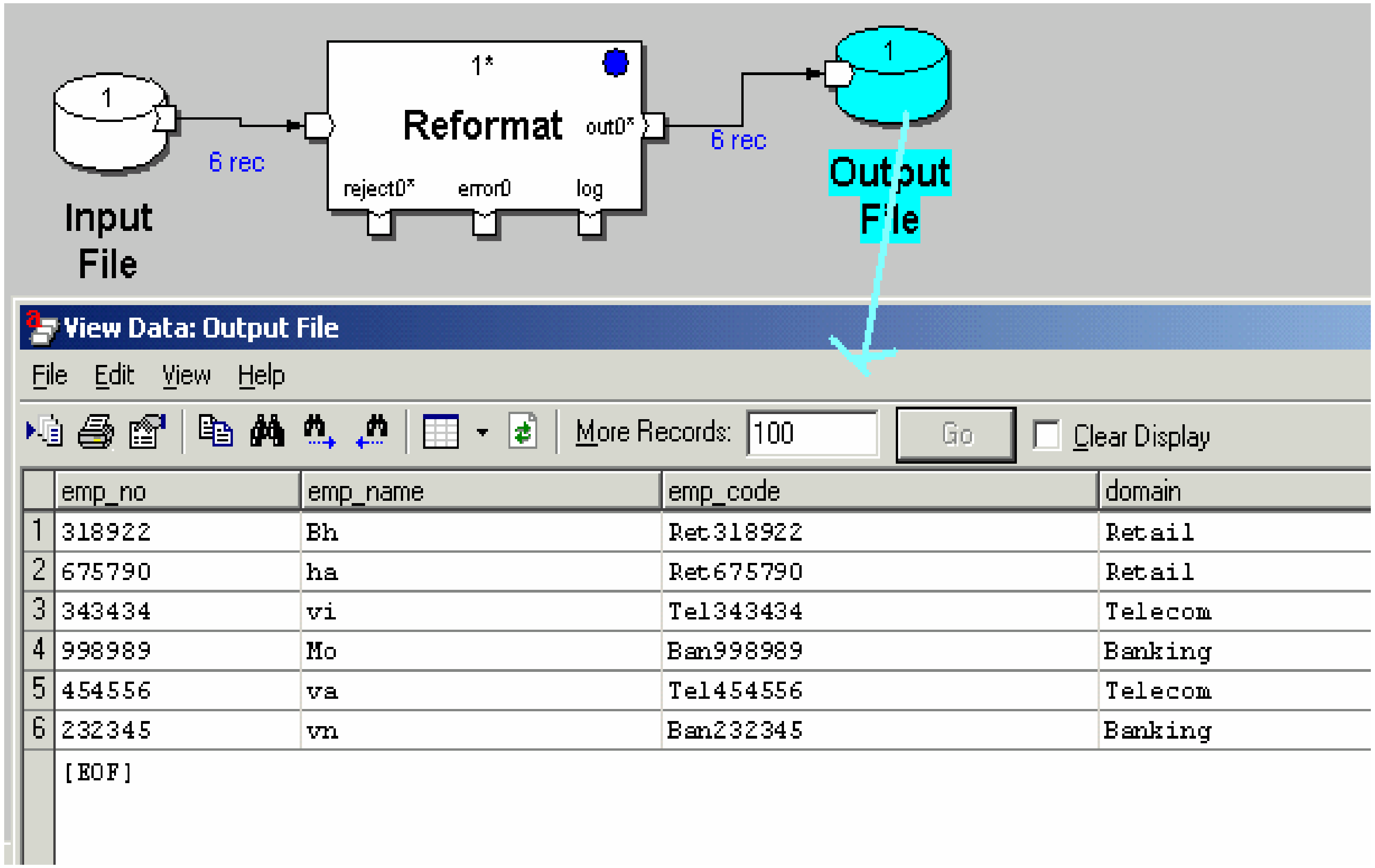
A drop-down menu specifying the number of errors to tolerate.

Transform functionality in Reformat


1
Input File
6 rec
1*
Reformat
out0*
reject0*
error0
log
6 rec
1
Output File
Transform Editor - Parameter 'transform0' of program 'Reformat'
File Edit View Debug Help

```
/*Reformat operation*/
out::reformat(in)=
begin
  out.emp_no :: in.emp_no;
  out.emp_name :: in.emp_name;
  out.domain :: in.domain;
  out.emp_code::string_concat((string_substring(in.domain,1,3)),in.emp_no);
end;
```

Reformatted output



	emp_no	emp_name	emp_code	domain
1	318922	Bh	Ret318922	Retail
2	675790	ha	Ret675790	Retail
3	343434	vi	Tel343434	Telecom
4	998989	Mo	Ban998989	Banking
5	454556	va	Tel454556	Telecom
6	232345	vn	Ban232345	Banking
	[EOF]			

Sort

Sort Component

- **Reads records from input port, sorts them by key, writes result to output port**
- **Parameters**
  - **Key**
  - **Max-core**

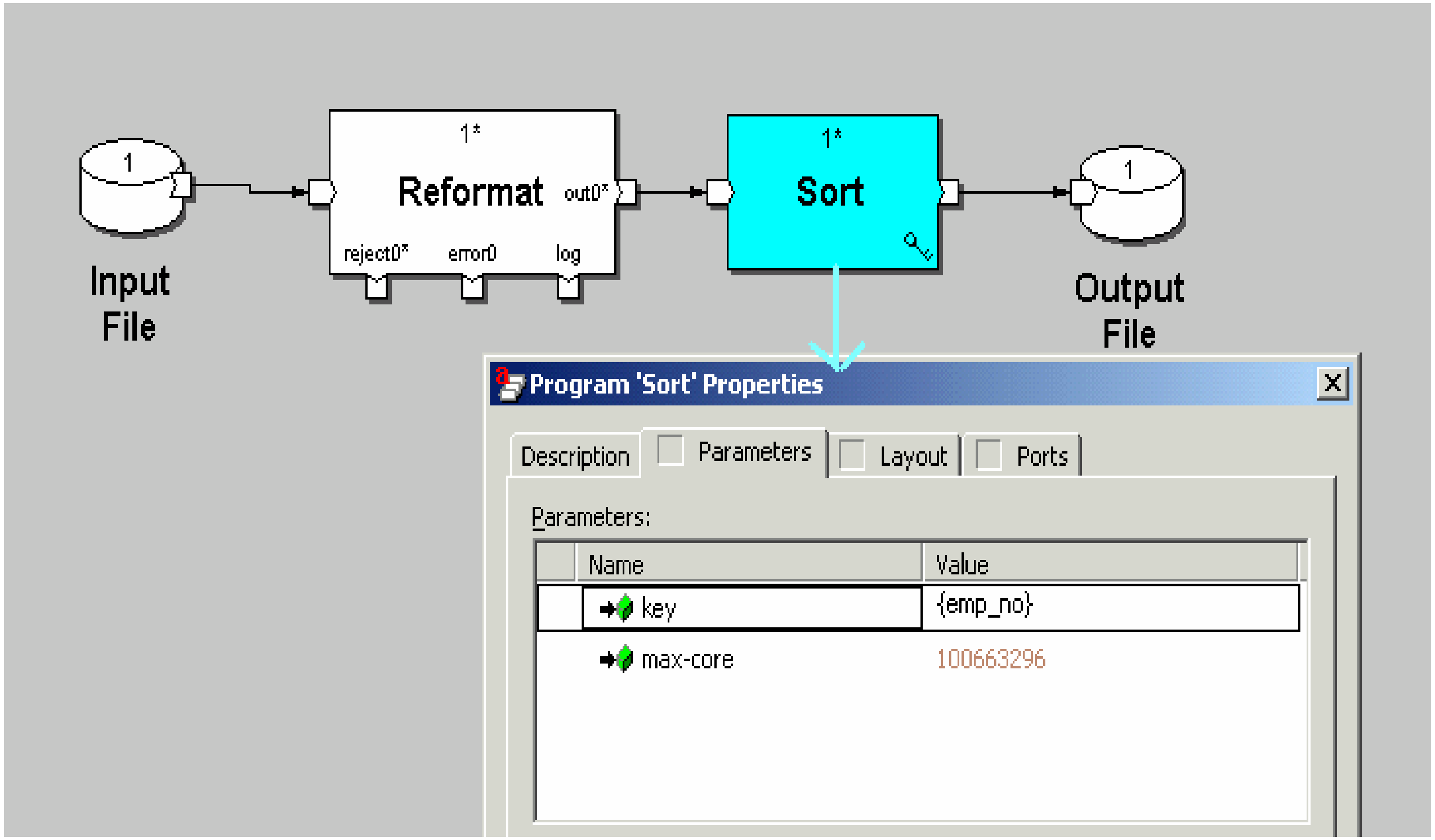
Keys

A key identifies a field or set of fields to organize a dataset

- **Single Field: *employee_number***
- **Multiple field or Composite key: *(last_name; first_name)***
- **Modifiers: *employee_number descending***

Max-core: Maximum memory usage in bytes

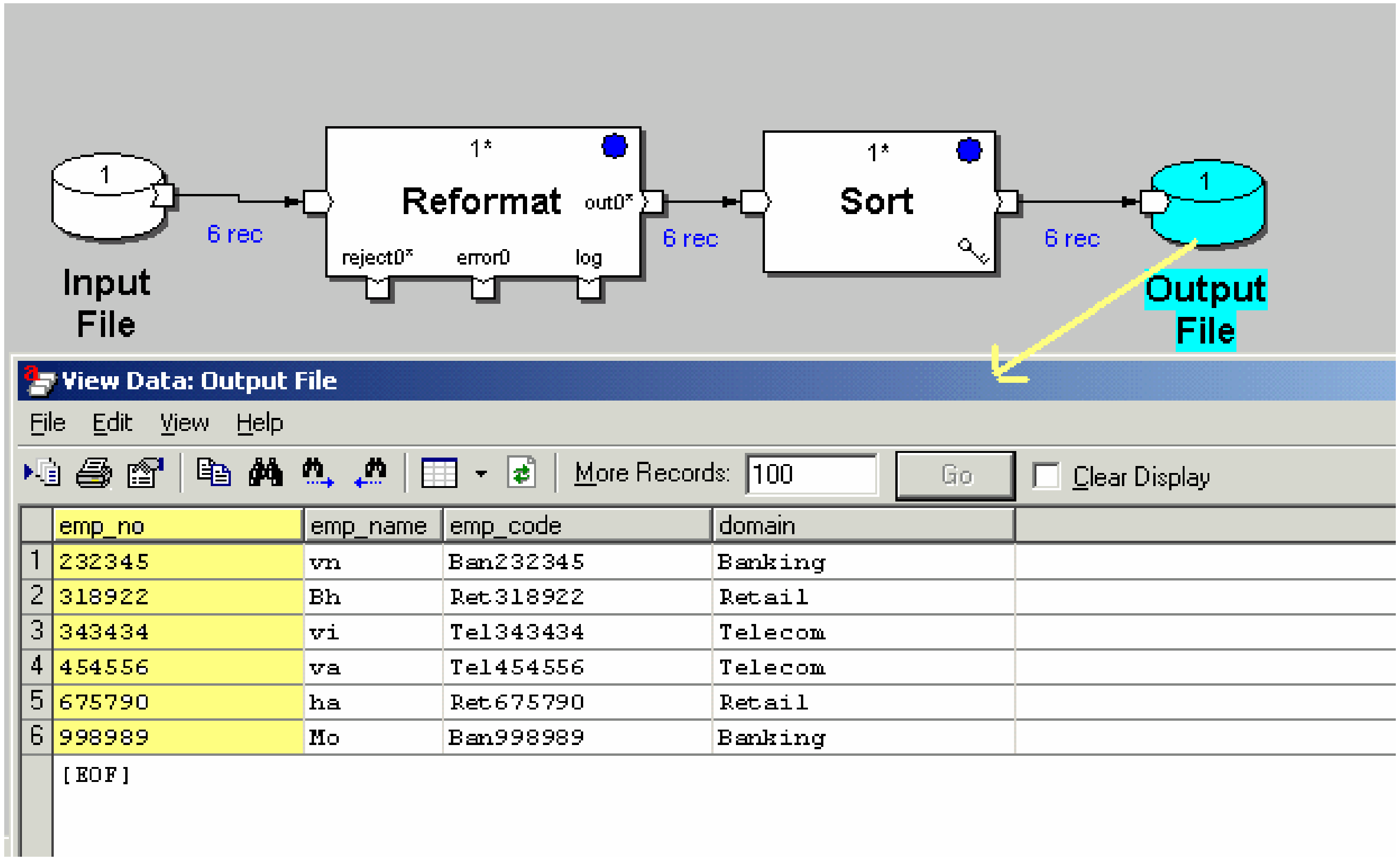
Sort Functionality



Parameters:

Name	Value
key	{emp_no}
max-core	100663296

Sorted output



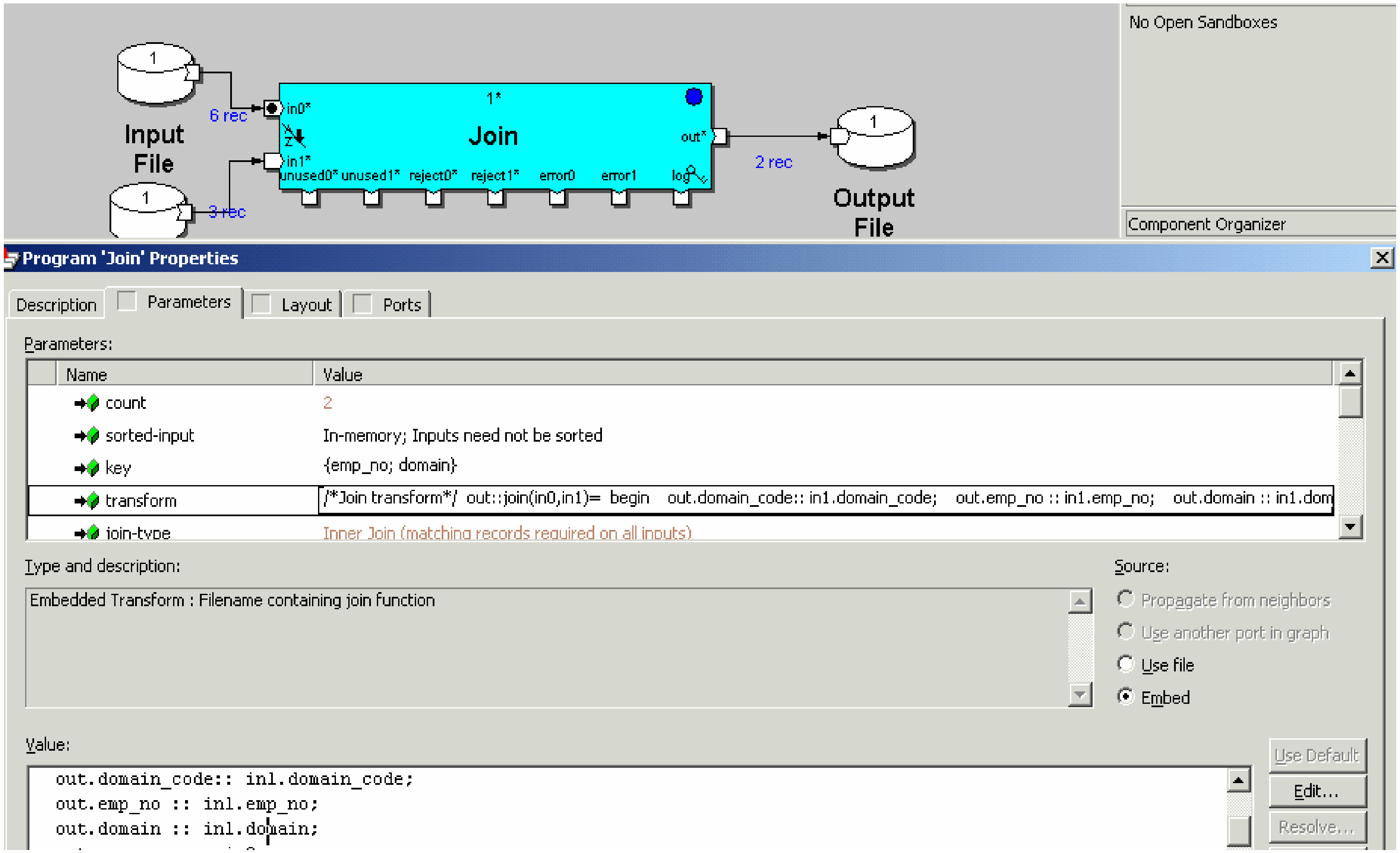
	emp_no	emp_name	emp_code	domain
1	232345	vn	Ban232345	Banking
2	318922	Bh	Ret318922	Retail
3	343434	vi	Tel343434	Telecom
4	454556	va	Tel454556	Telecom
5	675790	ha	Ret675790	Retail
6	998989	Mo	Ban998989	Banking
	[EOF]			

Join

1. Reads records from multiple input ports
2. Operates on records with matching keys using a multi-input transform function
3. Writes result to the output port

PORTS	***PARAMETERS***
▪in	▪count
▪out	▪key
▪unused	▪override key
▪reject (optional)	▪transform
▪error (optional)	▪limit
▪log (optional)	▪ramp

Join Parameters



Parameters:

Name	Value
count	2
sorted-input	In-memory; Inputs need not be sorted
key	{emp_no; domain}
transform	/*Join transform*/ out::join(in0,in1)= begin out.domain_code:: in1.domain_code; out.emp_no :: in1.emp_no; out.domain :: in1.dom
join-type	Inner Join (matching records required on all inputs)

Type and description:

Embedded Transform : Filename containing join function

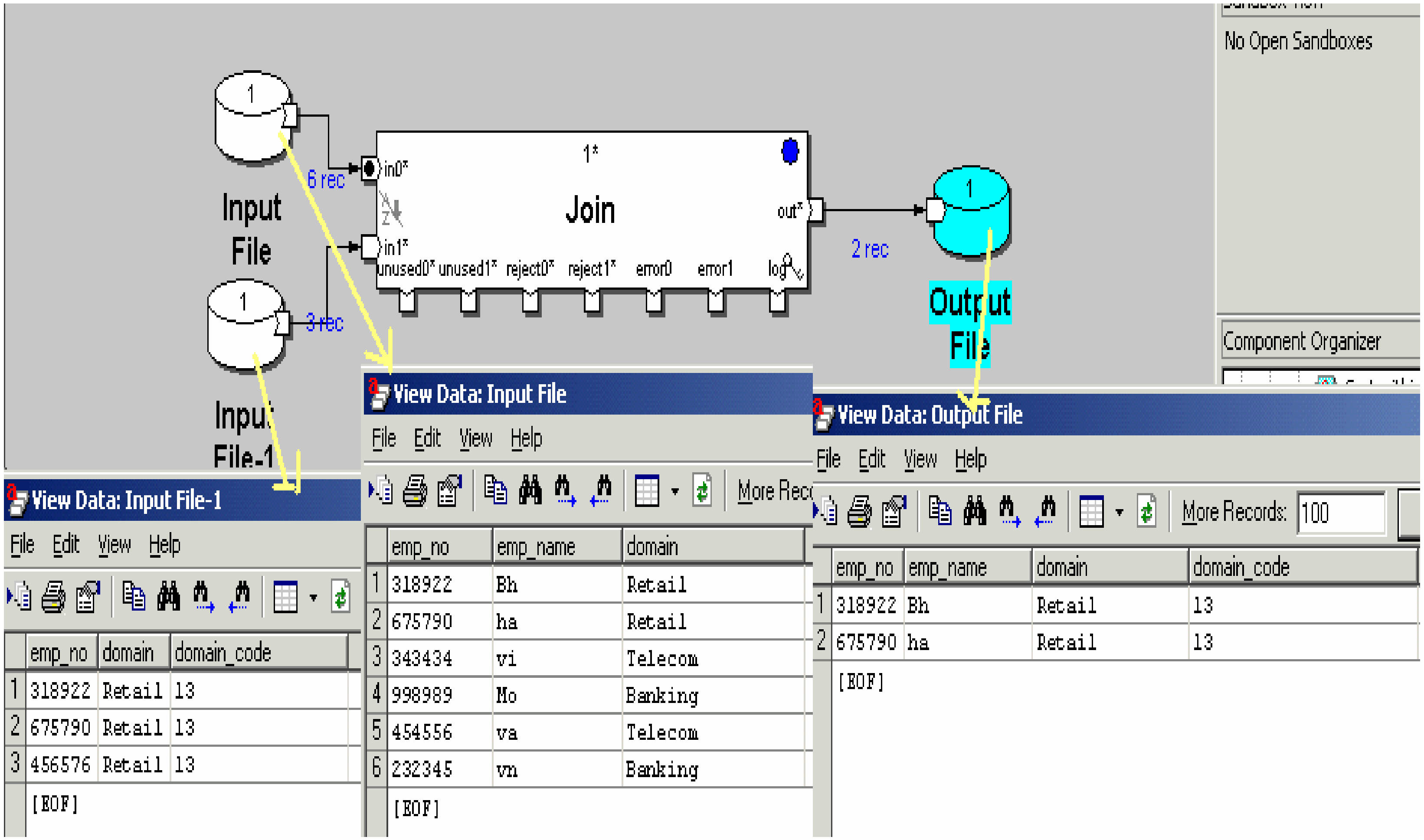
Source:

( ) Propagate from neighbors
( ) Use another port in graph
( ) Use file
[✓ Embed]

Value:

```
  out.domain_code:: in1.domain_code;
  out.emp_no :: in1.emp_no;
  out.domain :: in1.domain;
```

Joined output



View Data: Input File-1

	emp_no	domain	domain_code
1	318922	Retail	13
2	675790	Retail	13
3	456576	Retail	13
	[EOF]		

View Data: Input File

	emp_no	emp_name	domain
1	318922	Bh	Retail
2	675790	ha	Retail
3	343434	vi	Telecom
4	998989	Mo	Banking
5	454556	va	Telecom
6	232345	vn	Banking
	[EOF]		

View Data: Output File

	emp_no	emp_name	domain	domain_code
1	318922	Bh	Retail	13
2	675790	ha	Retail	13
	[EOF]			

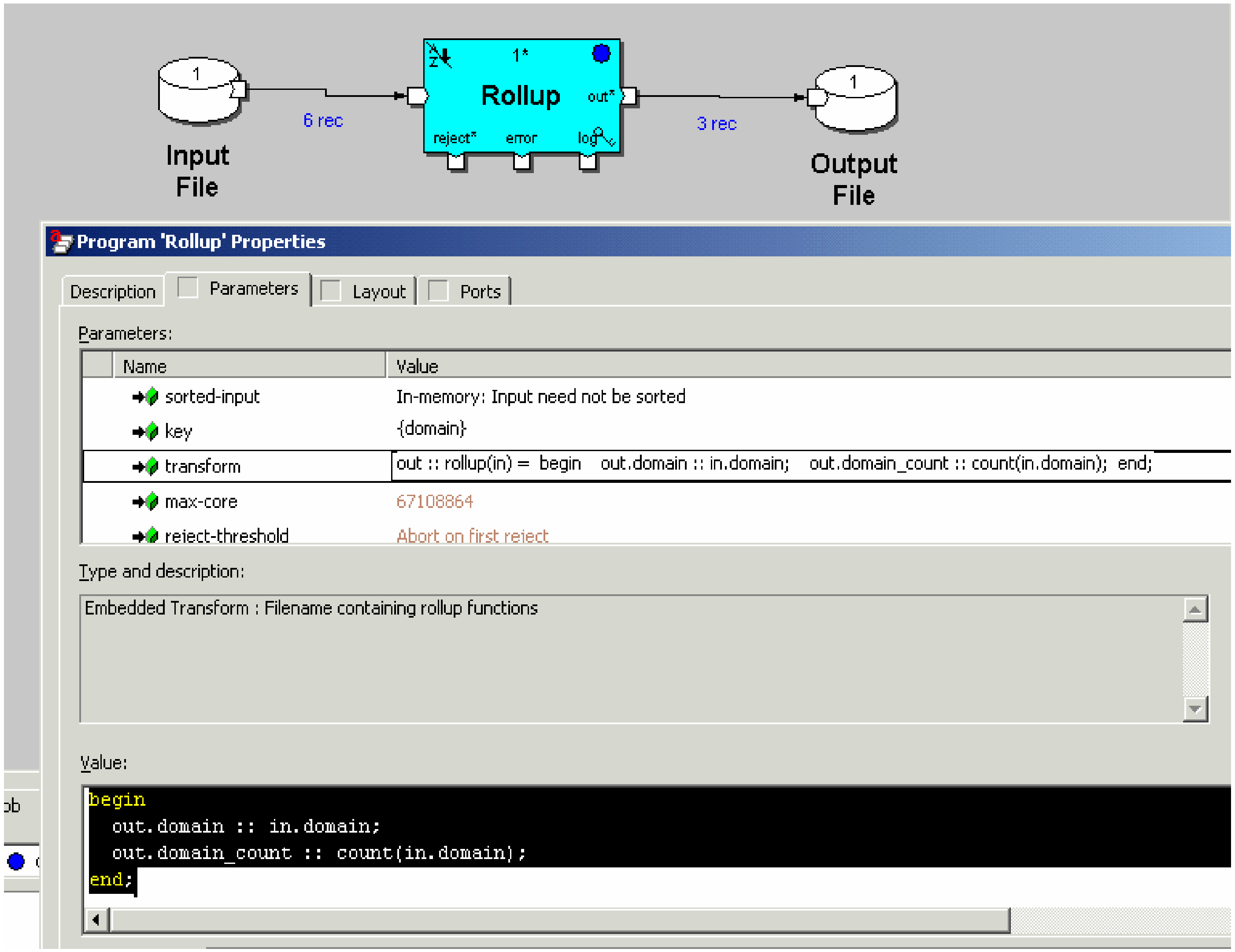
Rollup

Rollup evaluates a group of input records that have the same key, and then generates records that either summarize each group or select certain information from each group.

Parameters:

check-sort,sorted input	limit,Ramp
logging	log_group
log_input	log_intermediate
log_output	grouped-input
error_group	key
key-method	major-key
log_reject	max-core

Rollup - functionality



Parameters:

Name	Value
sorted-input	In-memory: Input need not be sorted
key	{domain}
transform	out :: rollup(in) = begin out.domain :: in.domain; out.domain_count :: count(in.domain); end;
max-core	67108864
reject-threshold	Abort on first reject

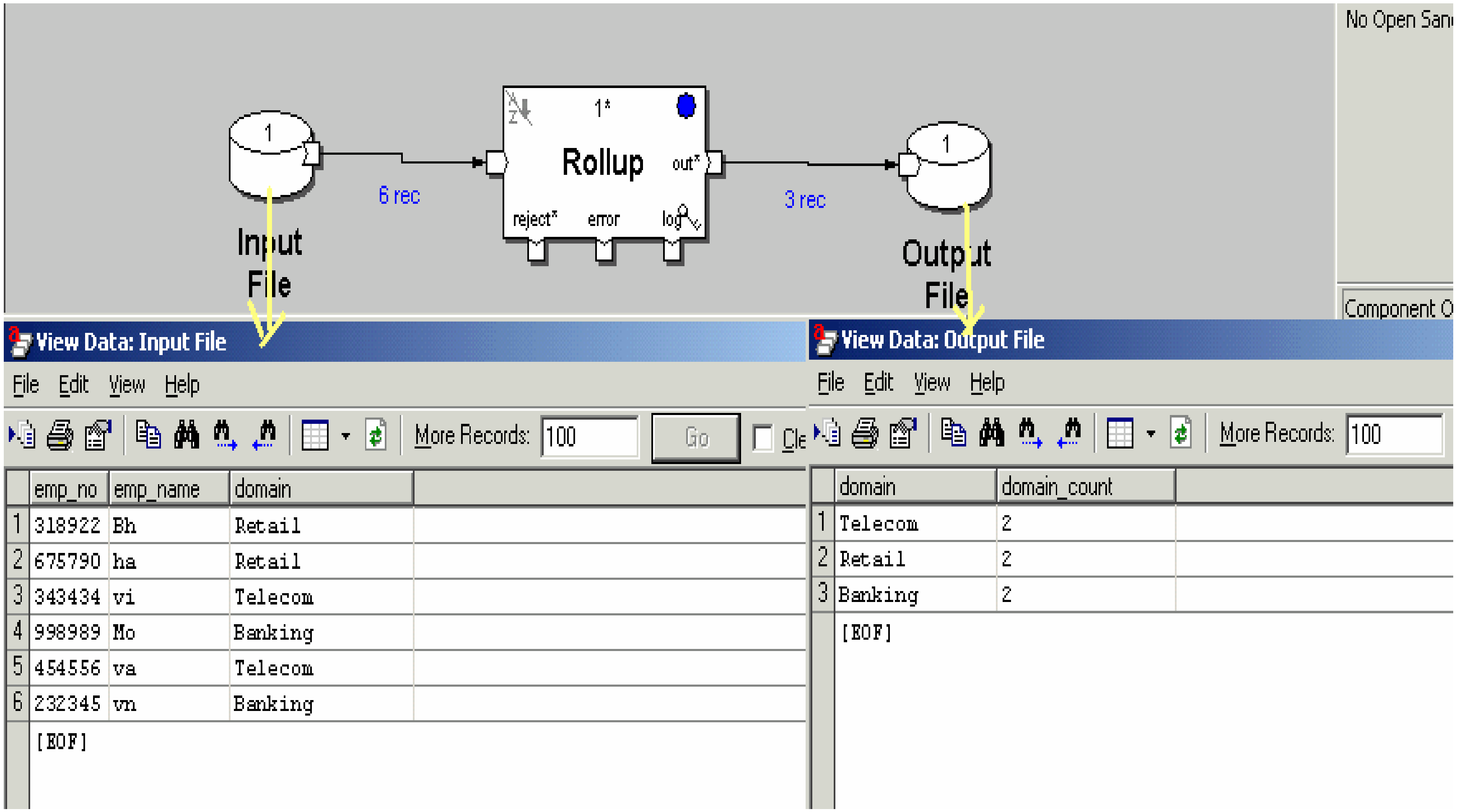
Type and description:

Embedded Transform : Filename containing rollup functions

Value:

```
begin
  out.domain :: in.domain;
  out.domain_count :: count(in.domain);
end;
```

Rollup - Output



	emp_no	emp_name	domain
1	318922	Bh	Retail
2	675790	ha	Retail
3	343434	vi	Telecom
4	998989	Mo	Banking
5	454556	va	Telecom
6	232345	vn	Banking

[EOF]

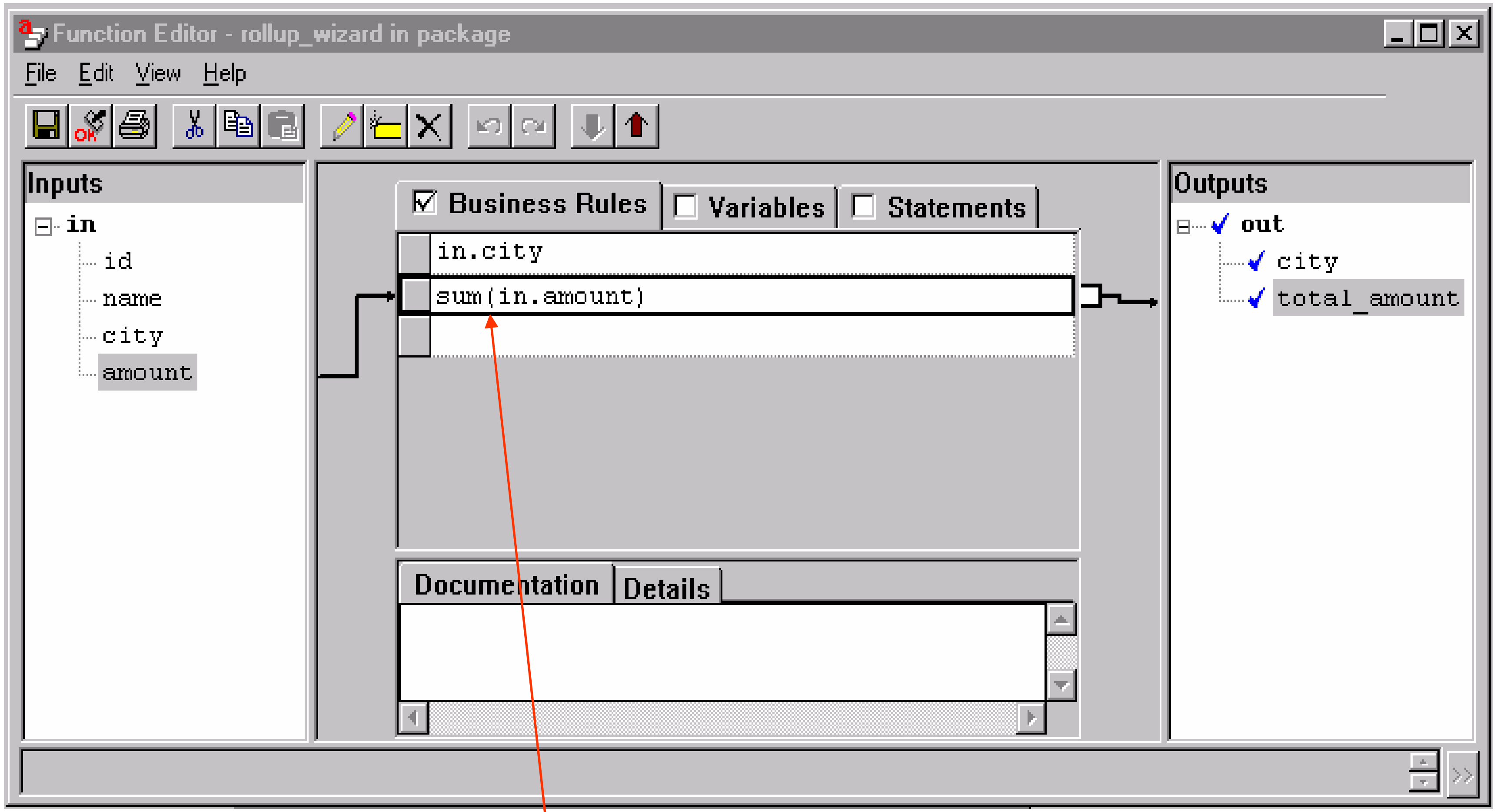
	domain	domain_count
1	Telecom	2
2	Retail	2
3	Banking	2

[EOF]

Built-in Functions for Rollup

- The following aggregation functions are predefined and are only available in the rollup component:
  - avg
  - max
  - min
  - count
  - first
  - Product
  - last
  - Sum
  - Multi-stage Transform – initialize,iterate,finalize,use of variables

Rollup Wizard



Inputs:
- in
  - id
  - name
  - city
  - amount

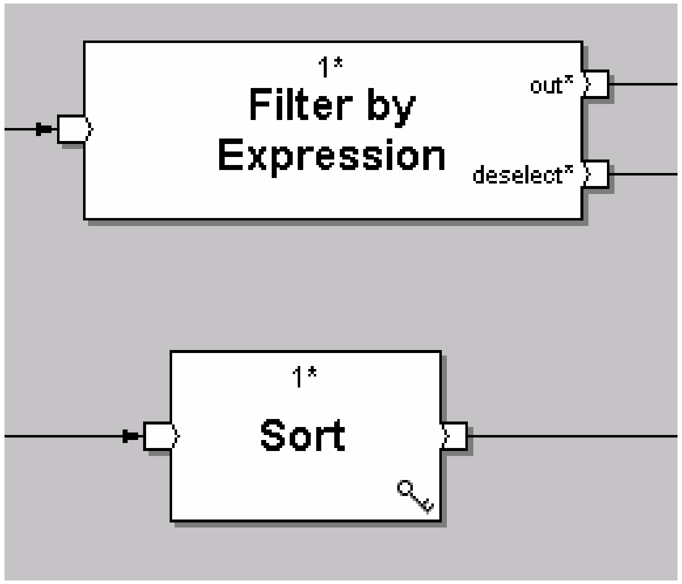
Business Rules:

```
in.city
sum(in.amount)
```

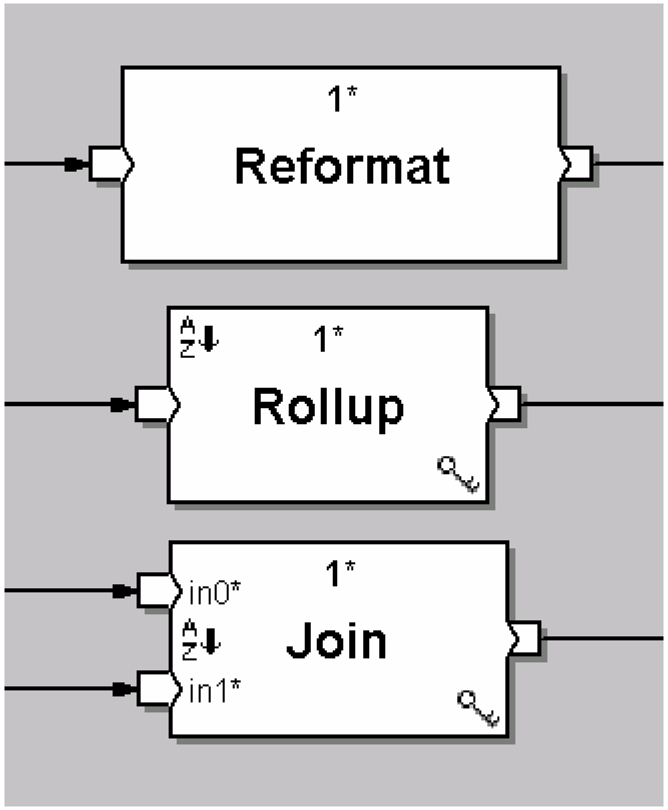
Outputs:
- out
  - city
  - total_amount

Note the use of an aggregation function in the expression

Simple and Complex Components



In these components the record format metadata does not change from input to output



In these components the record format metadata typically changes (goes through a transformation) from input to output

Priority Assignment

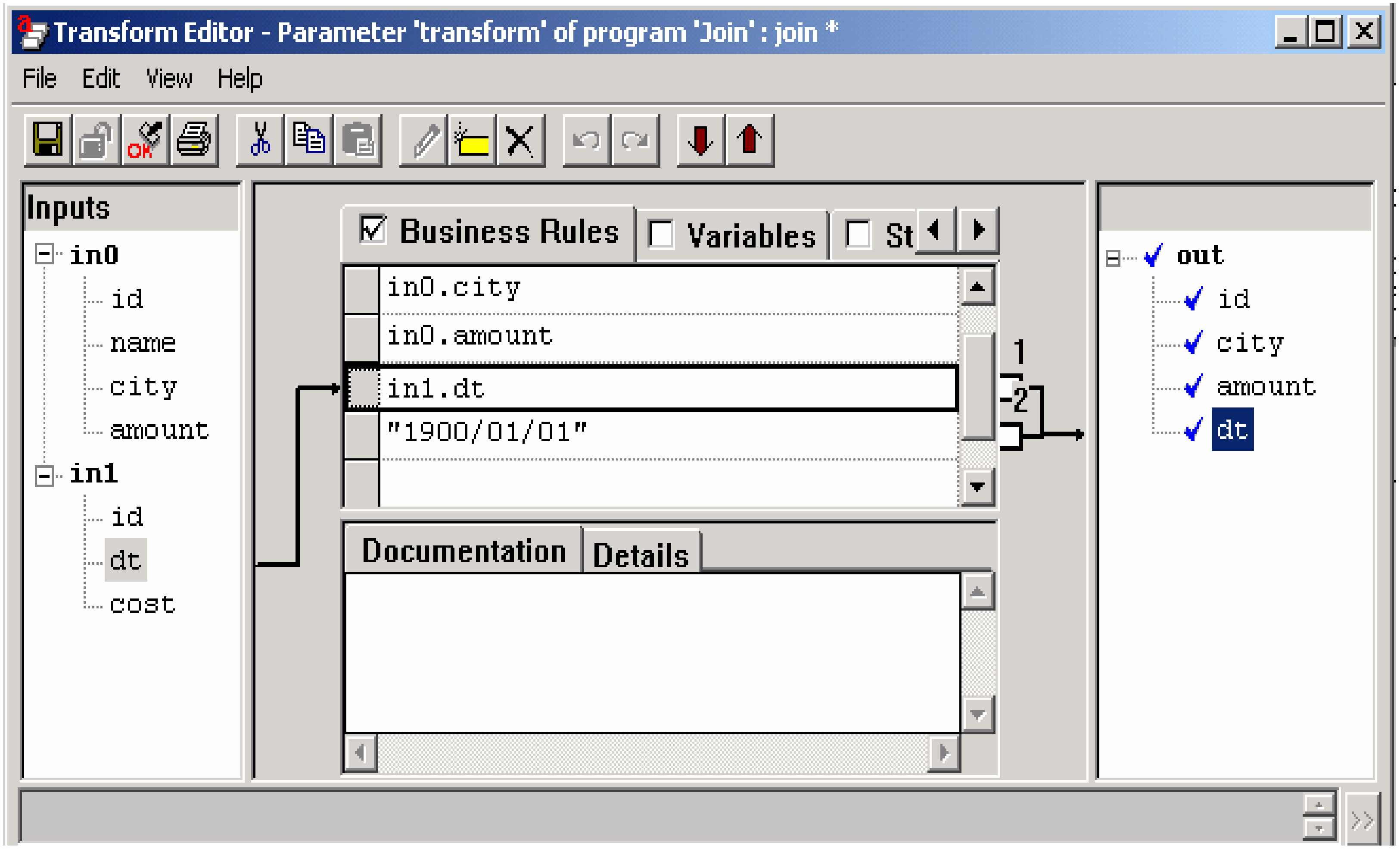
The *Priority* is the order of evaluation of rules in a transform function.

An example

A join component may have a transform function with prioritized rules as

```
out.ssn  :1:  in1.ssn;

out.ssn  :2:  in2.ssn;

out.ssn  :3:  "999999999";
```

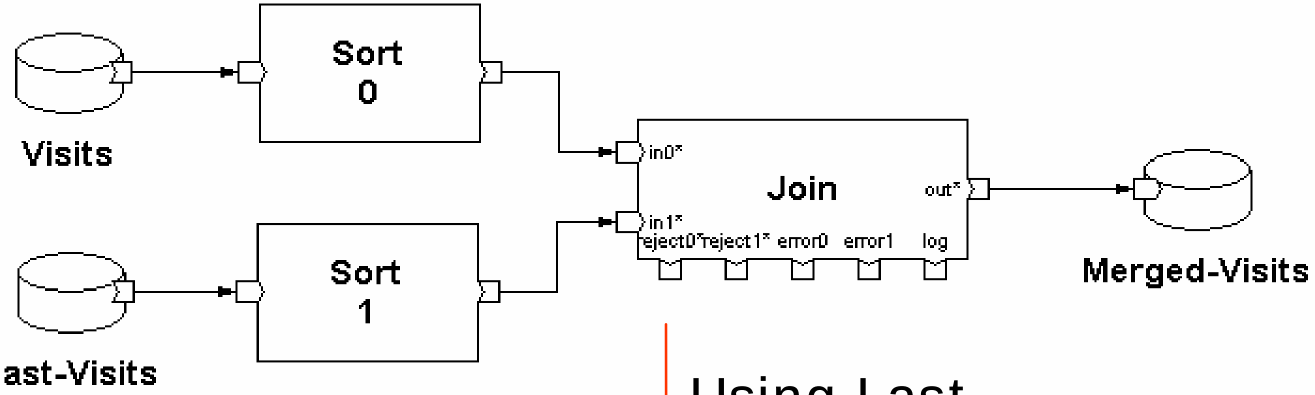
Priority Assignment contd…



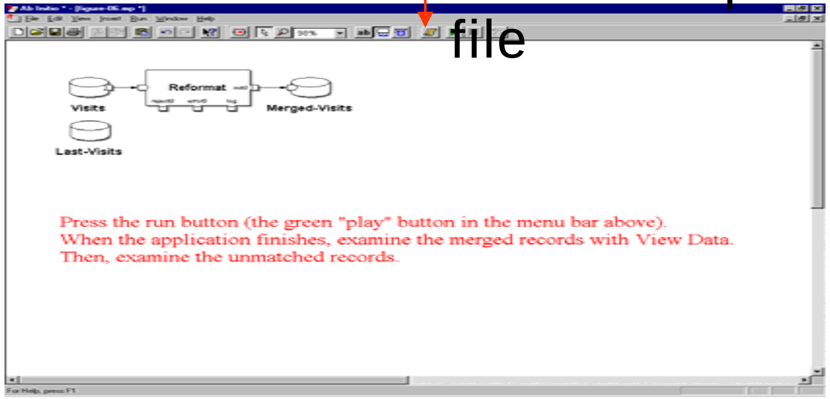
Transform Editor - Parameter 'transform' of program 'Join' : join *

```
in0.city
in0.amount
in1.dt
"1900/01/01"
```

Using lookup instead of Join



Using Last-Visits as a lookup file



Press the run button (the green "play" button in the menu bar above).
When the application finishes, examine the merged records with View Data.
Then, examine the unmatched records.

Using a lookup file in a Transform Function

Input 0 record format:

```
record
  decimal(4) id;
  string(6) name;
  string(8) city;
  decimal(3) amount;
end
```

Output record format:

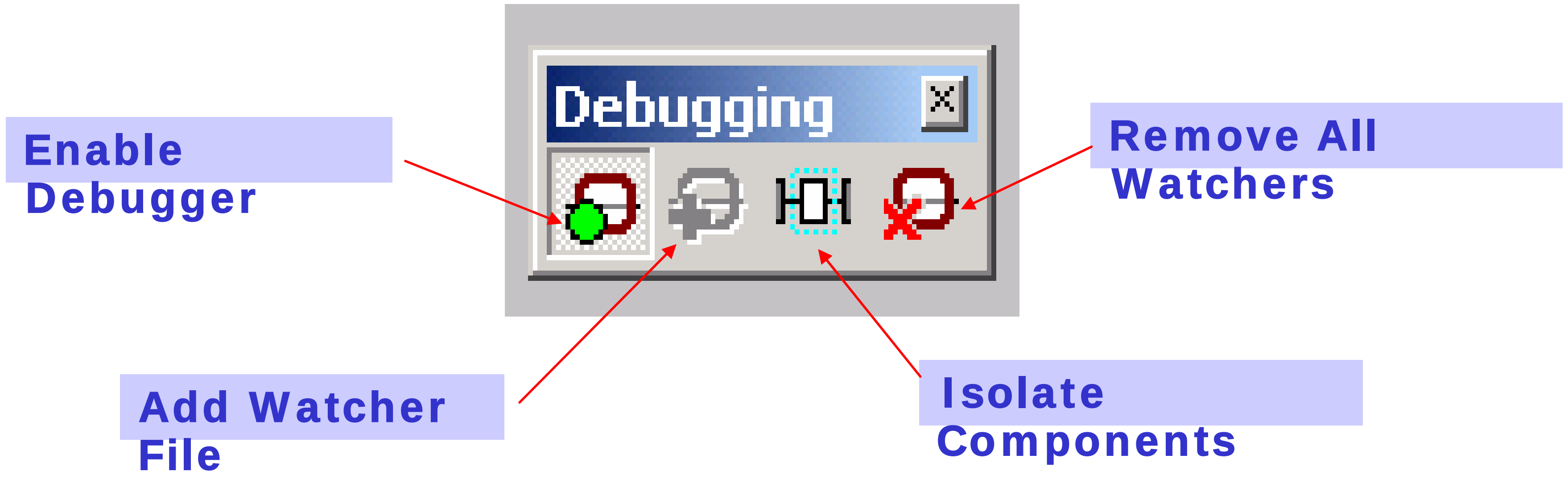
```
record
  decimal(4) id;
  string(8) city;
  decimal(3) amount;
  date(”YYYY/MM/DD”) dt;
end
```

Transform function:

```
out :: lookup_info(in) =
begin
  out.id     : : in.id;
  out.city   : : in.city;
  out.amount : : in.amount;
  out.dt     :1 : lookup(”Last-Visits”, in.id).dt;
  out.dt     :2 : ”1900/01/01”;
end;
```

The GDE Debugger

- The GDE has a built in debugger capability
- To enable the Debugger, Debugger:Enable Debugger
- The Debugger Toolbar



Multistage Transform

- Data transformation in multiple stages following several sets of rules
- Each set of rule form one transform function
- Information is passed across stages by temporary variables
- Stages include initialization, iteration, finalization and more
- Few multistage components are aggregate, rollup, scan

Aggregate/Rollup/Scan

- Generates summary records for group of input records

Database Components

* **Input Table**
  - unloads data records from a database into an Ab Initio graph
  - Source : DB table or SQL statement to SELECT from table
* **Output Table**
  - loads data records into a database
  - Destination : DB table or SQL statement to INSERT into table
* **Update Table**
  - executes UPDATE or INSERT statements in embedded SQL format to modify a DB table

*** Join with DB**

*** Truncate Table**

- **Deletes all the rows in a specified DB table**

*** Run SQL**

- **Executes SQL statements in a DB**

Built-In Functions

Ab Initio built-in functions are DML expressions that

- can manipulate strings, dates, and numbers
- access system properties

Function categories

- Date functions : now(), today(), date_to_int(), ..
- Inquiry and error functions: is_defined(), is_valid(), force_error(), ..
- Lookup functions: lookup(), lookup_local(), ..
- Math functions: ceiling(), floor(), ..
- Miscellaneous functions:decimal_round(), hash_value(), ..
- String functions: string_substring(), is_blank(), ..

Components contd..

Name	Description
Normalize	✸Generates multiple data records from each input data record ✸Separate a data record with a vector field into several individual records, each containing one element of the vector.
Denormalize Sorted	✸Consolidates groups of related data records into a single output record with a vector field for each group ✸Requires Grouped Input
Validate Records	Separates valid data records from invalid data records
Check Order	Tests whether data records are sorted according to a key-specifier.
Compare Records	Compares data records from two flows one by one
Generate Records	Generates a specified number of data records with fields of specified lengths and types.
Gather Logs	Collects the output from the log ports of components for analysis of a graph after execution
Sample	Selects a specified number of data records at random from one or multiple input flows

Parallelism in Ab Initio

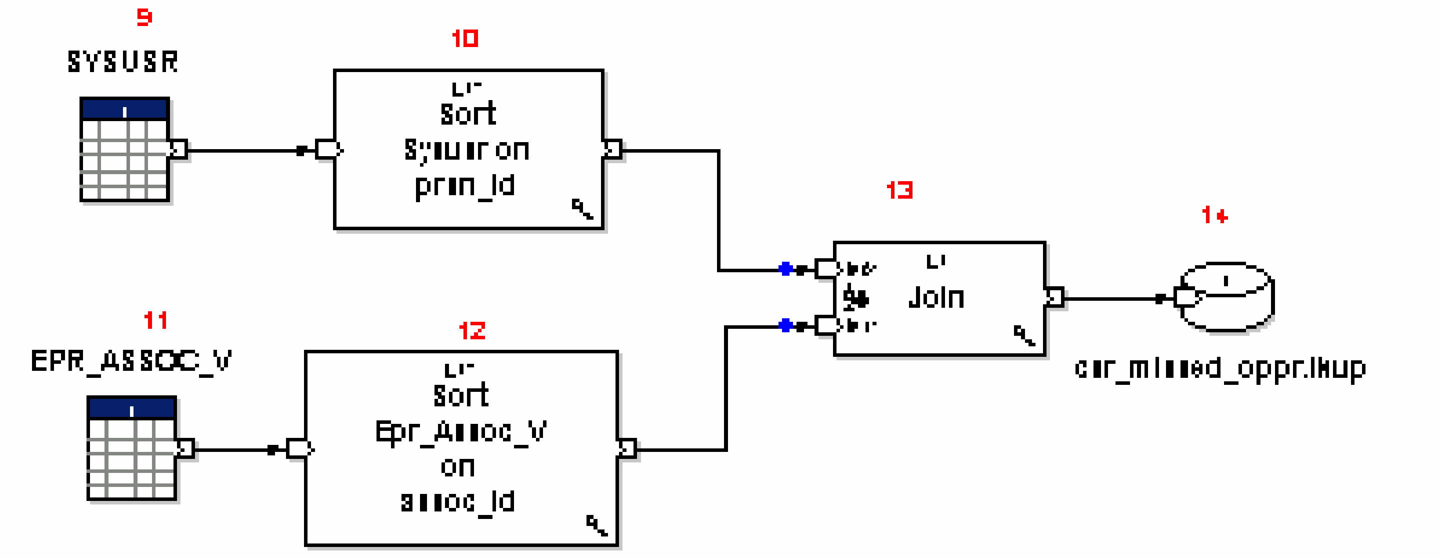
- ❑ Mechanism by which some or all constituents of an application – datasets and processing modules are replicated into a number of partitions, each spawning a process.
- ❑ This makes the Ab initio to process considerable huge volume (in millions) of records with an optimum usage of hardware available.

The power of Ab Initio lies in the fact that it can process data in parallel runtime environment

- ❑ Types of Parallelism
- ● Component Parallelism
- ● Pipeline Parallelism
- ● Data Parallelism

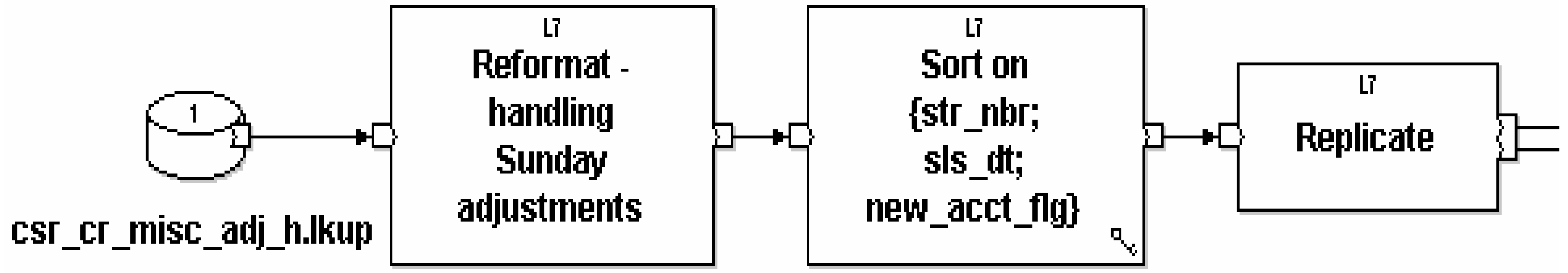
Component Parallelism

Component Parallelism is achieved when different instances of same component run on separate data sets**.** Component parallelism scales to the number of branches of a graph — the more branches a graph has, the greater the component parallelism. If a graph has only one branch, component parallelism cannot occur.



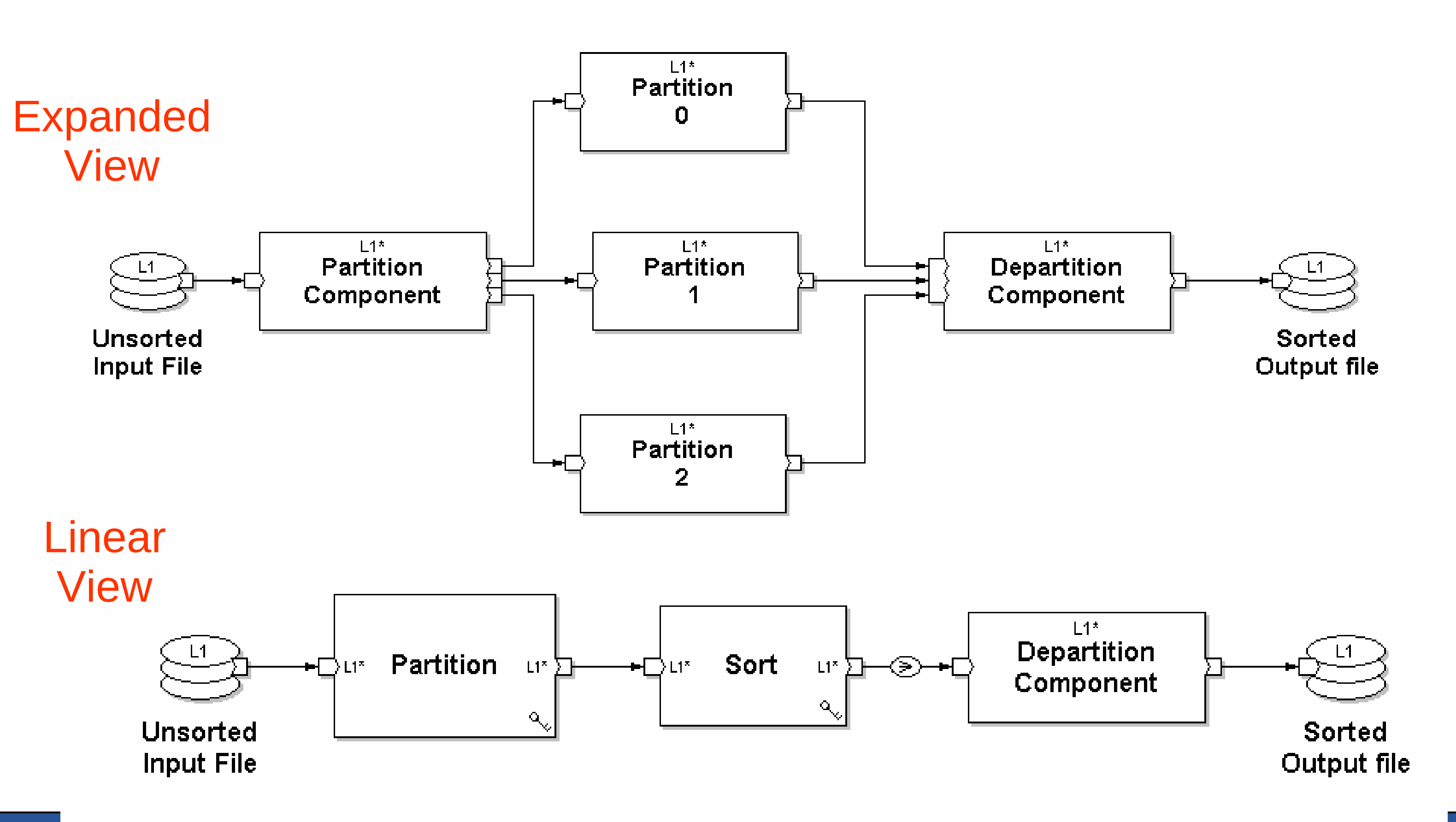
Pipeline Parallelism

Pipeline parallelism occurs when several connected program components on the same branch of a graph execute simultaneously. In this kind the two processing stages of the graph run concurrently.


1
csr_cr_misc_adj_h.lkup
L7
Reformat - handling Sunday adjustments
L7
Sort on {str_nbr; sls_dt; new_acct_flg}
L7
Replicate

Data Parallelism

When data is divided into segments or *partitions* and multiple instances of program components run simultaneously on each *partition*



Multifiles

✸A global view of a set of ordinary files called *partitions* usually located on different disks or systems

✸Ab Initio provides shell level utilities called *“m_ commands”* for handling multifiles (copy, delete,move etc.)

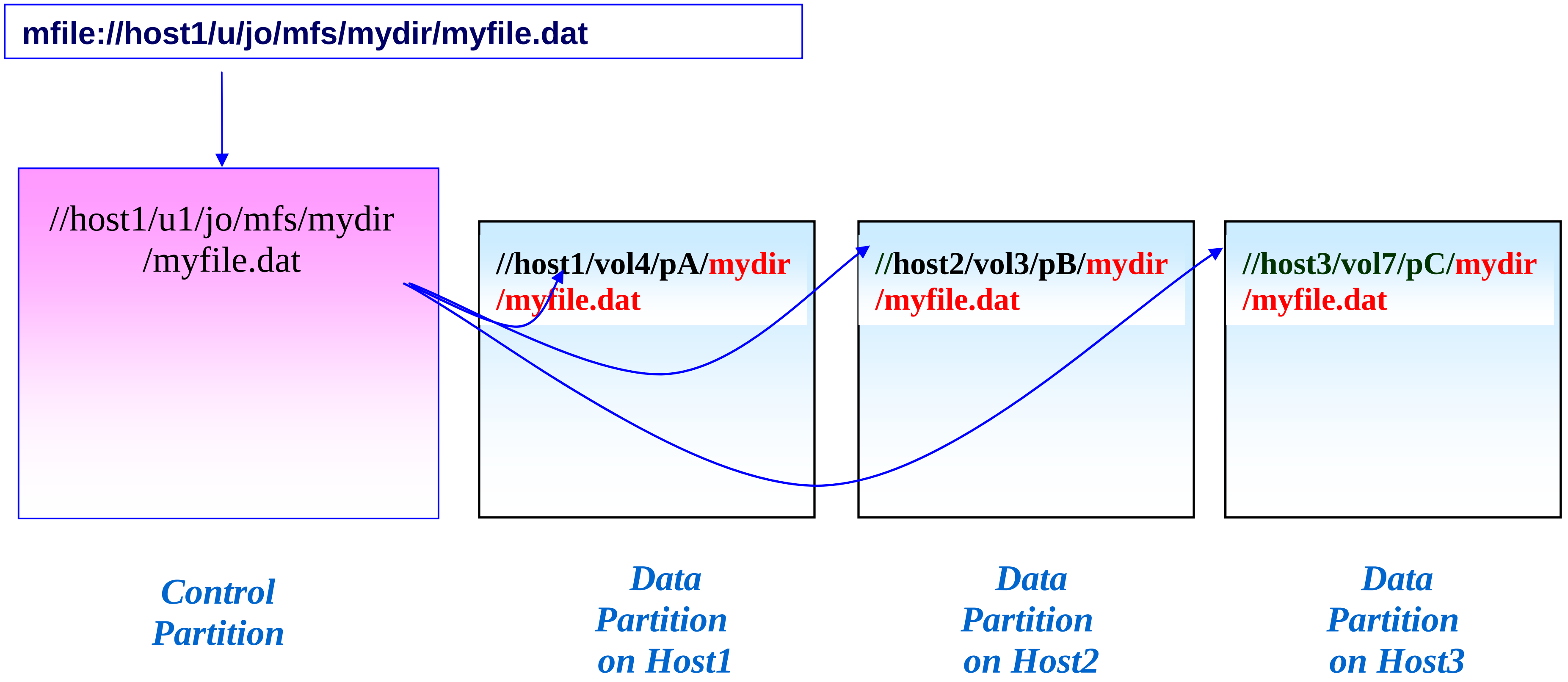
✸Multifiles reside on Multidirectories

✸Each is represented using URL notation with *“mfile”* as the protocol part:

⇨ mfile://pluto.us.com/usr/ed/mfs1/new.dat

A Multifile

A file spanning across partitions on same/different hosts



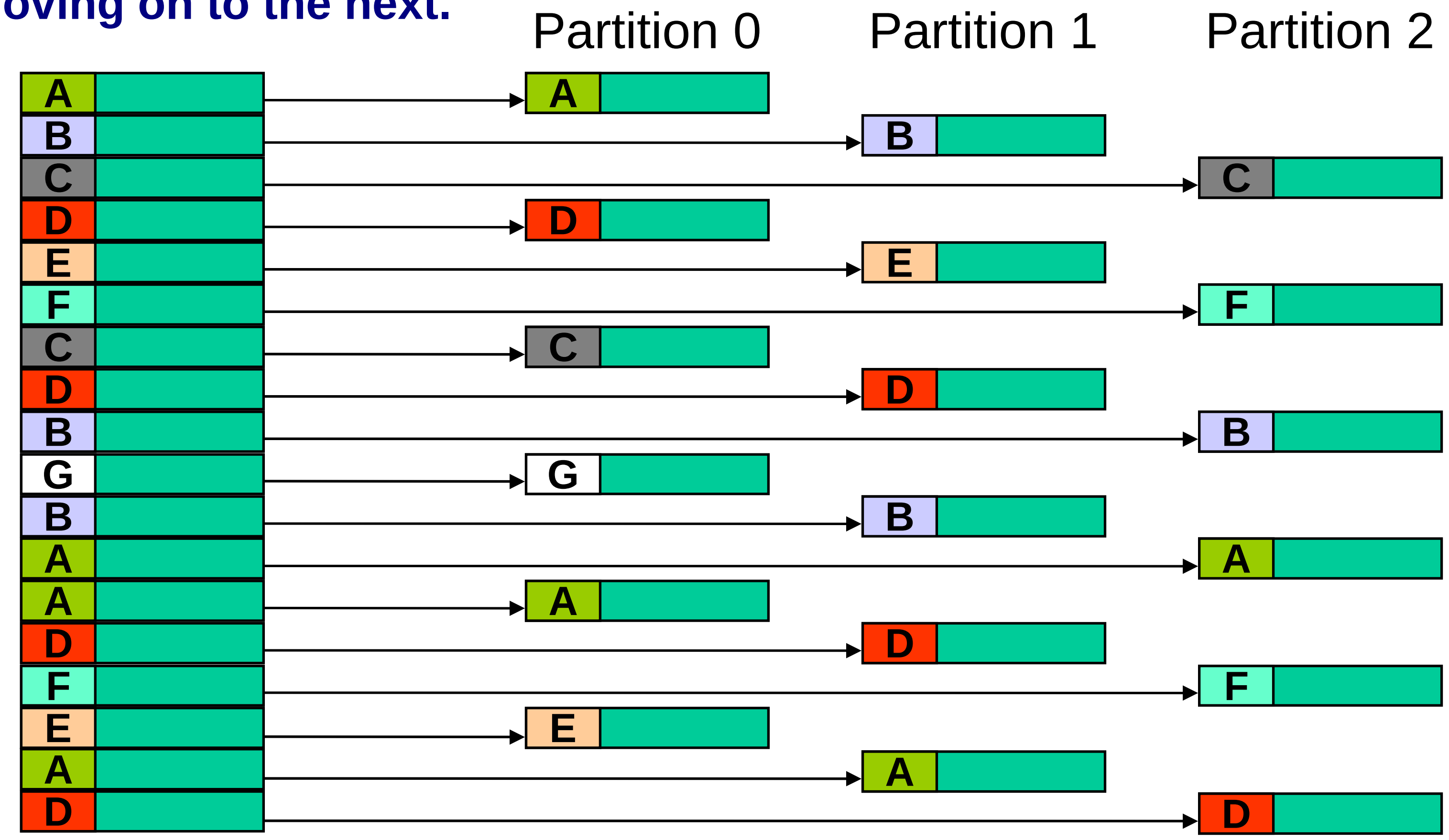
Data Partitioning Components

Data can be partitioned using

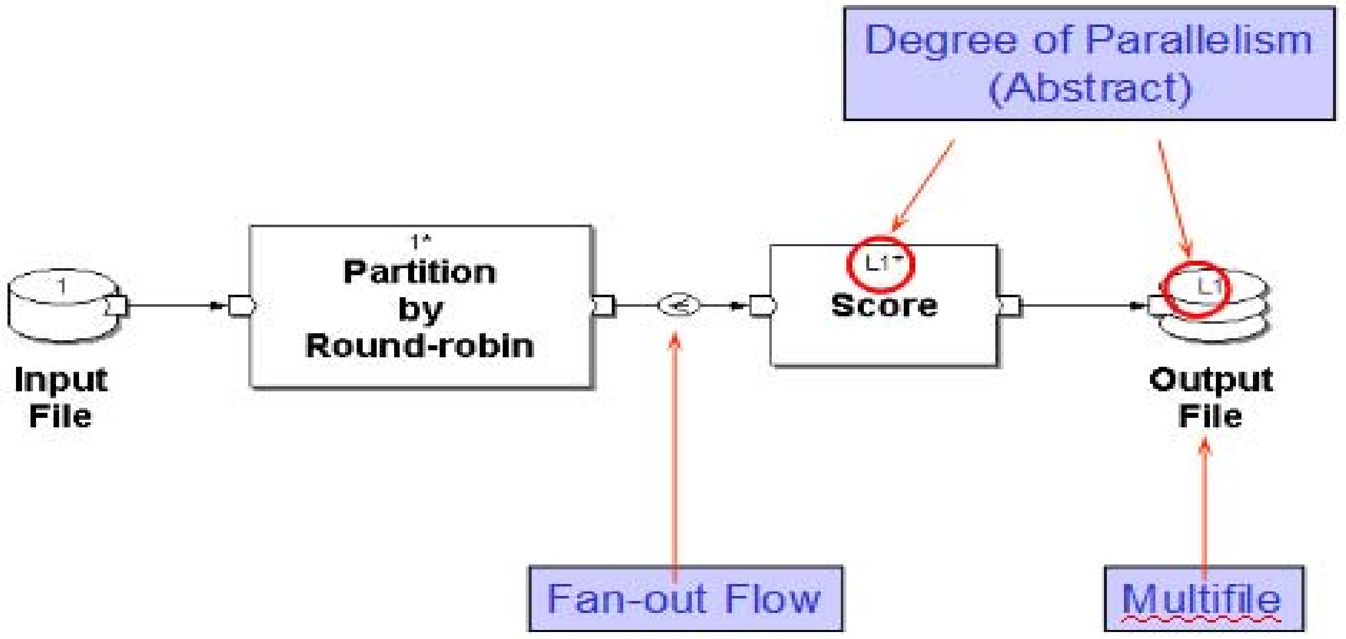
- **Partition by Round-robin**
- **Partition by Key**
- **Broadcast**
- **Partition by Expression**
- **Partition by Range**
- **Partition by Percentage**
- **Partition by Load Balance**

Round-robin Partition

Writes records to each partition evenly
Block-size records go into one partition before moving on to the next.

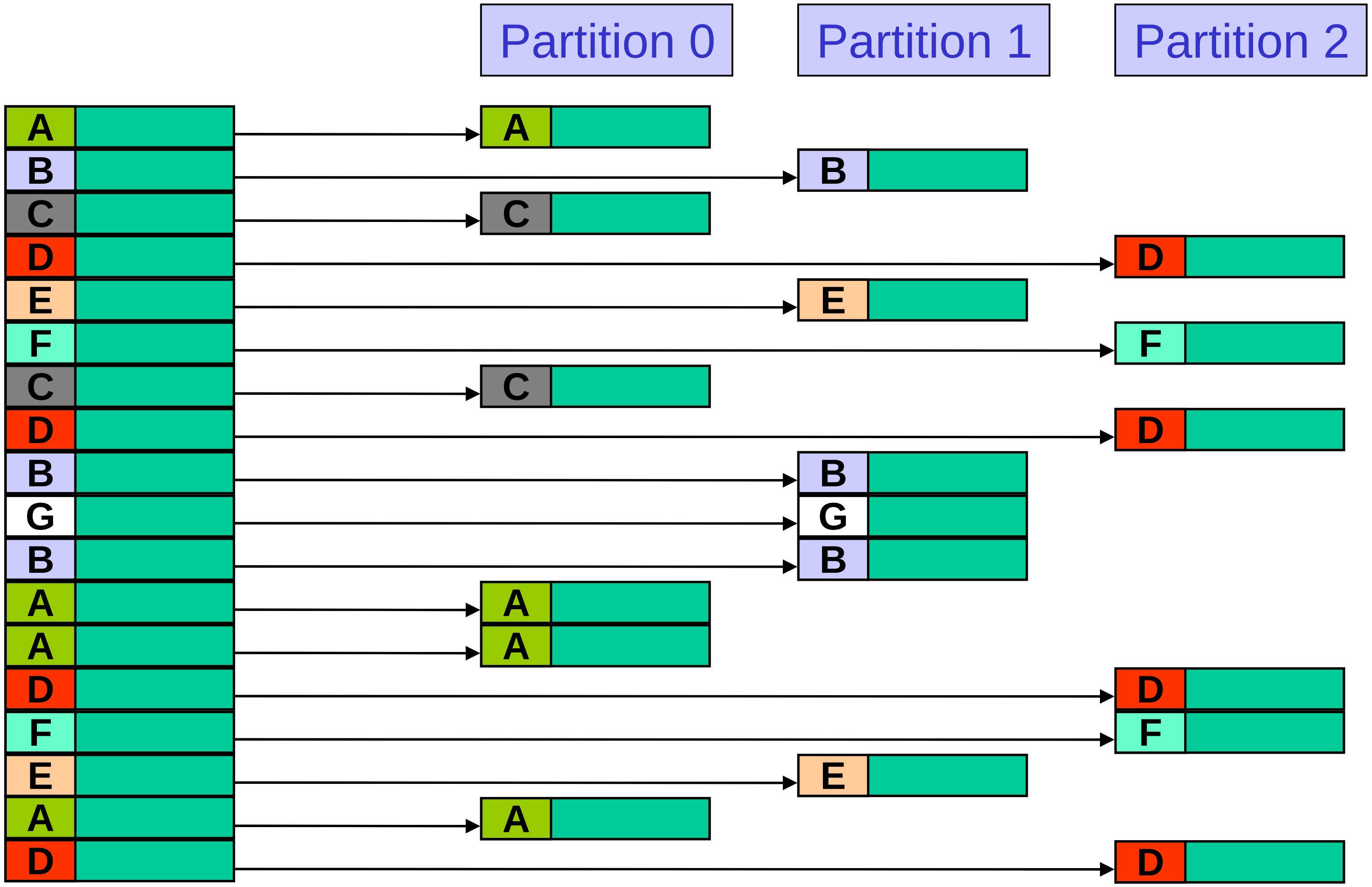


A Data Parallel Application: The Global View



Partitioning by Key

A hash code computed using the key determines which partition a record will be written on, meaning that records with the same key value will go to the same partition



Departitioning Components

- **Gather**
  - **Reads data records from the flows connected to the input port**
  - **Combines the records arbitrarily and writes to the output**
- **Concatenate**
  - **Concatenate appends multiple flow partitions of data records one after another**
- **Merge**
  - **Combines data records from multiple flow partitions that have been sorted on a key**
  - **Maintains the sort order**

Factors: Phases & Checkpoints

Phasing:

•**Breaking an application into phases limits the contention for**
Main memory.
Processor(s).

•**Breaking an application into phases cost**
Disk space.

Checkpoint - Purpose:

- ✧ **Provide same functionality as phase**
- ✧ **Additional: Provide restart capability**

How does it work ?

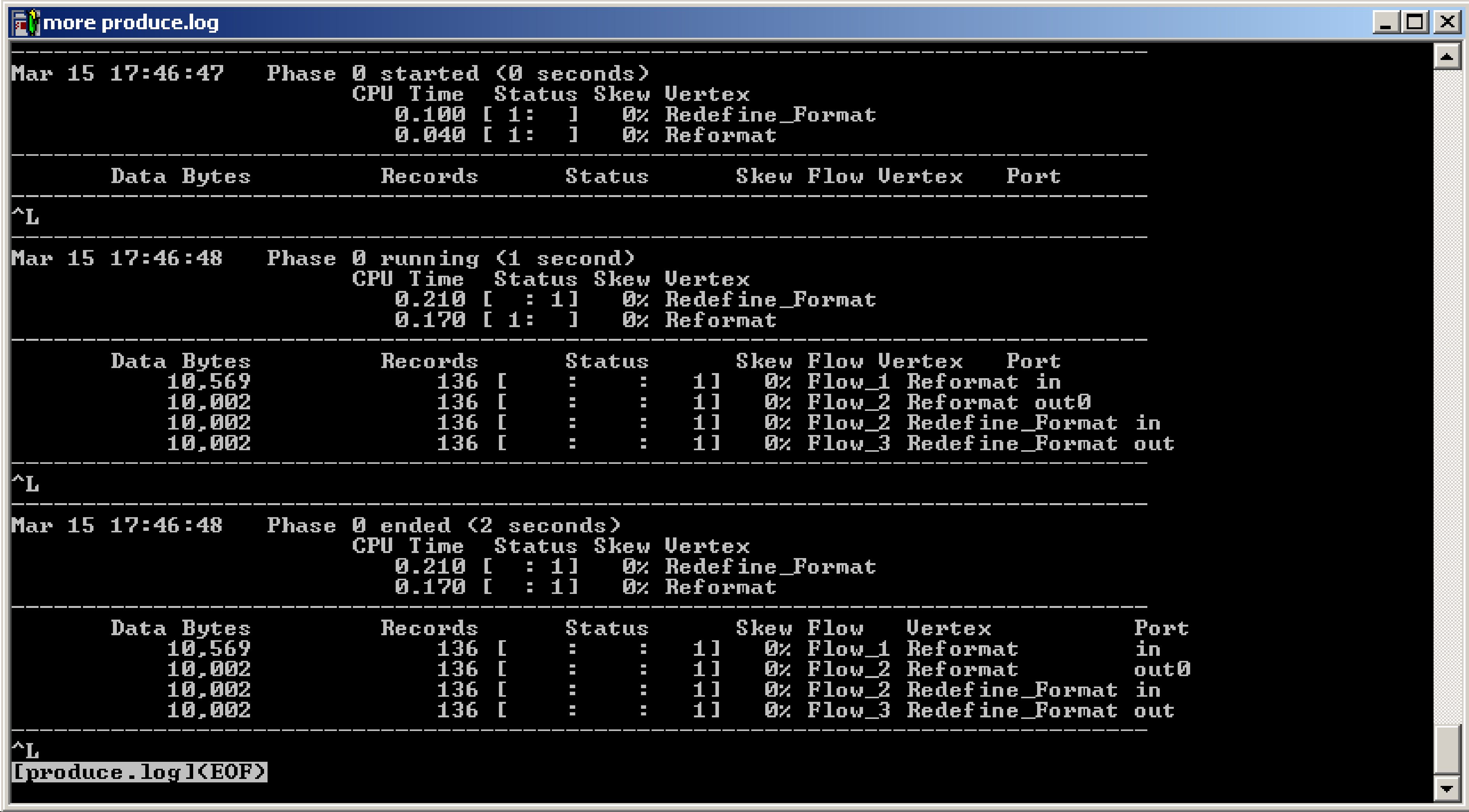
- ✧ **At job start, output datasets are copied to temporary files (in .WORK-serial or .WORK-parallel directories)**
- ✧ **At checkpoint completion, intermediate datasets and job state are stored in temporary files**
- ✧ **Recovery information is stored in host and vnode directories represented by AB_WORK_DIR defined in the Ab Initio environment**

AB_WORK_DIR

- Directory dedicated to Co>Ops
- Should have enough free space; Cannot be NFS or NAS mounted
- Holds Storage of Internal Log Files (used in recovery of Ab Initio Graph)
- Used when components are connected via name pipes
- Sub-directories of AB_WORK_DIR
    - host – Holds Control Node Recovery Files
    - vnode – Holds Processing Node Recovery Files
    - data – Holds files for Layouts
    - cache – Holds Cache Files needed by remote components
- Important logging information in “host” and “vnode” directories
- Usually does not have data files.
- Components with host layouts or database layouts, data written to “data” subdirectory
- AB_WORK_DIR fill up leads to non-recovery of Ab Initio Jobs.

Performance: Debugging Log file

A sample log file ..



```
--------------------------------------------------------------------------------
Mar 15 17:46:47   Phase 0 started (0 seconds)
                         CPU Time   Status Skew Vertex
                            0.100 [ 1:   ]   0% Redefine_Format
                            0.040 [ 1:   ]   0% Reformat
--------------------------------------------------------------------------------
       Data Bytes             Records       Status        Skew Flow Vertex    Port
--------------------------------------------------------------------------------
^L
--------------------------------------------------------------------------------
Mar 15 17:46:48   Phase 0 running (1 second)
                         CPU Time   Status Skew Vertex
                            0.210 [   : 1]   0% Redefine_Format
                            0.170 [ 1:   ]   0% Reformat
--------------------------------------------------------------------------------
       Data Bytes             Records       Status        Skew Flow Vertex    Port
           10,569                 136 [     :    :   1]    0% Flow_1 Reformat in
           10,002                 136 [     :    :   1]    0% Flow_2 Reformat out0
           10,002                 136 [     :    :   1]    0% Flow_2 Redefine_Format in
           10,002                 136 [     :    :   1]    0% Flow_3 Redefine_Format out
--------------------------------------------------------------------------------
^L
--------------------------------------------------------------------------------
Mar 15 17:46:48   Phase 0 ended (2 seconds)
                         CPU Time   Status Skew Vertex
                            0.210 [   : 1]   0% Redefine_Format
                            0.170 [   : 1]   0% Reformat
--------------------------------------------------------------------------------
       Data Bytes             Records       Status        Skew Flow   Vertex          Port
           10,569                 136 [     :    :   1]    0% Flow_1 Reformat        in
           10,002                 136 [     :    :   1]    0% Flow_2 Reformat        out0
           10,002                 136 [     :    :   1]    0% Flow_2 Redefine_Format in
           10,002                 136 [     :    :   1]    0% Flow_3 Redefine_Format out
--------------------------------------------------------------------------------
^L
[produce.log](EOF)
```

Performance: Debugging Log file

- **Reading the Log : CPU**
  - CPU time: total processing for component
  - Status: [Running : Finished]
  - Skew: among CPU times of each partition
  - Vertex: component

- **Reading the Log : DATA**
  - Data bytes: # processed
  - Records: # processed
  - Status: [unopened : opened : closed]
  - Skew: among data bytes in partitions
  - Flow: link between components
    - ⇨ data tracking info is displayed on flows in GDE
  - Vertex: component
  - Port: of component

- **Interpreting the log**
  - Compute data bytes/sec through component, in each partition
  - Look for serialization: effective CPU = (cpu time)/(elapsed time)
  - compare open vs. closed partitions:serialized when some partitions remain open long after others have closed data skew
  - Deadlock:no change in record counts over couple of intervals

Performance: In a nutshell ..

- Avoid Sorts as it is consuming more memory.
- Avoid components like Join with DB(hitting db for each and every record) .
- Use Lookups.
- Use In-memory Join/Rollup.
- Assign Driving Port of Join correctly.
- Filtering un-required data before processing.
- Phasing.

THANK YOU