Retek® Extract Transform and Load 10.3

Programmer's Guide

Retek

The software described in this documentation is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Information in this documentation is subject to change without notice.

Retek provides product documentation in a read-only format to ensure content integrity. Retek Customer Support cannot support documentation that has been changed without Retek authorization.

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403

888.61.RETEK (toll free US)
+1 612 587 5000

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46
Fax: +44 (0)20 7563 46 10

Printed in the United States of America.

The Apache Software License, Version 1.1

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

"This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED “AS IS” AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>. Portions of this software are based upon public domain software originally written at the National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign.

Customer Support

Customer Support hours:

Customer Support is available 7x24x365 via e-mail, phone and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance.

Contact Method	Contact Information
Internet (ROCS)	www.retek.com/support Retek's secure client Web site to update and view issues
E-mail	support@retek.com
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: +1 612-587-5800 EMEA: 011 44 1223 703 444 Asia Pacific: 61 425 792 927
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step by step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

Contents

Chapter 1 – Introduction

RETL is a high performance, scalable, platform independent, and parallel processing data movement tool. RETL attempts to address several primary needs:

- Database independent applications.
- Platform independent applications.
- Developing such applications more quickly than possible with conventional coding methods (e.g. custom crafted C and/or C++ code).
- High performance data processing.

In order to provide for these needs, the RETL defines an interface (in XML) that applications can call to define ETL functions. This interface is in a well-defined XML form that allows access to any database that the RETL supports.

RETL is a cross-platform development tool available on many different platforms (e.g. AIX, SunOS, and HP-UX). The XML definitions do not change on a per platform basis so moving between hardware platforms is a snap if you are using RETL.

Development of the XML instructions for RETL is much simpler, faster and less error prone than writing C and/or C++ code. The result is that applications can be completed much faster than possible with previous methods.

This guide shows you, the application developer, how to rapidly deploy RETL in your application and use it to manage your parallel processing system. Specifically, the guide provides information to:

- Install and set up RETL
- Configure RETL
- Administer RETL
- Develop applications using RETL

RETL is built on a C++ framework. Select the RETL operators you need to perform your application's data processing tasks. Manage your system resources to scale your application by partitioning data across multiple processing nodes.

RETL resides as an executable on your application's platform, and on any other system that serves as a source or target of data.

Technical specifications

RETL runs on the platforms listed in this section. Upon installation, a RETL executable for your chosen platform is installed on the server. For historical reasons the executable for RETL is called "rfx" and when referencing the executable we will use that name.

The release_notes.txt that documents the current configurations is included with the RETL CD. If you have a configuration that is not included in the release_notes.txt, verify with Retek Customer Support to see if your configuration is now available.

Operating systems

Sun Solaris (version 8)

IBM AIX (versions 4.3.3 and 5.1)

HP-UX (versions 11 and 11i)

Database management systems

Oracle (8i and 9i)

IBM DB2 (version 7.2.3)

NCR TeraData (release 2 version 4.3)

Note: as of version 10.2, RETL can be installed and configured to run completely independently of all databases and therefore does not require the configuration or installation of any database or database libraries. Of course, this version works on flat files only and has no database operators.

Data integration

Integrate external data of these forms:

- Unix flat files
- RDBMS tables including Oracle, IBM DB2, and NCR TeraData

Chapter 2 – Installation and system configuration

Installation

Install RETL on each server system that will be involved in outputting or processing data. For example, if one system outputs data files and another system inputs that data and processes it, install RETL on both systems.

1 Log in as the `root` user on the appropriate host.

2 Create Unix groups for the following environments:

- rfx – group that owns the RETL software

3 Create a UNIX operating system account on the appropriate host, using ksh as the default shell. `rfx - rfx group.`

4 Create a directory where you will install this software.

5 Log in to the Unix server as `rfx`.

6 Mount the RETL CD on your UNIX server.

7 Change directories to the CD mount point `<cdrom_path>/<rfx_dir>.`

8 At a Unix prompt, enter: `>./install.sh`

Note: You must be in the `< cdrom_path>/<rfx_dir >` for the installation to successfully complete.

9 Follow the prompts to install RETL for your configuration.

```
> install.sh
Enter directory for RETL software:
---> /files0/release/sunora328
Is this the correct directory for the install? y or n
RFX_HOME: /files0/release/sunora328
---> y
Creating RFX_HOME directory /files0/release/sunora328 ...
Creating install directory in /files0/release/sunora328 ...
Enter Unix Operating System:
        1) IBM-AIX 4.3.3
        2) IBM-AIX 5
        3) SUN Solaris 8
        4) HP 11i
---> 3
Enter Database Platform:
        1) DB2 7.2 (32 bit)
        2) Oracle 8.1.7 (32 bit)
        3) Oracle 9.x (32 bit)
        4) Oracle 9.x (64 bit)
        5) Teradata (32 bit)
        6) Standalone (32 bit)
        7) Standalone (64 bit)
---> 2
Successful completion of RETL Install
```

```
Next be sure to setup your environment as per the "Setup" section
of the Programmer's Guide.

After setup - verify the installation by running the following
command:
$RFX_HOME/bin/verify_retl -doracle
```

10 Review the install.log file in the `<base directory>/install` to verify that RETL was installed successfully.

11 Set up your environment as per the Setup section below.

12 Verify the installation and setup by running the "verify_retl" script (see Testing the Setup (verify_retl)).

Upgrade from a previous version of RETL

Upgrading versions of RETL is a very simple thing to do. In short, these are the steps that you should follow:

1 Choose a new (and different) location to install the new version of RETL.

2 Install the new version of RETL following the installation instructions.

3 Change the environment that RETL runs within so that it refers to the new RFX_HOME (e.g. change within .kshrc or .cshrc).

4 Double check the environment variables to make sure that you weren't explicitly referring to old RFX directories explicitly.

Separate the RETL specific changes that you make to your environment so that the environment variables can be easily removed, modified and/or replaced when necessary.

Version 10.3 of RETL includes better validation of properties and parameters than previous releases. In some cases WARNING messages appear where they did not in previous releases. This is expected. Your RETL installation is working fine but there are problems in the scripts. For more information see the troubleshooting section: "I've upgraded and now I get WARNINGS when I never did before! What's wrong!?"

Set up

1 After installation, set up your Unix environment for RETL. The following example shows the variables you need in your Unix profile to run RETL properly.

```
export RFX_HOME=<base directory>
export PATH=$RFX_HOME/lib:$RFX_HOME/bin:${PATH}
```

Note: The path to the database system libraries must be in the library path. The suggested library path environment variables per operating system are shown below. For example, if you are running the 32 bit Oracle 9i version of rfx you should include $ORACLE_HOME/lib32 in your library path. You should try connecting to the database before running rfx to ensure that your environment is setup properly.

Additional environment variables apply to different operating systems:

SUN:
```
export LD_LIBRARY_PATH=$RFX_HOME/lib:$LD_LIBRARY_PATH
```

HPUX:
```
export SHLIB_PATH=$RFX_HOME/lib:$SHLIB_PATH
```

AIX:
```
export LIBPATH=$RFX_HOME/lib:$LIBPATH
```

2 If you've selected an installation with database support, set up database and environment variables required for basic database setup. Please refer to the Appendix C – Database Configuration and Troubleshooting Guide for more information on setting up your database.

For Oracle:

```
export ORACLE_HOME=/your/Oracle_home/directory
export PATH=$ORACLE_HOME/bin;$PATH
```

For DB2:

```
. /your/DB2PATH/sqllib/db2profile
export PATH=/your/DB2PATH/bin;$PATH
```

For TeraData:

Current versions of RETL require that the ODBC driver be located at /usr/odbc/drivers/tdata.so. Additionally /usr/odbc/lib is included in your library path (use the appropriate library path variable from step 2 above).

```
export PATH=/usr/odbc/lib;$PATH
```

3 Log in to the Unix server as `rfx`. At the Unix prompt, enter:

```
>rfx
```

4 If RFX is installed correctly and the .profile is correct, the following results:

```
rfx: `--flow-file' (`-f') option required!
```

Note: see the Troubleshooting section at the end of this document if you have problems or issues.

Test the setup (verify_retl)

When setting up RETL in a new environment (or if you'd just like to verify that the RETL environment is set up properly) run the "verify_retl" script located in the `/bin` directory of the RETL installation (note verify_retl is available as of release 10.2).

The usage for verify_retl is as follows:

```
verify_retl [-doracle] [-ddb2] [-dtera] [-nodb] [-h]
```

Option	Description
-doracle	Checks environment variables etc, for the Oracle installation of RETL.
-ddb2	Checks environment variables etc, for the DB2 installation of RETL.
-dtera	Checks environment variables etc, for the TeraData installation of RETL.
-nodb	Checks environment variables for the standalone version of RETL.
-h	Displays the help message.

This generates the following output if successful:

```
Checking RETL Environment...found ORACLE environment...passed!
Checking RETL binary...passed!
Running samples...passed!
========================================================================
Congratulations! Your RETL environment and installation passed all tests. See
the programmer's guide for more information about how to further test your
database installation (if applicable)
========================================================================

Exiting...saving output in /files0/retl/tmp/verifyretl-20384.log
```

RFX Command line options

You can get help on rfx options on the command line by typing "rfx –h" on the command line. You should see something like the following:

```
>rfx -h
rfx 10.3 build 183

Usage: rfx [OPTIONS]...
   -h          --help                  Print help and exit
   -V          --version               Print version and exit
   -cSTRING    --config-file=STRING    Configuration File
   -nINT       --num-partitions=INT    Number of Partitions
   -x          --no-partitioning       Disable partitioning (debug)
   -sTYPE      --schema-display=TYPE   TYPE to display schema as.
                                       Valid values:
                                       NONE, META, SCHEMAFILE
   -lLOGFILE   --log=FILE              Log statistics/times to FILE
   -fSTRING    --flow-file=STRING      XML file containing flow
               --davinci-files         Produce daVinci files
               --graphviz-files        Produce GraphViz files
```

These options are discussed in more detail in the table below:

Option	Default Value	Description
-h --help	n/a	Shows the help message shown above.
-V --version	n/a	Displays the version and build number.
-cSTRING --config-file=STRING	$RFX_HOME/etc/rfx.conf	Overrides the default configuration file.
-nINT --num-partitions=INT	As specified in the rfx.conf – or 1 if no rfx.conf is found.	The number of partitions to use. This feature is intended for RETL experts only.
-x --no-partitioning	Partitioning as defined in the rfx.conf.	Disables partitioning.

Option	Default Value	Description
-sTYPE --schema-display=TYPE	META	Prints the input and output schema for each operator. Valid values and descriptions: NONE – rfx will not print any schema information. META – default. rfx prints hierarchies of each operator's inputs and outputs as meta-data. SCHEMAFILE – if specified, This option prints the input and output for each operator in schema file format so that developers can quickly and easily cut and paste rfx output to a file and break up flows. Developers could then modify these files for the purposes of specifying IMPORT and EXPORT schema files. **Note:** rfx should be run with the –sNONE option in production systems where unnecessary output is not needed. The –sSCHEMAFILE option is often useful in development environments where it is desirable to debug rfx flows by breaking them up into smaller portions.
-lLOGFILE --log=FILE	n/a	Specifies the logfile in which to log rfx statistics/times. If the logfile path is relative, the log file will be placed in the directory as defined in the TEMPDIR element of the rfx configuration file (rfx.conf). This changes the default log file as specified in rfx.conf and will turn on logging only if the log level in rfx.conf is 'on'. For more information about the LOGGER feature, see the next section, entitled "Configuration"
-fSTRING --flow-file=STRING	Stdin	Specifies the file to use as input to rfx. This is where the XML flow is located.
--davinci-files	Off	This option is not currently supported.
--graphviz-files	Off	This generates two ".dot" files which contain the flow before and after partitioning. In order to view these you need to download a copy of "dotty" which is included with the graphviz package available from AT&T Bell Labs: (http://www.research.att.com/sw/tools/graphviz/download.html).

Configuration

A configuration file may be specified by a user to control how rfx uses system resources, where to store temporary files, and to set defaults values for operators.

Configuration field descriptions

These explanations should assist you in modifying your configuration file. The RETL configuration file, rfx.conf, is located in the `<base_directory>/etc` directory.

Element Type	Attribute Name	Attribute Value	Description
CONFIGURATION			The root element of the RFX Configuration file. This element can have either NODE or DEFAULTS elements.
NODE	hostname	Host name	The name of the UNIX server where RETL is installed.
	bufsize	8..n	The number of records allowed between operators at any given time. The default bufsize is 2048 records. The bufsize can have a significant impact on performance. Setting this value too low causes a significant amount of contention between processes – slowing down RETL. A value that is too high can also slow RETL down because it will consume more memory than it needs to. Finding the right value for your hardware configuration and flow is part of tuning with RETL.
	numpartitions	1..n	Optional value, defaults to 1.

Element Type	Attribute Name	Attribute Value	Description
TEMPDIR	path	Valid path	TEMPDIR is a child element of the NODE Element. It is the path to the directory where the RETL writes temporary files. We recommend that the number of temporary directories equals the number of partitions. Ideally each temp directory should be on a separate disk controller. **Note:** These directories must always be local to the host where rfx is running. Use of network drives can have a drastic impact on performance. **Note:** Care should be taken to protect the files within this directory since they can sometimes contain userid and password information. Please talk to your system administrator about setting the permissions so that only the appropriate personnel can access these files (e.g. by setting: umask 077). **Note:** Temporary files should be removed from temporary directories on a daily or weekly basis. This very important server maintenance task aids in RETL debugging, reviewing database-loading utility log files, and so on. If a RETL module fails, the user should not re-run the module before removing the temporary files that were generated by the failing module.
GLOBAL			This element specifies global options for global settings within RFX.
	bytes_per_character	1,2, or 3	This setting specifies how RFX treats character data. In short, it affects how many bytes are in a character and can be used to allow RFX to work seamlessly with UNICODE data.
LOGGER			The LOGGER element specifies a facility for rfx to log information to. This gives a dynamic view of RETL to allow developers to get some information about the data that flows through each operator. This also enables developers to determine if/when deadlock conditions occur, debug problems and tune performance. See the properties that follow on how to configure the LOGGER for rfx.
	type	file	The type of logging facility to use. Currently the only value rfx allows is to log to a 'file'.

Element Type	Attribute Name	Attribute Value	Description
	dest	<output filename>	An optional output destination to log to. Currently, this value must be an absolute or relative filename. If the filename is relative, the log file will be placed in the directory as defined in the TEMPDIR element of the rfx configuration file (rfx.conf). The log file can be overridden by specifying –lLOGFILE as a command-line parameter to rfx. The default value is "rfx.log".
	level	0,1,2	Specifies the level of verbosity recorded in the log file. Higher levels mean more information is logged. The following characteristics can be recorded: • rfx start and stop time along with command line parameters. • The flow name – if specified as part of the flow. • Statistics per operator – start time, stop time, record counts and records per second per operator. loglevel values have the following meaning: "0" = no logging. "1" = logging of a and b above. "2" = logging of a, b and c above. **Note:** leave logging turned off in production systems where performance is a critical factor. The logging feature is not designed to log errors and the like, but is intended to give rough measures of flow performance characteristics and aid in debugging flows. As a system administration side-note, periodically remove the log file to free unneeded disk space.
DEFAULTS			This element is a child of CONFIGURATION element. This section is used to define one or more default PROPERTY values for operators that are reused frequently. Care should be taken when using and changing these defaults since they can change the results of an rfx flow without changing individual flows.
	operator		Name of the operator to assign default values. Refer to the following chapters for the operators that are available.
PROPERTY			This element is a child of the DEFAULTS element.

Element Type	Attribute Name	Attribute Value	Description
	name		The name of the operator property to assign a default value. Refer to Chapter 5 for the property names for each operator.
	value		The value assigned as the default for the specified operator property. Refer to Chapter 5 for valid property values for each operator.

The following is a sample resource configuration for RETL:

```
<CONFIG>
   <NODE       hostname="localhost"
               numpartitions="1"
               bufsize="2048" >
      <TEMPDIR path="/u00/rfx/tmp"/>
      <TEMPDIR path="/u01/rfx/tmp"/>
   </NODE>
   <GLOBAL     bytes_per_character="1" >
      <LOGGER  type="file"
               dest="rfx.log"
               level="0" />
   </GLOBAL>
   <DEFAULTS   operator="oraread">
      <PROPERTY name="maxdescriptors" value="100"/>
   </DEFAULTS>
</CONFIG>
```

Multi-byte character support

The optional variable "bytes_per_character" allows RETL to provide multi-byte character support. The number of bytes read or written is dependent upon the size of a character. RETL determines the size of a string by multiplying the number of characters by bytes_per_character. If you are using databases, bytes_per_character is set up by consulting your database configuration.

For Imports and Exports, RETL uses the schema file to determine the number of bytes a particular string will have. For a fixed length field the number of bytes is simply <number of chars> * <bytes_per_character>. For delimited fields, the maximum number of bytes is <maxlength>*<bytes_per_character>.

Chapter 3 – RETL interface

RETL can process datasets either directly from a database or from file-based systems. Depending upon the data source, one of two operators can perform this function. For reading data directly from database tables, use the DBREAD operators—ORAREAD, DB2READ, or TERAREAD–for the database. See Chapter 5 - Database operators, for more information about how to use each DBREAD operator. For situations where you expect RETL to process data in a file format, use the IMPORT operator. This operator imports a disk file into RETL, translating the data file into a RETL dataset.

Note: We use DBREAD and DBWRITE throughout this guide as shorthand to indicate any one of the database read or write operators as detailed in Chapter 5 - Database operators.

Schema file requirements

RETL stores information about each dataset that describes the data structures (the metadata). The means by which you supply metadata is dependent on whether you interface data using the DBREAD or IMPORT operator. When datasets are created using DBREAD, the metadata is read along with the data directly from the database table. In this case, RETL requires no further metadata. On the other hand, if the IMPORT operator is used for dataset input, RETL requires the use of a schema.

The schema ensures that datasets are consistent from the source data to the target data. RETL reads in a schema file that is specific to the dataset that is being imported. RETL uses the schema file to validate the input dataset structure. For example, if the dataset ultimately populates a target table in a database, the schema file dictates the type and order of the data, as well as the data characteristics, such as the character type, length, and nullability.

Two types of schema files are defined for use with an incoming text data file:

- Delimited file type
- Fixed length file type

Schema file XML specification

Element Type	Attribute Name	Attribute Value	Description
RECORD			The root element.
	Type	"delimited" or "fixed"	Describes whether the schema is fixed records or delimited records.
	final_delimiter	character	Required field. End of record delimiter that is used within the input file to distinguish the end of a record. This is extremely important to help distinguish between records in the event that the input datafile is corrupt. Generally this is a newline character for readability.
	Len	1..n	Required only for fixed length fields. This is the total length of the fixed length record. Note that this must be equivalent to the sum of all aggregate field lengths.
FIELD			Child element of the RECORD.
	Name		The column name of the given field.
	delimiter		This optional attribute is for delimited fields only. It specifies the end of field delimiter, which can be any character, but the default value is the pipe character ('|').
	datatype	Any one of: "int8", "int16", "int32", "int64", "uint8", "uint16", "uint32", "uint64", "dfloat", "sfloat", "string", "date", "time", or "timestamp"	The datatype of the field. If the **date** data type is declared, the incoming data must be formatted as '**YYYYMMDD**'. If the **time** data type is declared, the incoming data must be formatted as '**HH24MISS'**. If the **timestamp** data type is declared, the incoming data must be formatted as '**YYYYMMDDHH24MISS'**
	nullable	"true" or "false"	Optional field describing whether null values are allowed in the field. The default value is false. If true then a nullvalue MUST be specified.

Element Type	Attribute Name	Attribute Value	Description
	nullvalue	Quoted string	Required if nullable=”true”. This is the text string that an IMPORT operator looks at to determine whether or not a field is null when reading flat files. This should be a unique value that will not otherwise be found within the dataset. For fixed length fields, this value must be equal in length to the field length so that the record length remains fixed. **Note**: the null string may specified (“”) for a fixed length field rather than specifying a number of blanks equivalent to the length of the field.
	maxlength	Number	Required only for delimited “string” fields. Specifies the maximum allowable length of the string field.
	len	Number	The length of the field. Only used for fixed length fields and is required for fixed length fields.
	strip	“leading”, “trailing”, “both”, or “none”	Optional attribute used for delimited “string” fields to determine whether and how white space (tabs and spaces) should be stripped from input field. The default value is “none”.

Delimited record schema

When you create a schema for use with a delimited file type, follow these guidelines:

- The delimiter can be any symbol.

  Note: Make sure the delimiter is never part of your data.

- To define the field's nullability, set **nullable** to 'true' or 'false'.
- If the field is nullable, set the default **nullvalue.** This is often set to "" – the null string.
- For the **string** data type, specify the **maxlength.**

An example delimited schema file

```
<RECORD type="delimited" final_delimiter="0x0A">
     <FIELD name="colname1" delimiter="|" datatype="int8" nullable="false"/>
     <FIELD name="colname2" delimiter="|" datatype="int16" nullable="true"
            nullvalue=""/>
     <FIELD name="colname3" delimiter="|" datatype="int32"
            nullable="true"  nullvalue=""/>
     <FIELD name="colname4" delimiter="|" datatype="int64"
            nullable="true"  nullvalue=""/>
     <FIELD name="colname5" delimiter="|" datatype="dfloat"
            nullable="true"  nullvalue=""/>
     <FIELD name="colname6" delimiter="|" datatype="string" maxlength="5"
            nullable="false"                    />
     <FIELD name="colname7" delimiter="|" datatype="date"
            nullable="false"                    />
     <FIELD name="colname7" delimiter="|" datatype="timestamp"
            nullable="false"                    />
     <FIELD name="colname8" delimiter="|"   datatype="uint8"
            nullable="false"                    />
     <FIELD name="colname9" delimiter="|"   datatype="uint16"
            nullable="true"  nullvalue=""/>
     <FIELD name="colname10" delimiter="|" datatype="uint32"
            nullable="true"  nullvalue=""/>
     <FIELD name="colname11" delimiter="|" datatype="uint64"
            nullable="true"  nullvalue=""/>
     <FIELD name="colname12" delimiter="|" datatype="sfloat"
            nullable="true"  nullvalue=""/>
</RECORD>
```

Fixed length record schema

When you create a schema for use with a fixed length records, follow these guidelines:

- Specify the length (“len”) of the record.
- Specify a “len” value for every field. The total of all fields must be equivalent to the record length.
- To define the field’s nullability, set `nullable` to “true” or “false”.
- If the field is nullable, set the default `nullvalue`. Remember that since this is a fixed length field the null value must be the correct length for the field.
- If possible, specify the “final_delimiter” to ensure that input files are at least record-delimited so that RETL can recover in the event that data is corrupted or invalid.

Note: Make sure the final_delimiter is never part of your data.

An example fixed length schema file

```
<RECORD type="fixed"     len=”99”        final_delimiter="0x0A">

     <FIELD name="colname1"    len=”2”    datatype="int8"
            nullable="false"                    />

     <FIELD name="colname2"    len=”5”    datatype="int16"
            nullable="false"/>

     <FIELD name="colname3"    len=”10”  datatype="int32"
            nullable="false"/>

     <FIELD name="colname4"    len=”18”  datatype="int64"
            nullable="false"/>

     <FIELD name="colname5"    len=”6”  datatype="dfloat"
            nullable="true"  nullvalue="NULVAL"/>

     <FIELD name="colname6"    len=”5”   datatype="string" nullable="false"
            />

     <FIELD name="colname7"    len=”8”   datatype="date"
            nullable="false"                    />

     <FIELD name="colname7"    len="14" datatype="timestamp"
            nullable="false"                    />

     <FIELD name="colname8"    len=”2”   datatype="uint8"
            nullable="false"                    />

     <FIELD name="colname9"    len=”4”   datatype="uint16"
            nullable="true"  nullvalue="XXXX"/>

     <FIELD name="colname10"  len=”5”   datatype="uint32"
            nullable="true"  nullvalue="     "/>

     <FIELD name="colname11"  len=”18” datatype="uint64"
            nullable="false"/>

     <FIELD name="colname12"  len="18" datatype="sfloat"
            nullable="false"/>

</RECORD>
```

Consider carefully your **nullvalue**

Because nullvalues can be changed between import and export schemas, it is possible to lose data if you are not careful. However, this property can also be used to assign default values to null fields on export. No matter how you use them, care should be taken when selecting values for a field's nullvalue. Unless you are assigning a default value to a null field on export we strongly recommend against using a value that can actually appear as a valid value within a given dataset.

To illustrate consider the following schema (named "1.schema"):

```
<RECORD type="delimited" final_delimiter="0x0A">
  <FIELD name="colname1" delimiter="|" datatype="int8"
nullable="false" />
  <FIELD name="colname2" delimiter="|" datatype="dfloat"
nullable="true" nullvalue=""/>
</RECORD>
```

Also consider this schema (named "2.schema") where the nullvalue has been modified for the second field:

```
<RECORD type="delimited" final_delimiter="0x0A">
  <FIELD name="colname1" delimiter="|" datatype="int8"
nullable="false" />
  <FIELD name="colname2" delimiter="|" datatype="dfloat"
nullable="true" nullvalue="0.000000"/>
</RECORD>
```

When running through the following flow:

```
<FLOW>
  <OPERATOR type="import">
    <PROPERTY name="inputfile" value="1.dat" />
    <PROPERTY name="schemafile" value="1.schema" />
    <OUTPUT name="import.v" />
  </OPERATOR>
  <OPERATOR type="export">
    <INPUT name="import.v" />
    <PROPERTY name="outputfile" value="2.dat" />
    <PROPERTY name="schemafile" value="2.schema" />
  </OPERATOR>
</FLOW>
```

Where 1.dat contains the following data:

```
2|
3|4.000000
4|5.000000
5|0.000000
```

The following is the resulting output to 2.dat:

```
2|0.000000
3|4.000000
4|5.000000
5|0.000000
```

By using an export schema that has valid values to represent the nullvalue for the second field, this flow has assigned a default value to all of the second field's null values.

Then, if you change the input and schema files as in the following flow:

```
<FLOW>
  <OPERATOR type="import">
    <PROPERTY name="inputfile" value="2.dat" />
    <PROPERTY name="schemafile" value="2.schema" />
    <OUTPUT name="import.v" />
  </OPERATOR>
  <OPERATOR type="export">
    <INPUT name="import.v" />
    <PROPERTY name="outputfile" value="3.dat" />
    <PROPERTY name="schemafile" value="1.schema" />
  </OPERATOR>
</FLOW>
```

3.dat will look like this:

```
  2|
3|4.000000
4|5.000000
5|
```

Configure RFX to print schema output in SchemaFile format

One error prone area in the development of RFX is when developers break up and put together flows. The interface between flows is the schema files that tell RFX how to read/write files. Writing schema files manually is tedious and error prone. We highly recommend that developers use the '-sSCHEMAFILE' command line option to speed up development. For more information about this feature, see the section RFX Command line options.

Chapter 4 – RETL program flow

Program flow overview

The following text and diagram provides an overview of the RETL input-process-output model. The data input operators to RETL processing include the following:

- DBREAD, where data is read directly from a database
- IMPORT, where RETL accepts a data file
- GENERATOR, where RETL itself creates data input

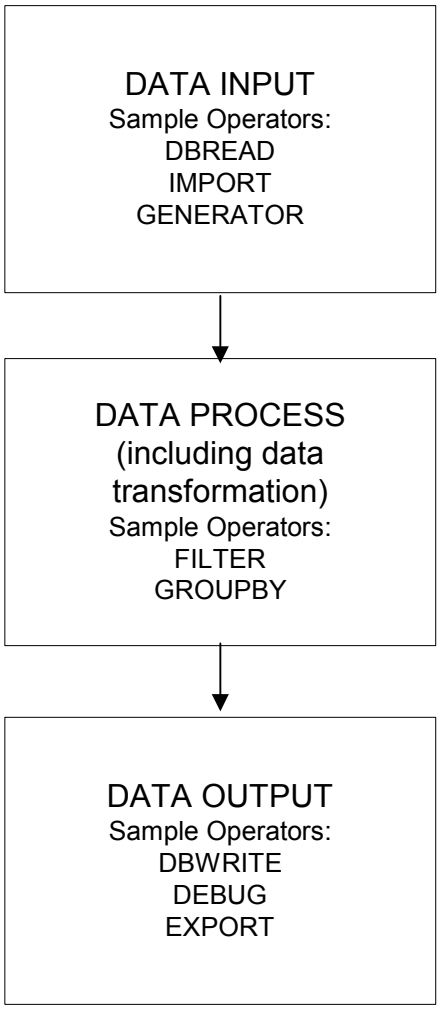
Data process operators commonly include those that transform data in the dataset being processed, including:

- GROUPBY, which can be used to sum values in a table column

The data output process can include the use of the following operators:

- DBWRITE, where RETL writes data directly to the database
- DEBUG, which can be used to print a specific number of records in the dataset to the screen
- EXPORT, which can export data to a flat file

The general flow



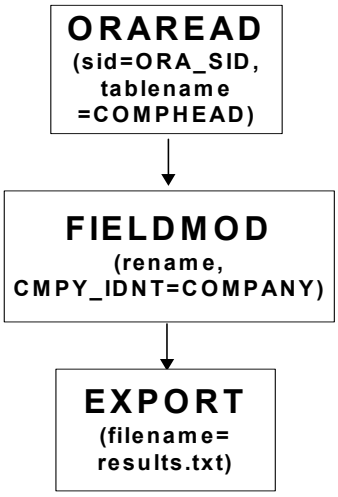
The diagram above is a general representation of the form that all flows take. Every flow must have a Data Input (or Source) and a Data Output (or Sink). These flows may optionally perform transformations on the input data before sending it to the Data Output.

Some things to note about all flows:

- When an operator generates an output dataset, that output dataset must be input into another operator. The data "flows" from one operator to another along the datasets (the arrows between the operators above).
- DATA INPUT operators have no dataset inputs – they get data from outside sources (e.g. Oracle database, DB2 database, TeraData database or flat file).
- Similarly, DATA OUTPUT operators are terminal points within the flow. They do not generate output datasets – rather they export records to an external repository (for example, Oracle, DB2, TeraData databases, or flat files).

In the next several sections we look at specific flows.

A simple flow



The diagram above is one way to represent a simple flow. This flow contains three operators: ORAREAD, FIELDMOD and EXPORT. This flow also contains two datasets – these are the simple arrows that connect the operator boxes above. The arrow from ORAREAD to FIELDMOD means that records flow from the ORAREAD operator to the FIELDMOD. Likewise the arrow from FIELDMOD to EXPORT indicates that the records from the FIELDMOD are passed onto the EXPORT operator.

In this flow diagram, several attributes are also shown to give us more information about what exactly this flow does. This flow pulls the "COMPHEAD" table out of the "ORA_SID" database (via the ORAREAD operator), renames the column "CMPY_IDNT" to "COMPANY" (via the FIELDMOD operator), and exports the resulting records to the file "results.txt" (via the EXPORT operator).

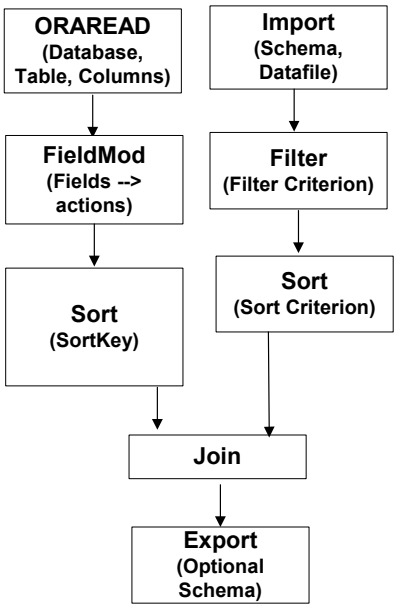
Here is XML for the above flow:

```
<FLOW name="cmpy.flw">
   <OPERATOR type="oraread">
      <PROPERTY name="sid"           value="ORA_SID"/>
      <PROPERTY name="connectstring"
value="RETLdev/develops"/>
      <PROPERTY name="query"   value="select * from
COMPHEAD"/>
      <OUTPUT   name = "data.v"/>
   </OPERATOR>
```

```
   <!-- This is the format for a comment.  They can of
course be
        multi-line, but cannot be nested. -->
   <OPERATOR type="fieldmod">
          <PROPERTY name="rename"
value="CMPY_IDNT=COMPANY"/>
          <INPUT    name = “data.v"/>
          <OUTPUT   name = "comphead.v"/>
   </OPERATOR>
   <OPERATOR type="export">
       <INPUT    name="comphead.v"/>
       <PROPERTY name="outputfile" value="results.txt"/>
   </OPERATOR>
</FLOW>
```

A more complex flow

Flows can get much more complex than the simple flow shown above.



The above flow pulls in data simultaneously from a flat file and Oracle Database. The dataset from the file is filtered based upon the specified filter criterion then sorted. In parallel, the dataset from the Oracle database is modified via the FIELDMOD operator and then sorted. The resulting datasets are joined and the results are output to a flat file via the Export operator.

Several things to note:

- RETL is capable of dealing with multiple data sources within a single flow.
- Although not shown in the above diagram, RETL is capable of outputting results to multiple files and/or database tables.
- RETL can perform complex transformations by combining operators.

Chapter 5 – Database operators

RETL provides read-write operators for three database platforms:

- Oracle
- IBM DB2
- NCR TeraData

Each of the following read-write operators is described in this chapter, along with tag usage example code:

- ORAREAD
- ORAWRITE
- DB2READ
- DB2WRITE
- TERAFEREAD
- TERAWRITE

Note: We use the term DBREAD as shorthand for any one of the ORAREAD, TERAFEREAD or DB2READ operators.

Note: We use the term DBWRITE as shorthand for any one of the ORAWRITE, TERAWRITE or DB2WRITE operators.

ORAREAD

Performs the read operation of data out of the Oracle database.

ORAWRITE

Performs the load operation of a RETL dataset into the Oracle database. The Oracle SQL*Loader utility (sqlldr) is used with this operator. There are two types of loads: direct or conventional.

DB2READ

Performs the read operation of data out of the DB2 database.

DB2WRITE

Performs the load operation of a RETL dataset into the DB2 database. The DB2 load utilities allow two types of loads: direct load using the autoloader utility (db2atld) or the conventional load using the import utility.

TERAFEREAD

Performs the read operation of data out of the TeraData database using FASTEXPORT (fexp).

TERAWRITE

Performs the load operation of a RETL dataset into the TeraData database using mload or fastload from the TeraData utilities. It uses "mload" when writing records to a non-empty table and "fastload" when creating a new table or writing to an empty table.

Read-write operators XML specification table

The following table outlines database read/write operators.

Note: Throughout the RETL Programmer's Guide, 'schema' can apply to the data structure that the RETL applies to a dataset (such as a 'schema file'), or 'schema' can apply to the database owner of a table (the 'schemaowner' property of a database write operator).

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
ORAREAD				
	PROPERTY	dbname	database name (ORACLE_SID)	Name of the Oracle database.
	PROPERTY	sid		An alias for "dbname" above.
	PROPERTY	connectstring	username/password	Database login name and password separated by a forward slash - "/".
	PROPERTY	maxdescriptors	number	Maximum number of columns allowed to be returned from the database. This is an optional property. The default value is 70.
	PROPERTY	query	select statement	The SQL query used to extract the data from the database. SQL syntax must be executable for the database environment. **Note:** If any kind of function is applied to a column in the select statement, the column will be extracted as a DFLOAT even if the column in the database is defined as an int or string type. You may have to perform a CONVERT to change datatypes in order to compensate for this behavior.

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	datetotimestamp	"true" or "false"	This is an optional property. When set to "true" the ORAREAD returns the dates with timestamp included, otherwise dates are returned without the timestamp. The default for this property is "false".
	OUTPUT	name.v		The output dataset name.
ORAWRITE				
	PROPERTY	sid	ORACLE_SID	Name of the Oracle database.
	PROPERTY	dbuserid	username/password	Database login name and password separated by a forward slash - "/".
	PROPERTY	maxdescriptors	number	Maximum number of columns allowed to be written to the database.
	PROPERTY	schemaowner	schema name	The database owner of the table.
	PROPERTY	tablename	table name	Name of the Oracle table to which the data is written.
	PROPERTY	method	"direct" or "conventional"	This is an optional property to specify the loading method. The default is "conventional". Direct - load the data using SQL*Loader utility with direct=true. Conventional - Load the data using SQL*Loader utility with direct=false.
	PROPERTY	mode	"append" or "truncate"	This is an optional property. Append – add the new data in addition to existing data on the target table. Truncate – delete the data from the target table before loading the new data. The default value is "append".

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
		createtablemode	"recreate" or "create"	This is an optional property that allows the developer to specify table creation mode. If this property is not given, the flow will fail if the specified table does not already exist. recreate – drops the table and recreates the same table before writing the data. create – use this option when the target table being written to does not exist. It will create the table before writing the data.
	PROPERTY	sp_prequery	string	This is a special query that allows the flow developer to specify a stored procedure to be run **prior** to the main query that ORAWRITE executes. The current implementation does not allow variables to be specified nor does it allow data to be returned.
	PROPERTY	sp_postquery	string	This is a special query that allows the flow developer to specify a stored procedure to be run **after** to the main query that ORAWRITE executes. The current implementation does not allow variables to be specified nor does it allow data to be returned.
	INPUT	name.v		The input dataset name.
DB2READ				
	PROPERTY	dbname	database name	Name of the DB2 database.
	PROPERTY	userid	username	Database login name.
	PROPERTY	password	password	Database login password.
	PROPERTY	maxdescriptors	number	Maximum number of columns allowed to be returned from the database.

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	query	select statement	The SQL query used to extract the data from the database. SQL syntax must be executable for the database environment.
	OUTPUT	name.v		The output dataset name.
DB2WRITE				
	PROPERTY	dbname	database name	Name of the DB2 database.
	PROPERTY	userid	username	Database login name.
	PROPERTY	password	password	Database login password.
	PROPERTY	maxdescriptors	number	Maximum number of columns allowed to be written to the database.
	PROPERTY	schemaowner	schema name	The database owner of the table.
	PROPERTY	tablename	table name	Name of the DB2 table where the data is written.
	PROPERTY	tablespace	tablespace name	This option applies when using the recreate or created option. The new table create is assigned to the specified tablespace. Otherwise the tablespace defaults to the USERID.
	PROPERTY	method	“direct” or “conventional”	Direct: loads the data using autoloader utilities in DB2. Conventional: loads the data using the db2load utilities in DB2.

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	mode	"insert", "replace", "restart", or "terminate"	This optional property performs the appropriate action as per the "IBM DB2 Universal Command Reference version 7": insert - Adds the loaded data to the table without changing the existing table data (default). replace - Deletes all existing data from the table, and inserts the loaded data. restart – Restarts a previously interrupted load operation. The load operation automatically continues from the last consistency point in the load, build, or delete phase. terminate - Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started.
	PROPERTY	createtablemode	"recreate" or "create"	Recreate: drops the table and recreates the same table, before writing the data. Create: use this option when the target table being written to does not exist. It creates the table before writing the data. If this property is not specified, RETL assumes the target table already exists and will write to the table immediately.
	PROPERTY	primaryindex	comma separated list of columns.	This optional parameter is a comma-separated list of indexes that allows a new table to be created with primary indexes. If this option is not specified then the table will be created without indexes.
	INPUT	name.v		The input dataset name.
TERAFEREAD				
	PROPERTY	dbname	database name	Name of the TeraData server.

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	userid	username	Database login name.
	PROPERTY	password	password	Database login password.
	PROPERTY	maxdescriptors	number	Maximum number of columns allowed to be returned from the database.
	PROPERTY	query	SQL Select statement	The SQL query used to extract the data from the database. SQL syntax must be executable for the database environment.
	PROPERTY	schemaowner	schema name	Schema owner of the schema where the TERAFEREAD operator will create log tables.
	PROPERTY	numberofsessions	number	Number of read sessions per AMP (a database node). If not specified, the default is one (1) session per AMP.
	OUTPUT	name.v		The output dataset name.
TERAWRITE				
		dbname	database name	Name of the TeraData database.
		userid	username	Database login name.
		password	password	Database login password.
		maxdescriptors	number	Maximum number of columns allowed to be written to the database.
		schemaowner	schema name	The database owner of the table.
		tablename	table name	Name of the TeraData table where the data is written.
		logschema	schema name	The database where the terawrite operator writes log and error tables. These tables are dropped after a successful load.

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
		createtablemode	"recreate" or "create"	Recreate – drops the table and recreates the same table before writing the data. Create – Use this option when the target table being written to does not exist. It creates the table before writing the data. If this property is not specified, RETL assumes the target table is already there and it will try to write to the table immediately.
		primaryindex	comma separated column names.	An optional parameter is a comma-separated list of indexes that allows a new table to be created with primary indexes. If this option is not specified then the table will be created without indexes.
	INPUT	name.v		The input dataset name.

Tag usage examples

```
<OPERATOR type="oraread">
      <PROPERTY name="sid"              value="RETLdb"/>
      <PROPERTY name="connectstring"    value="RETLdb/rpassword"/>
      <PROPERTY name="query"            value="select * from rtbl "/>
      <PROPERTY name="maxdescriptors"   value="100"/>
      <PROPERTY name="datetotimestamp" value="false"/>"
      <OUTPUT   name="test.v"/>
</OPERATOR>

<OPERATOR type="orawrite">
<PROPERTY name="sid"          value="RETLdb"/>
<PROPERTY name="dbuserid"     value="username/password"/>
<PROPERTY name="tablename"    value="ORG_LOC_DM"/>
<PROPERTY name="mode"         value="append"/>
<PROPERTY name="method"       value="conventional"/>
<INPUT    name="test.v"/>
</OPERATOR>

<OPERATOR type="db2read">
      <PROPERTY name="userid"   value="RETLdb"/>
      <PROPERTY name="password" value="password1"/>
      <PROPERTY name="dbname"   value="database1"/>
      <PROPERTY name="query"    value="select day_idnt from
   time_day_dm"/>
<OUTPUT   name="test.v"/>
</OPERATOR>

<OPERATOR type="db2write">
      <PROPERTY name="userid"             value="RETLdb"/>
      <PROPERTY name="password"           value="rpassword"/>
      <PROPERTY name="dbname"             value="develops"/>
      <PROPERTY name="schemaowner"        value="RETL"/>
      <PROPERTY name="method"             value="conventional"/>"
      <PROPERTY name="tablename"          value="andy_temp"/>
      <PROPERTY name="mode"               value="replace"/>
      <PROPERTY name="createtablemode"    value="recreate"/>
<OUTPUT   name="test.v"/>
</OPERATOR>
```

```
<OPERATOR type="teraferead">
<PROPERTY name="userid"        value="RETL"/>
<PROPERTY name="password"      value="develops"/>
<PROPERTY name="dbname"        value="msprts01"/>
<PROPERTY name="schemaowner"   value="RETL"/>
<PROPERTY name="query"         value="select * from
 RETLdm.maint_dim_key_dm"/>"
<OUTPUT       name="test.v"/>
</OPERATOR>

<OPERATOR type="terawrite">
<PROPERTY name="userid"        value="RETL"/>
<PROPERTY name="password"      value="develops"/>
<PROPERTY name="dbname"        value="msprts01"/>
<PROPERTY name="schemaowner"   value="RETL"/>
<PROPERTY name="logschema"     value="RETLlog”/>"
<PROPERTY name="tablename"     value="RETLlog.temp"/>
<INPUT     name="test.v"/>
</OPERATOR>
```

Chapter 6 – Input and output operators

Each of the following input and output operators is described in this chapter, along with tag usage example code:

- DEBUG
- NOOP
- EXPORT
- IMPORT

DEBUG

Reads the records from a single input dataset and prints all records to the screen.

NOOP

The NOOP acts as a terminator for datasets. In short it does nothing with the data – just throws it away.

EXPORT

Writes a RETL dataset to a flat file, either as delimited or fixed-length records, depending upon the schema.

IMPORT

Imports a flat file into RETL, translating the data file into a RETL dataset.

Input and output operators XML specification table

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
DEBUG				
	INPUT	name.v		The input dataset name.
NOOP				
	INPUT	name.v		The input dataset name.
IMPORT				
	PROPERTY	inputfile	input data file	The name of the text file to import into RETL.
	PROPERTY	schemafile	schema file	The name of the schema file describing the layout of the data file.
	PROPERTY	rejectfile	reject file	The name of the file where records that do not match the input schema are deposited.
	PROPERTY	allowedrejects	number >= 0	Optional property. A value greater than zero indicates how many errors are allowed before rfx exits with a critical error. A zero indicates that any number of errors will be accepted (this is the default setting).
	OUTPUT	name.v		The output dataset name.
EXPORT				
	PROPERTY	outputfile	output data file	The output text file name.
	PROPERTY	outputmode	“overwrite” or “append”	Optional property specifies whether export overwrites an existing file or appends to it. Default is to overwrite.

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	schemafile	schema file	Optional property. The schema file describing the layout of the output file. If not specified, uses current schema (as output from the previous dataset). This property may be used to reorder and/or drop fields from the output and reformat the fixed length, delimiter and/or nullvalue characteristics of the schema. The default delimiter ('|') is used if no schema file is specified.
	INPUT	name.v		The output dataset name.

Tag usage examples

```
<OPERATOR  type="import" name="import1,0">
<PROPERTY name="inputfile"    value="import.dat"/>
<PROPERTY name="schemafile"   value="ImportOp.schema"/>
</OPERATOR>

<OPERATOR  type="import" name="import1,0">
<PROPERTY name="inputfile"    value="import.dat"/>
<PROPERTY name="schemafile"   value="ImportOp.schema"/>
<OUTPUT   name=”test.v"/>
</OPERATOR>

<OPERATOR  type="export" name="export1,0">
<PROPERTY name="outputfile"   value="output.dat"/>
<PROPERTY name="schemafile"   value="outputOp.schema"/>
<INPUT    name="test.v"/>
</OPERATOR>
```

Chapter 7 – Join operators

RETL provides the following join operators, each of which is described in this chapter:

- INNERJOIN
- LEFTOUTERJOIN
- RIGHTOUTERJOIN
- FULLOUTERJOIN
- LOOKUP

Note: For INNERJOIN, LEFTOUTERJOIN, RIGHTOUTERJOIN and FULLOUTERJOIN the input datasets must be sorted on the key.

INNERJOIN

Transfers records from both input datasets whose key fields contain equal values to the output dataset. Records whose key fields do not contain equal values are dropped.

LEFTOUTERJOIN

Transfers all values from the left dataset, and transfers values from the right dataset only where key fields match. The operator drops the key field from the right dataset. Otherwise, the operator writes default values.

RIGHTOUTERJOIN

Transfers all values from the right dataset, and transfers values from the left dataset only where key fields match. The operator drops the key field from the left dataset. Otherwise, the operator writes default values.

FULLOUTERJOIN

For records that contain key fields with identical and dissimilar content, the FULLOUTERJOIN operator transfers records from both input datasets to the output dataset.

LOOKUP

Similar to the INNERJOIN operator. However, there is no need to do a sort on the input datasets. Looks up a value from one dataset whose key fields match the lookup dataset and outputs the matching values.

When using a lookup in multiple partitions be sure to “hash” the source dataset but not the lookup dataset. Remember also that the lookup dataset must be small enough to fit in memory otherwise severe performance problems may result. If in doubt as to whether the lookup dataset will fit in memory always use INNERJOIN or OUTERJOIN.

Join operators XML specification table

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
INNERJOIN				
	PROPERTY	key	key field	Key columns to be joined.
	INPUT	inputname1.v		Input dataset 1 – sorted on the key columns.
	INPUT	inputname2.v		Input dataset 2 – sorted on the key columns.
	OUTPUT	outputname.v		The output dataset name.
LEFTOUTERJOIN RIGHTOUTERJOIN FULLOUTERJOIN				
	PROPERTY	key	key field	Key columns to be joined.
	PROPERTY	nullvalue	column name	The default value that is assigned to the null column.
	INPUT	inputname1.v		Input dataset 1. (left dataset) – sorted on the key columns
	INPUT	inputname2.v		Input dataset 2. (right dataset) – sorted on the key columns
	OUTPUT	name.v		The output dataset name.
LOOKUP				
	PROPERTY	tablekeys	key field	Key columns to be looked up.
	PROPERTY	ifnotfound	“reject”, “continue”, “drop”, or “fail”	What to do with the record if the result did not match. If not specified, this option will default to “fail”.
	PROPERTY	allowdups	“true” or “false”	Optional property defaults to “true”. This property allows the lookup to return more than 1 record if the lookup dataset has more than 1 matching set of keys.
	INPUT	inputname1.v		First input dataset is always the data that needs to be processed.
	INPUT	inputname2.v		Second input dataset is always the “lookup” dataset.
	OUTPUT	resdataset.v		The first output is the successful lookup output dataset name.

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	OUTPUT	rejdataset.v		The second will contain all records not matching the “tablekeys” property. This needs to be specified if the “ifnotfound” option is set to “reject”.

Tag usage examples

```
<OPERATOR  type="innerjoin" name="innerjoin,0">
<PROPERTY  name="key"          value="LOC_KEY"/>
<INPUT           name="test_1.v"/>
<INPUT           name="test_2.v"/>
<OUTPUT       name="output.v"/>
</OPERATOR>

<OPERATOR  type="leftouterjoin" name="leftouterjoin,0">
<PROPERTY  name="key"          value="LOC_KEY"/>
<PROPERTY  name="nullvalue"    value="PROD_SEASN_KEY=-1"/>
<INPUT     name="left.v"/>
<INPUT     name="right.v"/>
<OUTPUT   name="result.v"/>
</OPERATOR>

<OPERATOR  type="rightouterjoin" name="rightouterjoin,0">
<PROPERTY  name="key"          value="LOC_KEY"/>
<PROPERTY  name="nullvalue"    value="PROD_SEASN_KEY=-1"/>
<INPUT     name="right.v"/>
<INPUT     name="left.v"/>
<OUTPUT    name="output.v"/>
</OPERATOR>


<OPERATOR  type="fullouterjoin" name="fullouterjoin,0">
<PROPERTY  name="key"          value="LOC_KEY"/>
<PROPERTY  name="nullvalue"    value="PROD_SEASN_KEY=-1"/>
<INPUT     name="right.v"/>
<INPUT     name="left.v"/>
<OUTPUT    name="output.v"/>
</OPERATOR>

<OPERATOR  type="lookup">
<PROPERTY  name="tablekeys"    value="LOC_KEY,SUPP_KEY"/>
<PROPERTY  name="ifnotfound"   value="reject"/>
<INPUT     name="dataset.v"/>
<INPUT     name="lookupdataset.v"/>
<OUTPUT    name="result.v"/>
<OUTPUT    name="reject.v"/>
</OPERATOR>

<OPERATOR  type="lookup">
<PROPERTY  name="tablekeys"    value="LOC_KEY"/>
<PROPERTY  name="ifnotfound"   value="continue"/>
<PROPERTY  name="allowdups"    value="true"/>
<INPUT     name="dataset.v"/>
<INPUT     name="lookupdataset.v"/>
<OUTPUT    name="result.v"/>
</OPERATOR>
```

Chapter 8 – Sort and merge operators

RETL provides the following sort and merge operators, each of which is described in this chapter:

- COLLECT and FUNNEL
- SORTCOLLECT and SORTFUNNEL
- HASH
- SORT
- MERGE

COLLECT and FUNNEL

Both operators combine records from input datasets as they arrive. This can be used to combine records that have the same schema from multiple sources. Note that these operators are sometimes used implicitly by RETL to rejoin datasets that have been divided during partitioning in parallel processing environments (where the number of RETL partitions is greater than one).

SORTCOLLECT and SORTFUNNEL

Like COLLECT and FUNNEL these operators combine records from input datasets. These operators maintain sorted order of multiple datasets already sorted by the key fields. Records are collected in sorted order using a merge sort algorithm. If incoming records are unsorted, COLLECT followed by the SORT operator should be used instead.

HASH

The hash-partitioning operator examines one or more fields of each input record, called “hash key” fields, to assign records to a processing node. Records with the same values for all hash key fields are assigned to the same processing node. This type of partitioning method is useful when grouping or sorting data to perform a processing operation.

SORT

Sorts the records in the RETL dataset based on one or more key fields in the dataset.

MERGE

Merges input datasets columns record by record. The number of records contained in the inputs’ datasets must match. Columns are ordered by the order of the input datasets, first input dataset, second input dataset and so on.

Sort and merge operators XML specification table

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
COLLECT FUNNEL				**Note:** These operators are aliases for each other. We use COLLECT as a general rule but either operator will work.
	INPUT	inputname1.v		Input dataset 1.
	INPUT	inputname2.v		Input dataset n. There may be 0 or more additional datasets.
	OUTPUT	outputname.v		The output dataset name.
HASH				
	PROPERTY	key	key field	Key columns to be hashed.
	INPUT	inputname.v		Input dataset.
	OUTPUT	name.v		The output dataset name.
SORT				
	PROPERTY	key	key field	Key columns to sort by.
	PROPERTY	order	"desc" or "asc"	Optional property. Set to "desc" for descending sort order and "asc" for ascending sort order. Default value is "asc".
	PROPERTY	removedup	"true"	Optional value. If set then duplicate records will be removed. Default is false.
	PROPERTY	tmpdir	Directory	Optional property. We recommend that this property not be specified. Temporary directories should be specified in the rfx.conf. This property allows one to override the temporary directory to use for the sort command. **Note:** If used, the directory should be periodically cleaned as the TEMPDIR from the rfx.conf file (see Chapter 2 above).
	INPUT	inputname.v		Input dataset.
	OUTPUT	name.v		The output dataset name.
SORTCOLLECT SORTFUNNEL				**Note:** These operators are aliases for each other. We use SORTCOLLECT as a general rule but either operator will work.
	PROPERTY	key	key field	Key columns to sort by.

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	order	"desc" or "asc"	Optional property defaults describes the sort order of the incoming key fields. Default value is "asc" for ascending order. Specify "desc" if keys are sorted in descending order.
	INPUT	inputname1.v		Input dataset 1.
	INPUT	inputname_n.v		Input dataset n – can be 0 or more datasets.
	OUTPUT	resultdataset.v		Output dataset.
MERGE				
	INPUT	inputname1.v		Input dataset 1.
	INPUT	inputname_n.v		Input dataset n – this can be more than 2 datasets.
	OUTPUT	resultdataset.v		The output dataset.

Tag usage examples

```
<OPERATOR  type="collect">
      <INPUT      name="test_1.v"/>
      <INPUT      name="test_2.v"/>
      <INPUT      name="test_2.v"/>
      <OUTPUT     name="output.v"/>
</OPERATOR>

<OPERATOR  type="hash">
      <PROPERTY   name="key"  value="LOC_KEY"/>
      <INPUT      name="left.v"/>
      <OPERATOR   type="sort" >
            <PROPERTY   name="key"  value="LOC_KEY"/>
            <OUTPUT     name="result.v"/>
      </OPERATOR>
</OPERATOR>

<OPERATOR  type="sortcollect">
      <PROPERTY name="key" value="LOC_KEY"/>
      <INPUT    name="test_1.v"/>
      <INPUT    name="test_2.v"/>
      <OUTPUT   name="output.v"/>
</OPERATOR>

<OPERATOR type="merge">
      <INPUT      name="test_1.v"/>
      <INPUT      name="test_2.v"/>
      <OUTPUT     name="output.v"/>
</OPERATOR>
```

Chapter 9 – Mathematical operators

RETL provides the following mathematical operators, each of which is described in this chapter:

- BINOP
- GROUPBY
- GROUPBY on multiple partitions

BINOP

Performs basic algebraic operations on two fields. Addition, subtraction, multiplication and division are supported for all numeric types. The left and right operands may also be defined as constants. Note that division by zero produces a null value. Only addition is supported for string fields; this becomes concatenation. Binop is not currently supported for date fields.

GROUPBY

The input to the GROUPBY operator is a dataset to be summarized; the output is a dataset containing one record for each group in the input dataset. Each output record contains the fields that define the group and the output summaries calculated by the operator. One simple summary is a count of the number of records in each group. Another kind of summary is a statistical value calculated on a particular field for all records in a group.

Note: If all rows in a column are null, the GROUPBY operator will provide a null column total.

GROUPBY on multiple partitions

When running RETL using multiple partitions, run the GROUPBY operator with the 'parallel' property set to 'true' to get the performance increase from the multiple partitions. A HASH/SORT needs to be performed before the parallel GROUPBY. If the same KEYS are used to GROUPBY that were used for the HASH/SORT, data with the same KEY set will not be spread to multiple partitions. If the next operator is a serial operator, there will be an implied FUNNEL operator called by RETL after the GROUPBY to gather the data from the different partitions together into a single dataset. If the next operator is a parallel operator, RETL will not call a FUNNEL operator and the previous HASH/SORT will be maintained. If not using the same KEYS to GROUPBY, a SORTFUNNEL is required to collect the data from the multiple partitions back into a single, sorted dataset. A final serial GROUPBY must then be performed because data may have been spread across multiple partitions and not included in the GROUPBY calculation. If the next operator is a serial operator, then nothing more needs to be done. If the next operator is a parallel operator, another HASH/SORT must be performed because the HASH/SORT from the parallel GROUPBY will not be maintained because a serial GROUPBY was performed after that.

Mathematical operators XML specification table

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
BINOP				
	INPUT	inputname.v		Input dataset.
	PROPERTY	left / constleft	field name or constant value	If property name = 'left' then the column name for the left operand must be specified. If property name = 'CONSTLEFT' then a constant value must be specified.
	PROPERTY	right / constright	field name or constant value.	If property name = 'right' then the column name for the right operand must be specified. If property name = 'CONSTRIGHT' then a constant value must specified.
	PROPERTY	dest	destination Field	The result destination field name.
	PROPERTY	desttype	type	Optional field. The type of the destination field. The default value is the type of the left field.
	PROPERTY	operator	'+' or '-' or '*' or '/'	'+' for addition '-' for subtraction '*' for multiplication '/' for division
	OUTPUT	outputname.v		The output dataset name.
GROUPBY				
	INPUT	Input.v		Input dataset.
	PROPERTY	parallel	"true" or "false"	True – If running in multiple partitions. False – If running in a single partition (default).
	PROPERTY	key	key field	Key columns to groupby.
	PROPERTY	reduce	column name	Column to be summed, maxed, etc. Multiple "reduce" properties can be specified, but associated "min", "max", "sum", "first", "last", and "count" properties must be specified immediately following each reduce property.

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	min, max, sum, first, last, count	destination Column	Type of groupby operation and assigns the column to a new column name.
	OUTPUT	name.v		The output dataset name.

Tag usage examples

```
 <OPERATOR  type="binop">
  <INPUT   name="import.v" />
  <PROPERTY name="left"        value="field1" />
  <PROPERTY name="operator"    value="+" />
  <PROPERTY name="constright" value="2" />
  <PROPERTY name="dest"        value="destfield" />
  <OUTPUT   name="binop.v" />
 </OPERATOR>

<OPERATOR  type="binop">
  <INPUT   name="import.v" />
  <PROPERTY name="left"        value="field1" />
  <PROPERTY name="operator"    value="-" />
  <PROPERTY name="right"       value="field2" />
  <PROPERTY name="dest"        value="destfield" />
  <OUTPUT   name="binop.v" />
 </OPERATOR>

<OPERATOR  type="groupby">
<INPUT           name="input_1.v"/>
<PROPERTY  name="parallel" value="true"/>
<PROPERTY  name="key"      value="sex"/>
<PROPERTY  name="reduce"   value="age"/>
<PROPERTY  name="min"      value="min_age"/>
<PROPERTY  name="max"      value="max_age"/>
<PROPERTY  name="first"    value="first_age"/>
<PROPERTY  name="last"     value="last_age"/>
<PROPERTY  name="count"    value="count_age"/>
<PROPERTY  name="sum"      value="sum_age"/>
<PROPERTY  name="reduce"   value="birthday"/>
<PROPERTY  name="min"      value="min_birthDay"/>
<PROPERTY  name="max"      value="max_birthDay"/>
<PROPERTY  name="first"    value="first_birthDay"/>
<PROPERTY  name="last"     value="last_birthday"/>
<PROPERTY  name="count"    value="count_birthday"/>
<OUTPUT    name="output.v"/>
</OPERATOR>
```

The following example shows what happens if the dataset was previously hashed and sorted on a different key than the group by key in the “groupby” operator, and if the groupby was run in multiple partitions.

In this case an additional “sortcollect” and “groupby” operator needs to be used to guarantee the correct result.

```
<OPERATOR  type="hash">
    <PROPERTY  name="key"       value="ZIP_CODE"/>
    <PROPERTY  name="key"       value="NAME"/>
    <OPERATOR  type="sort">
        <PROPERTY  name="key"       value="ZIP_CODE"/>
        <PROPERTY  name="key"       value="NAME"/>
        <OUTPUT    name="output1.v"/>
    </OPERATOR>
</OPERATOR>
<OPERATOR  type="groupby">
   <INPUT     name="output1.v"/>
   <PROPERTY  name="parallel" value="true"/>
   <PROPERTY  name="key"      value="sex"/>
   <PROPERTY  name="reduce"   value="age"/>
   <PROPERTY  name="count"    value="count_age"/>
   <PROPERTY  name="reduce"   value="birthday"/>
   <PROPERTY  name="max"      value="max_birthday"/>
   <OPERATOR  type="sortcollect">
       <PROPERTY  name="key"         value="sex"/>
       <OPERATOR  type="groupby">
          <PROPERTY  name="parallel" value="true"/>
          <PROPERTY  name="key"      value="sex"/>
          <PROPERTY  name="reduce"   value="age"/>
          <PROPERTY  name="count"    value="count_age"/>
          <PROPERTY  name="reduce"   value="birthday"/>
          <PROPERTY  name="max"      value="max_birthday"/>
          <OUTPUT    name="output2.v"/>
       </OPERATOR>
    </OPERATOR>
</OPERATOR>
```

Chapter 10 – Structures and data manipulation operators

RETL provides the following operators for the manipulation of structures and data, each of which is described in this chapter:

- CONVERT
- FIELDMOD
- FILTER
- GENERATOR
- REMOVEDUP

CONVERT

The convert operator is used to convert an existing datatype to a new datatype from an input dataset. The following paragraph describes the syntax of the CONVERT property.

- The root tag <CONVERT>, containing one or more <CONVERTFUNCTION> or <TYPEPROPERTY> tags
- The <CONVERT> tag requires the following attributes:
  - ***Destfield***: destination field name or new name.
  - ***Sourcefield***: original field name or old field name.
  - ***Newtype***: new datatype
  - The <CONVERTFUNCTION> tag allows conversion of a column from one data type to a different data type and has the following attributes:
    - ***name -*** The value of name of the conversion function determines what conversion is done. The name generally follows the form: <typeTo>_from_<typeFrom>. Conversions are defined between all numeric values and from numbers to strings as well. The table below shows some example conversions.
  - The <TYPEPROPERTY> tag allows turning the column into nullable or not null and has the following attributes:
    - ***name***: nullable
    - ***value***: “true” or “false”

Conversion functions

There are many conversion functions to allow conversion between types. Much of this is done with default conversions between types (see Appendix A – Default Conversions). When using a default conversion you simply specify "default" as the name of the conversion.

Here are the other (non-default) conversion functions:

Conversion Name	Description
make_not_nullable	Converts nullable field to not null; requires functionarg tag. See the "Tag usage examples," section later in this chapter.
make_nullable	Converts non-nullable field to nullable field; requires functionarg tag.
string_length	Converts a string to a UINT32 with a value equivalent to the string's length.
string_from_int32	Converts to STRING from INT32.
string_from_int64	Converts to STRING from INT64.
string_from_int16	Converts to STRING from INT16.
string_from_int8	Converts to STRING from INT8.
int32_from_dfloat	Converts to INT32 from DFLOAT rounding the result to the nearest integer value.
int64_from_dfloat	Converts to INT64 from DFLOAT rounding the result to the nearest integer value.
string_from_date	Converts to STRING from DATE.
dfloat_from_string	Converts to DFLOAT from STRING.
string_from_dfloat	Converts to STRING from DFLOAT.

Note: RETL doesn't handle invalid conversions well. Care should be taken to ensure that the data is well formed by the time a conversion takes place. Currently overflow and/or underflow conversion errors are not detected by RETL. For example, converting the INT16 "-12" to a UINT16 will result in undefined behavior.

FIELDMOD

Used to remove, duplicate, drop and rename columns within RETL. There are several optional parameters that one can use in the fieldmod operator. Here are some notes on how to use the fieldmod operator:

- Use the keep property when you need to drop a large number of columns and/ or ignore columns that you don't know about (allowing your flows to be more resilient to change). Any columns that are not specified in the "keep" property will be dropped. The column names are separated by a space. Multiple keep properties can be specified.
- If you want keep most of columns and just drop a few columns use the "drop" property. The columns to drop are separated by a space. The drop property is processed after the keep property, which means a column is dropped if it is specified on both keep and drop property. Multiple drop properties can be specified. Duplicated columns in single or multiple drop properties will generate an error (e.g. 'Delete of field failed (field 'Emp_Age' not found)'" since the column has been dropped).
- The "rename" and "duplicate" properties affect the dataset after keep and drop properties have been processed. If you attempt to rename or duplicate a column that has been dropped, RFX will generate an error (e.g. Error: the column 'Emp_Name' has been dropped and can not be duplicated).

FILTER

Used to filter a dataset into two categories:

- Those that meet the filter criterion and,
- Those that do not.

Some uses might be to: filter records where a field must be equivalent to a certain string, filter on records where a certain field is less than 10, or filter on records where two fields are equal. This is somewhat like the WHERE clause of a SQL statement.

GENERATOR

Used to create new datasets for testing or for combining with other data sources. It can act as a standalone operator that generates the dataset from scratch, or as an operator that can add fields to an existing dataset. The following paragraph describes the syntax of the "SCHEMA" property, which in the example below, is represented within the **CDATA** tag.

The SCHEMA property should specify the XML that will generate the fields. The document type definition (DTD) for the XML can be found in the file: "generate.dtd." The basic outline is:

- The root tag <GENERATE>, containing one or more <FIELD> tags
- The <FIELD> tag specifies the "name" and "type" of the field as attributes, and can contain exactly one field generation tag. Field generation tags include:
  - <SEQUENCE> with the following attributes:
    - ***init***: starting number in sequence. The default is "0".
    - ***incr***: increment to add for each record in sequence. The default is "1".
    - ***limit***: max number, once reached, will start from beginning. Optional, unlimited if not specified.
    - ***partnum_offset***: (true/false) will add the partition number to the init value for parallel operation. The default is false.
    - ***partcount_incr***: (true/false) will multiply the incr value by the number of partitions. The default is false.
  - The <CONST> tag allows the specification of a single attribute, *value*, which is the value that the field will take on.
  - The <VALUELIST> tag has no attributes, but has multiple <CONST> tags within it, specifying the list of values that should be cycled through

The following are the only data types that can be used within the GENERATOR operator:

- int8
- int16
- int32
- int64
- dfloat
- string
- date

REMOVEDUP

The REMOVEDUP operator performs a record-by-record comparison of a sorted dataset to remove duplicate records. Duplicates are determined based upon the key fields specified.

Structures and data manipulation operators XML specification table

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
CONVERT				
	INPUT	input.v		This property is optional. If specified, this is the input dataset.
	PROPERTY	convertspec		Describes the new column data structure to be converted. (See example for more detail.)
	OUTPUT	name.v		The output dataset name.
FIELDMOD				
	INPUT	input.v		This property is optional. If specified, this is the input dataset.
	PROPERTY	keep	column_name	This optional property is a space-separated list of the columns to be kept - the named fields will be retained and all others will be dropped. This property may be specified multiple times.
	PROPERTY	drop	column_name	This optional property is a space-separated list of the columns to be dropped - the named fields will be dropped and all other fields will be retained. This property may be specified multiple times.
	PROPERTY	rename	new_column_name =old_column_name	Column to be renamed. The new column name is separated by an equal sign from the old column name.
	PROPERTY	duplicate	column_name =target_colmn_name	Duplicate one column to another column. The source column is separated by an equal sign from the target column name.
	OUTPUT	name.v		The output dataset name.

<table>
<tr><th>OPERATOR Type (attribute)</th><th>Sub-Tag Name</th><th>Property Name</th><th>Property Value</th><th>Description</th></tr>
<tr><td>FILTER</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>INPUT</td><td>inputname.v</td><td></td><td>Input dataset.<table><tr><th>Operation</th><th>Description</th></tr><tr><td>"NOT"</td><td>Logical NOT</td></tr><tr><td>"AND"</td><td>Logical AND</td></tr><tr><td>"OR"</td><td>Logical OR</td></tr><tr><td>"IS_NULL"</td><td>True if field is null.</td></tr><tr><td>"IS_NOT_NULL"</td><td>True if field is not null.</td></tr><tr><td>"GE"</td><td>Logical >=</td></tr><tr><td>"GT"</td><td>Logical ></td></tr><tr><td>"LE"</td><td>Logical <=</td></tr><tr><td>"LT"</td><td>Logical <</td></tr><tr><td>"EQ"</td><td>True if right and left fields are equal.</td></tr><tr><td>"NE"</td><td>True if right and left fields are not equal.</td></tr></table></td></tr>
<tr><td></td><td>PROPERTY</td><td>rejects</td><td>“true” or “false”</td><td>Optional filter value whose default value is false. If true then the operator will expect a second OUTPUT to be specified into which rejected (i.e. filtered) records will be deposited.</td></tr>
<tr><td></td><td>OUTPUT</td><td>name.v</td><td></td><td>The first output dataset specified contains all of the records for which the filter expression was true.</td></tr>
<tr><td></td><td>OUTPUT</td><td>reject.v</td><td></td><td>This property is required if the “rejects” property (above) is specified. This output dataset is the set of records that did not meet the filter condition.</td></tr>
<tr><td>GENERATOR</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>INPUT</td><td>input.v</td><td></td><td>This property is optional. If specified, this is the input dataset.</td></tr>
<tr><td></td><td>PROPERTY</td><td>parallel</td><td>“true” or “false”</td><td>True – if generator option should be run in parallel.
False – if generator option should be run in a single partition (default).</td></tr>
</table>

OPERATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	numrecords	1..n	This optional field is used when no INPUT is specified to determine how many records are generated. If an input is specified this value is ignored. The default value is 1.
	PROPERTY	schema		Describes the new column data structure to be generated. (See example for more detail.)
	OUTPUT	name.v		The output dataset name.
REMOVEDUP				
	INPUT	input.v		This property is optional. If specified, this is the input dataset.
	PROPERTY	key	column_name	Columns that have the same data to be removed. More than one column property can be specified.
	PROPERTY	keep	first or last	Either keeps the first or last data if key fields are the same. It will default to first.
	OUTPUT	name.v		The output dataset name.

Tag usage examples

```
<OPERATOR  type="filter">
     <INPUT       name="input.v" />
     <PROPERTY name="filter"
                     value="SSN = 123456789 AND
                            SALARY > 124.22 AND DOB >
                           19981010"/>
     <OUTPUT   name="valid.v" />
     <OUTPUT   name="reject.v" />
 </OPERATOR>

<OPERATOR type="generator">
     <PROPERTY  name="numrecords" value="5"/>
     <PROPERTY  name="schema">
    <![CDATA[
               <GENERATE>
                  <FIELD name="DM_RECD_LOAD_DT" type="date"
                           nullable="false">
                     <CONST value="19800101"/>
                  </FIELD>
                  <FIELD name="LOC_KEY"
                           type="int8"  nullable="true">
                     <SEQUENCE init="1" incr="1"/>
                  </FIELD>
               </GENERATE>
            ]]>
     </PROPERTY>
</OPERATOR>
```

```
<OPERATOR type="generator">
   <INPUT       name="input.v"/>
   <PROPERTY name="schema">
           <![CDATA[
              <GENERATE>
                  <FIELD name="F_FULL_PO_COUNT" type="int64"
                           nullable="true">
                       <CONST value="1"/>
                  </FIELD>
                  <FIELD name="F_PART_PO_COUNT" type="int64"
                           nullable="true">
                       <CONST value="0"/>
                  </FIELD>
              </GENERATE>
           ]]>
   </PROPERTY>
  <OPERATOR type="debug"/>
  </OPERATOR>


<OPERATOR type="convert">
      <INPUT       name="input.v"/>
      <PROPERTY name="convertspec">
         <![CDATA[
            <CONVERTSPECS>
               <CONVERT destfield="SUPP_IDNT"
                  sourcefield="SUPPLIER"
                  newtype="string">
                  <CONVERTFUNCTION name="string_from_int64"/>
               </CONVERT>
               <CONVERT destfield="LOC_IDNT"
                  sourcefield="LOCATION"
                  newtype="string">
                  <CONVERTFUNCTION name="string_from_int32"/>
                  <TYPEPROPERTY name=”nullable” value=”true”/>
               </CONVERT>
            </CONVERTSPECS>
         ]]>
      </PROPERTY>
     <OPERATOR type="debug"/>
  </OPERATOR>


<OPERATOR  type="removedup">
      <INPUT       name="input.v" />
      <PROPERTY name="key"          value="SKU_KEY”/>
      <PROPERTY name="key"          value="LOC_KEY”/>
      <PROPERTY name="keep"         value="LAST”/>
      <OUTPUT   name="output.v" />
</OPERATOR>
```

```
<OPERATOR  type="fieldmod">
<INPUT    name="input.v" />
  <PROPERTY  name="keep"        value="Emp_Name Emp_Age"/>
  <PROPERTY  name="keep"        value="Emp_ID"/>
<OUTPUT    name="output.v" />
</OPERATOR>

<OPERATOR  type="fieldmod">
<INPUT    name="input.v" />
  <PROPERTY  name="rename"      value="Emp_Name=EmpName"/>
<OUTPUT    name="output.v" />
</OPERATOR>

<OPERATOR  type="fieldmod">
<INPUT    name="input.v" />
  <PROPERTY  name="drop"        value="Emp_Title"/>
<OUTPUT    name="output.v" />
</OPERATOR>

<OPERATOR  type="fieldmod">
<INPUT    name="input.v" />
  <PROPERTY  name="duplicate"  value="Emp_Name=Name"/>
<OUTPUT    name="output.v" />
</OPERATOR>

<OPERATOR type="convert">
      <INPUT name="inv_sbc_lw_dm.v"/>
      <PROPERTY name="convertspec">
         <![CDATA[
               <CONVERTSPECS>
                  <CONVERT destfield="SBCLASS_KEY"
                     sourcefield="SBCLASS_KEY">
                     <CONVERTFUNCTION name="make_not_nullable">
                        <FUNCTIONARG name="nullvalue"
                                     value="-1"/>
                     </CONVERTFUNCTION>
                  </CONVERT>
               </CONVERTSPECS>
            ]]>
      <OUTPUT name="converted.v"/>
<OPERATOR/>
```

Chapter 11 – Other operators

This chapter describes these RETL operators and their usage:

- COMPARE
- SWITCH
- CHANGECAPTURE
- COPY
- DIFF
- CLIPROWS

COMPARE

Performs a field-by-field comparison of records in two presorted input datasets. This operator compares the values of top-level, non-vector data types such as strings. All appropriate comparison parameters are supported (for example, case sensitivity and insensitivity for string comparisons). COMPARE does not change the schema, partitioning, or content of the records in either of the two input datasets. It transfers both datasets intact to a single output dataset generated by the operator. The comparison results are also recorded in the output dataset.

Appends a 'compare code' to beginning of each record with a value set as follows depending upon the outcome of the comparison:

Code	Other
-1	first dataset FIELD is LESSTHAN second dataset FIELD
0	first dataset FIELD is the same as second dataset FIELD
1	first dataset FIELD is GREATERTHAN second dataset FIELD
-2	record doesn't exist in first dataset
2	record doesn't exist in second dataset

SWITCH

Assigns each record of an input dataset to an output dataset, based on the value of a specified field.

CHANGECAPTURE

Use CHANGECAPTURE to compare two datasets. The assumption is that the flow is comparing an original dataset with a modified copy (changed) dataset. The output is a description of what modifications have been made to the original dataset. Here are the changes that are captured:

- Inserts: records added to the original dataset.
- Deletes: records deleted from the original dataset.
- Edits: records matching the given keys fields whose other fields have values have been modified from the original records.
- Copies: records not changed from the original.

The operator works based on these conditions:

- Two inputs (the original and changed datasets), one output containing the combination of the two (edits are taken from the modified side) and a new field to show what change (if any) occurred between the original and the changed dataset.
- The two input datasets must have the same schemas
- The two datasets must be previously sorted on specified key fields

CHANGECAPTURE identifies duplicate records between two datasets by comparing specified key fields. It then compares the value of the fields between the two datasets. The operator adds a field to the output records that contain one of four possible values, representing the conditions noted earlier (inserts, deletes, edits, or copies). The actual values assigned to each of the possibilities can be changed from the default if desired. It is possible to assign the name of this field and to indicate that records falling into one or more of the above categories be filtered from the output dataset, rather than being passed along with the change code.

Here is an example to illustrate:

Original Dataset		
Key	**Value**	**Other**
John	5	corn
Jill	10	beans
Allie	2	pizza
Frank	14	42fas

Changed Dataset		
Key	**Value**	**Other**
John	5	corn
George	7	olives
Allie	5	pizza
Frank	14	Bogo

The resulting OUTPUT would look like this:

Key	Value	Other	Change	CodeField
John	5	corn	copy	2
Jill	10	beans	deletion	1
George	7	olives	insertion	0
Allie	5	pizza	edit	3
Frank	14	bogo	copy	2

Note that the "Change" column above is just for illustration. Only the "codefield" above would be added to the actual output dataset.

CHANGECAPTURE is often used in conjunction with the SWITCH operator.

COPY

The COPY operator copies a single input dataset to one or more output datasets.

DIFF

The DIFF operator performs a record-by-record comparison of two versions of the same dataset (the 'before' and 'after' datasets) and outputs one dataset that contains the difference between the compared datasets.

CLIPROWS

The CLIPROWS operator performs a record-by-record comparison of a sorted dataset to “clip” a number of rows from a group of records. For each group value of the input keys it will return up to the first or last N entries in the group. It sends to its OUTPUT dataset the given records for further processing and discards the rest.

Other operators XML specification table

OPREATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
COPY				
	INPUT	inputname.v		The input dataset.
	OUTPUT	output1.v		The output dataset 1.
	OUTPUT	output2.v		The output dataset 2.
COMPARE				
	INPUT	input_1.v		Input dataset 1.
	INPUT	input_2.v		Input dataset 2.
	PROPERTY	key	column_name	Column name to be compared between the two datasets defined above.
	OUTPUT	name.v		Output dataset.
CLIPROWS				
	INPUT	input.v		Input dataset.
	PROPERTY	key	column_name	Required property. Indicates that the given column is part of the group definition. Multiple keys can be specified. If you are using partitioning with cliprows, make sure you are using a hash operator with the same key values. That is, the cliprows keys and hash keys must be the same otherwise incorrect results will be returned.
	PROPERTY	which	“first” or “last”	Required property. Indicates that the given column is part of the group definition. Multiple keys can be specified.
	PROPERTY	count	number >= 1	Required property. Indicates the number of rows to clip.
	OUTPUT	name.v		Output dataset.
DIFF				
	INPUT	input_1.v		Input dataset 1.
	INPUT	input_2.v		Input dataset 2.

OPREATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	key	column_name	The key field to perform the comparison. Can be multiple columns.
	PROPERTY	allvalues	'true' or 'false'	True – compares all the values between the datasets. False – Does not compare all values. If this property is 'false' then the 'VALUES' property must be set (default).
	PROPERTY	values	column_name	After the "key" property has been used to join datasets, the "values" property is used to determine differences among the records in the joined datasets. Must be present in both before and after schemas. Determines if the record is a copy or a delete. The fields are processed one by one in the order they are present in the record.
	PROPERTY	copycode	2	The code that is written to the first field of the output dataset. The default is 2 if it is a copy.
	PROPERTY	editcode	3	The code that is written to the first field of the output dataset. The default value is 3.
	PROPERTY	deletecode	1	The code that is written to the first field of the output dataset. The default for a delete is 1.
	PROPERTY	insertcode	0	The code that is written to the first field of the output dataset. The default for an insert is 0.
	PROPERTY	dropcopy	true	To drop a copy record from the output. The default is true.
	PROPERTY	dropedit	true	To drop an edit record from the output. The default is true.
	PROPERTY	dropdelete	true	To drop a delete record from the output. The default is true.
	PROPERTY	dropinsert	true	To drop an insert record from the output. The default is true.

OPREATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	sortascending	true	The operator assumes that the records are sorted in ascending order by default; use this field to change the sort order.
	OUTPUT	name.v		Output dataset.
CHANGECAPTURE				
	INPUT	original.v		Input dataset 1. This is the original dataset.
	INPUT	changed.v		Input dataset 2. This is the modified dataset.
	PROPERTY	key	column_name	Records that match key fields are considered to be the same when determining existence in one dataset versus the other. After the two records are matched via the keys basis the operator can determine if there are copies or edits. If a record exists in the original dataset but not the changed dataset then it is considered to be an delete. Likewise if a record exists in the changed dataset but not the original it is an insertion. There can be multiple instances of this property, allowing the use of composite keys.
	PROPERTY	value	column_name	This property indicates which fields to base the comparisons after records have been matched on key fields. When the given values match the records are considered to be copies. When the values differe the records are considered to be edits. There can be multiple instances of this property, allowing the use of composite values.
	PROPERTY	dropcopy	“true” or “false”	Indicates whether records determined to be copies should be filtered out of the output stream. The default is true.

OPREATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	dropedit	“true” or “false”	Similar to dropcopy, but with regard to edits. The default is false.
	PROPERTY	dropdelete	“true” or “false”	Similar to dropedit, but with regard to deletes.
	PROPERTY	dropinsert	“true” or “false”	Similar to dropedit, but with regard to inserts.
	PROPERTY	copycode	0	The value to set the change code field to for records that are copies. The default is 0.
	PROPERTY	insertcode	1	Similar to copycode, but for inserts. The default is 1.
	PROPERTY	deletecode	2	Similar to copycode, but for deletes. The default is 2.
	PROPERTY	editcode	3	Similar to copycode, but for edits. The default is 3.
	PROPERTY	codefield	column_name	The field name to set the change code field to.
	PROPERTY	sortascending	“true” or “false”	A Boolean property to indicate whether the incoming records are sorted in ascending order (true) on the key fields, or in descending order (false). The default is true.
	OUTPUT	name.v		Output dataset name.
SWITCH				
	INPUT	input.v		Input dataset name.
	PROPERTY	switchfield	field_name	Fieldname to determine the switch.
	PROPERTY	casevalues	“value1 = 0, value2 = 1”	This property assigns the switch value base on a comma separated list of mappings from value to output dataset. Note: output datasets are numbered starting from 0 starting from first specified to the last.
	PROPERTY	discard	“value”	Value to be discarded or dropped.

OPREATOR Type (attribute)	Sub-Tag Name	Property Name	Property Value	Description
	PROPERTY	ifnotfound	“allow”, “fail”, “ignore”	This property determines what to do with the record if the casevalues are not found. Allow – continues to next record and keeps the record in the output dataset. Fail – halts the process (default). Ignore – ignores the record from the dataset and writes to the reject dataset.
	OUTPUT	valid.v		The valid output dataset.
	OUTPUT	reject.v		The reject output dataset.

Tag usage examples

```
 <OPERATOR  type=" copy">
      <INPUT     name="input.v" />
      <OUTPUT    name="copy1.v" />
      <OUTPUT    name="copy2.v" />
 </OPERATOR>

<OPERATOR type="switch">
      <PROPERTY name="switchfield" value="Emp_Age" />
      <PROPERTY name="casevalues" value="35=0, 36=1" />
      <PROPERTY name="discard" value="38" />
      <PROPERTY name="ifnotfound" value="allow" />
      <INPUT name="import1.v" />
      <OUTPUT name="ds1.v" />
      <OUTPUT name="ds2.v" />
      <OUTPUT name="default.v" />
 </OPERATOR>

<OPERATOR  type="compare">
      <PROPERTY  name="key" value="NAME"/>
      <INPUT     name="import1.v"/>
      <INPUT     name="import2.v"/>
      <OUTPUT    name="compare.v"/>
</OPERATOR>

<OPERATOR type="changecapture">
      <PROPERTY name="key"           value="emp_age" />
      <PROPERTY name="value"         value="emp_name" />
      <PROPERTY name="codefield"     value="change_code" />
      <PROPERTY name="copycode"      value="0" />
      <PROPERTY name="editcode"      value="3" />
      <PROPERTY name="deletecode"    value="2" />
      <PROPERTY name="insertcode"    value="1" />
      <PROPERTY name="dropcopy"      value="false" />
      <PROPERTY name="dropedit"      value="true" />
      <PROPERTY name="dropdelete"    value="true" />
      <PROPERTY name="dropinsert"    value="true" />
      <PROPERTY name="allvalues"     value="true" />
      <PROPERTY name="sortascending" value="false" />
      <INPUT    name="import1.v" />
      <INPUT    name="import2.v" />
      <OUTPUT   name="changecapture.v" />
</OPERATOR>

<OPERATOR  type="cliprows">
      <PROPERTY  name="key" value="LOC"/>
      <PROPERTY  name="which" value="first"/>
      <PROPERTY  name="count" value="2"/>
      <INPUT name="import1.v"/>
      <OUTPUT name="cliprows.v"/>
</OPERATOR>
```

```
<OPERATOR type="diff">
      <PROPERTY name="key"          value="emp_name" />
      <PROPERTY name="allvalues"    value="true" />
      <INPUT    name="import1.v" />
      <INPUT    name="import2.v" />
      <OUTPUT   name="diff.v" />
</OPERATOR>
```

Appendix A – Default Conversions

These are the default conversions for use by the CONVERT operator.

from UINT8

Conversion Name	Description
default	Converts to INT8 from UINT8.
default	Converts to UINT16 from UINT8.
default	Converts to INT16 from UINT8.
default	Converts to UINT32 from UINT8.
default	Converts to INT32 from UINT8.
default	Converts to UINT64 from UINT8.
default	Converts to INT64 from UINT8.
default	Converts to FLOAT from UINT8.
default	Converts to DFLOAT from UINT8.

from INT8

Conversion Name	Description
default	Converts to UINT8 from INT8.
default	Converts to UINT16 from INT8.
default	Converts to INT16 from INT8.
default	Converts to UINT32 from INT8.
default	Converts to INT32 from INT8.
default	Converts to UINT64 from INT8.
default	Converts to INT64 from INT8.
default	Converts to FLOAT from INT8.
default	Converts to DFLOAT from INT8.

from UINT16

Conversion Name	Description
default	Converts to UINT8 from UINT16.
default	Converts to INT8 from UINT16.
default	Converts to INT16 from UINT16.
default	Converts to UINT32 from UINT16.
default	Converts to INT32 from UINT16.
default	Converts to UINT64 from UINT16.
default	Converts to INT64 from UINT16.
default	Converts to FLOAT from UINT16.
default	Converts to DFLOAT from UINT16.

from INT16

Conversion Name	Description
default	Converts to UINT8 from INT16.
default	Converts to INT8 from INT16.
default	Converts to UINT16 from INT16.
default	Converts to UINT32 from INT16.
default	Converts to INT32 from INT16.
default	Converts to UINT64 from INT16.
default	Converts to INT64 from INT16.
default	Converts to FLOAT from INT16.
default	Converts to DFLOAT from INT16.

from UINT32

Conversion Name	Description
default	Converts to UINT8 from UINT32.
default	Converts to INT8 from UINT32.
default	Converts to UINT16 from UINT32.
default	Converts to INT16 from UINT32.
default	Converts to INT32 from UINT32.
default	Converts to UINT64 from UINT32.
default	Converts to INT64 from UINT32.
default	Converts to FLOAT from UINT32.
default	Converts to DFLOAT from UINT32.

from INT32

Conversion Name	Description
default	Converts to UINT8 from INT32.
default	Converts to INT8 from INT32.
default	Converts to UINT16 from INT32.
default	Converts to INT16 from INT32.
default	Converts to UINT32 from INT32.
default	Converts to UINT64 from INT32.
default	Converts to INT64 from INT32.
default	Converts to FLOAT from INT32.
default	Converts to DFLOAT from INT32.

from UINT64

Conversion Name	Description
default	Converts to UINT8 from UINT64.
default	Converts to INT8 from UINT64.
default	Converts to UINT16 from UINT64.
default	Converts to INT16 from UINT64.
default	Converts to UINT32 from UINT64.
default	Converts to INT32 from UINT64.
default	Converts to INT64 from UINT64.
default	Converts to FLOAT from UINT64.
default	Converts to DFLOAT from UINT64.

from INT64

Conversion Name	Description
default	Converts to UINT8 from INT64.
default	Converts to INT8 from INT64.
default	Converts to UINT16 from INT64.
default	Converts to INT16 from INT64.
default	Converts to UINT32 from INT64.
default	Converts to INT32 from INT64.
default	Converts to UINT64 from INT64.
default	Converts to FLOAT from INT64.
default	Converts to DFLOAT from INT64.

from SFLOAT

Conversion Name	Description
default	Converts to UINT8 from SFLOAT.
default	Converts to INT8 from SFLOAT.
default	Converts to UINT16 from SFLOAT.
default	Converts to INT16 from SFLOAT.
default	Converts to UINT32 from SFLOAT.
default	Converts to INT32 from SFLOAT.
default	Converts to UINT64 from SFLOAT.
default	Converts to INT64 from SFLOAT.

Note: conversions to integer types will truncate decimal values.

from DFLOAT

Conversion Name	Description
default	Converts to UINT8 from DFLOAT.
default	Converts to INT8 from DFLOAT.
default	Converts to UINT16 from DFLOAT.
default	Converts to INT16 from DFLOAT.
default	Converts to UINT32 from DFLOAT.
default	Converts to INT32 from DFLOAT.
default	Converts to UINT64 from DFLOAT.
default	Converts to INT64 from DFLOAT.

Note: conversions to integer types will truncate decimal values.

Appendix B – Troubleshooting Guide

This Troubleshooting Guide has been developed to provide RETL installation, development and operation suggestions for common support issues. The suggested resolutions are in response to product and documentation feedback from RETL users to the RETL product team.

When trying to run rfx I get this error: ksh: rfx: cannot execute.

RETL is not installed or configured properly. Try running verify_retl from the command line to get more information about the problem. Reinstalling RETL may also resolve the problem.

When trying to run rfx from the command line you get an error message like the following:

```
C++ runtime abort: terminate() called by the exception handling
mechanism
Abort(coredump)
```

Or

```
ld.so.1: rfx: fatal: libclntsh.so.9.0: open failed: No such file or
directory
```

```
Killed
```

This is also likely to be a configuration and/or setup problem. Run verify_retl from the command line to help diagnose the problem.

You probably don't have your library path setup properly. Here are several things you can check:

1 Are you using the correct library path variable? The environment variable to use for the library path depends upon the platform upon which you are running rfx - (see Installation and setup for the correct variable).

2 Is $RFX_HOME/lib in your library path?

3 Is the database library in your LD_LIBRARY path? (e.g. $ORACLE_HOME/lib32)

4 Are you linking against an old version of database and/or rfx libraries? (We suggest that you remove any unnecessary library paths and place the paths necessary to run rfx at the front of your library path to minimize this possibility.)

For example if you are using an ORACLE database you probably want to setup your environment like the following:

```
export ORACLE_HOME=/your/oracle/home/here
```

```
export LD_LIBRARY_PATH=$ORACLE_HOME/lib32:$LD_LIBRARY_PATH
export
LD_LIBRARY_PATH64=$ORACLE_HOME/lib:$LD_LIBRARY_PATH64
```

```
export PATH=$ORACLE_HOME/bin:$PATH
```

Are there any suggestions for how best to develop and debug flows?

Yes, there are several important steps you should take when developing and debugging to keep you on the right track.

1 Do not optimize until you have a working flow. Specifically, set numpartitions to 1. Do not use the HASH operator. When your initial development is complete then worry about tuning for performance.

2 When building new flows, start small and work your way up. Add another operator only after you've tested the smaller flow. Backup your work often so that you can look at the diffs between flows in order to narrow problems to smaller segments.

3 As you are coding, copy and paste from examples that you know work to avoid syntax/xml errors if you are using a text editor.

4 When you run into problems, try breaking up your flows. Insert debugs and/or exports to show that your flow is working to a certain point (think binary search here). Move these operators further down the flow until you figure out what is causing it to break. If you are running into core dumps or abnormal program termination, see "My flow is core dumping - what is wrong!?". A feature that has been added since version 10.3 is the ability of RFX to print schemafiles for each dataset connecting operators (by adding '-sSCHEMAFILE' to the command line options). This greatly aids when breaking up and debugging flows, by printing out each input/output schema in schemafile format. See the section RFX Command line options for more information about the '-sSCHEMAFILE' option.

5 As of release 10.3 you can look at the logfile to see when operators start, stop and how many records are flowing through them. Turn on more details by specifying a higher "level" in the LOGGER section of the configuration (see Configuration section for more details).

When running rfx I get the following error: Operator: db2write is not a registered operator.

RETL runs but does not recognize the name of the operator you've specified. If you are developing new flows, check your spelling. If you are using already existing flows with a new installation of rfx, and rfx is complaining about a database operator, then it is likely that you selected the wrong database type upon installation. The last possibility is that you are trying to run a new script with an older version of rfx that does not recognize a new operator type. Be sure that you are running the correct version of rfx for use with the flow.

When running rfx I get the following error: `Error Connecting to the database()`

You've probably typed an invalid database name, userid, or password. Verify that these parameters are correct (See steps 1-4 of Appendix B – Database Troubleshooting) and try again.

"My flow is core dumping - what is wrong!?"

There are several common things you can look for if this is happening.

1 If a schema file is specified in an EXPORT make sure that it matches the dataset exactly. Remember, columns can be removed from the output, but for the columns that are output be sure the schema file is correct. Specifically, look for mismatched datatypes. The best way to check to see if this is your problem is to remove the schemafile from the EXPORT. This allows the EXPORT operator to simply output the data using the existing schema. If your flow runs successfully after this change the schemafile is your problem. Try using the "-sSCHEMAFILE" option to help construct schemas.

2 Look for syntax errors – are you missing a closing tag (e.g. </OPERATOR> or </PROPERTY>), a quote? Did you misspell a tag? Did you put a "\>" by mistake instead of "/>" to terminate an element? Does your schemafile contain the correct syntax?

How do I tell what commands are being sent to the database?

Previous versions of rfx streamed database commands to the console by default. To see these commands starting from version 1.7,you should ensure that the environment variable "RFX_SHOW_SQL" is defined within your environment. Assuming that you are using ksh, you can do this by putting the following statement into your .profile or .kshrc, for example:

```
export RFX_SHOW_SQL=1
```

I got an error message about operator "sort:3". How do I figure out which operator "sort:3" is?

RFX names operators internally based upon the type and their position within the specified XML flow. Assuming that numpartitions is set to 1, sort:3 is the fourth sort operator in the XML flow (RFX starts counting at 0).

This becomes a bit more complex if you are using partitioning (numpartitions>1). In short, partitioning splits data and creates parallel data streams within the flow. In effect this multiplies the number of operators being used depending upon the position within the flow and other factors.

There are two ways to determine what operator RFX is referring to in this case. The easiest way is to set numpartitions to 1 and count the operators of that type within the flow. If you cannot do this, then run the flow again except specify the "--graphviz-files" option. This generatestwo ".dot" files which contain the flow before and after partitioning. In order to view these you need to download a copy of "dotty" which is included with the graphviz package available from AT&T Bell Labs: (http://www.research.att.com/sw/tools/graphviz/download.html).

I've hashed and sorted my data, but for some reason my output data is not sorted. What is wrong?

Your partitioning is probably set to a value greater than 1. Hashing your data allows RFX to break up your data and parallelize your flow into separate data streams. Later in the flow RFX will rejoin these separate data streams. By default RFX will do this by using the COLLECT or FUNNEL operator. However, these operators do not keep the dataset in sorted order, hence your unsorted output. You can correct this problem by explicitly using a SORTCOLLECT to rejoin the data whenever you are rejoining sorted data.

This problem is also avoided by not partitioning (set partitioning to 1 and/or don't specify a hash). As always, we recommend that you keep flows as simple as possible until or unless you need the performance boost that partitioning can give you.

I'm using sortcollect (or sortfunnel), but my data is not sorted! What is wrong?

Sort funnel performs a merge sort. It requires sorted input and can merge multiple inputs maintaining sorted order. It cannot perform a full sort of the data. You should use sort to do a full sort of a single data stream. Sortfunnel can be used to merge two data sources that are already sorted on the same keys – the result is a single data stream sorted on the same key.

I am missing fields within my flow. Where did they go?

Often the cause of this problem is that a schema file is incorrect (search for missing quotes, extra quotes) or the database schema is not what you expected. If your flow is not working correctly, take a look at the schemas that are output to stdout when RFX first starts up (specify the "-s" option if necessary to make RFX output the schemas). Look carefully at the schemas displayed for each operator within the flow. Make sure that these display ALL of the fields that you expect to see for this part of the flow. If you are missing fields or fields are of the wrong type, trace backwards to where the schema originates to try and find the source of the problem.

If you are using partitioning and are seeing operators that you do not expect, refer to I got an error message about operator "sort:3". How do I figure out which operator "sort:3" is? This section helps you understand exactly what operators RFX is seeing so that you can more easily find the source of your problem.

How do I filter values from two different tables?

Currently filter assumes that all values required for the filter are in one record. This makes it a bit more difficult to compare values from multiple tables. The way to handle this is to JOIN the two tables and then filter the values after the join.

How do I filter out a set of dynamic values from a data stream?

How do I filter out a set of values? For example – I want to remove any records whose "FIRST_NAME" column does not have a value in the set {Joe, Susan, Mary} and keep all the rest.

This is easily done with RFX using the LOOKUP operator. Simply use the filter set as the key for the lookup. Import a flat file and or table with a single column describing the values to filter – FIRST_NAME = {Joe, Susan, Mary} and use that as the input for the lookup table of the lookup. Any matches are discarded and any misses are processed.

So the actual lookup would look something like this:

```
<OPERATOR  type="lookup">
<PROPERTY  name="tablekeys"   value="FIRST_NAME"/>
<PROPERTY  name="ifnotfound"  value="continue"/>
<INPUT     name="records_to_filter.v"/>
<INPUT     name="set_names_to_drop.v"/>
<OUTPUT    name="records_in_set.v"/>
<OUTPUT    name="records_not_in_set.v"/>
</OPERATOR>
```

How do I translate from database types (e.g. NUMBER(12,4)) to RFX types?

Numbers work differently within RFX than they do within databases. This can cause a good deal of confusion. In short, RFX does not currently support arbitrary precision math and therefore has a limited amount of precision to deal with.

Specifically, here is the size and precision that each type has available:

RFX Type	Max Size	Max Precision	Validation/Numeric Range
DFLOAT	25	15	Min= 2.2250738585072014E-308 Max= 1.7976931348623157E+308
SFLOAT	25	6	Min=1.17549435E-38 Max=3.40282347e+38
INT8	4	4	Min=-128 Max=127
INT16	6	6	Min=-32768 Max=32767
INT32	11	11	Max=2147483647 Min=-2147483648
INT64	20	20	Min=-9223372036854775808 Max=9223372036854775807
UINT8	3	3	Min=0 Max=255

RFX Type	Max Size	Max Precision	Validation/Numeric Range
UINT16	5	5	Min=0 Max=65535
UINT32	10	10	Min=0 Max=4294967295U
UINT64	20	20	Min=0 Max=18446744073709551615
DATE	8	8	Is specified in MMDDYYYY where: MM=01-12 DD=01-31 YYYY=0000-9999
TIME	8	8	Is specified in HH24MISS format where: HH24=00-23 MI=00-59 SS=00-59
TIMESTAMP	16	16	Is specified in YYYYMMDDHH24MISS format where: MM=01-12 DD=01-31 YYYY=0000-9999 HH24=00-23 MI=00-59 SS=00-59

Note the “Max Precision” column above. For integral types the precision is roughly equivalent to the size (the caveat being the min and max values in the table); however for floating point types (sfloat and dfloat) the precision indicates how many total digits to the left of the decimal point (assuming that the float is written in scientific form) you can look at before rounding errors are encountered.

If you are determining what types to use from an imported file, we recommend that you use the biggest numeric type you can (i.e. int64, uint64 or dfloat) to ensure that data is properly preserved and the flow works for as many different sets of data as possible. If you know your data very well and are sure that you can use a small type, you may get a bit of a performance boost by making this a restriction. RFX warns you on the IMPORT if your data cannot be represented by the type that you’ve chosen.

Is there a common set of Properties for all database operators?

Originally, RFX was developed to try and stay consistent with each of the database vendor's terminology. This, however, made it very difficult to write code that was portable between the different databases and to understand how to write flows for different databases. Starting in release 10.2 the following common properties have been added to each of the database operators:

- userid
- password
- tablename

Over time we will attempt to merge the database operators so that RFX is database independent without the use of shell variables.

In my query, I am explicitly selecting the timestamp field as 'YYYYMMDDHH24MISS', but the data in the output file reads "invalid date". What am I doing wrong?

Currently RFX cannot read DATE, TIME and TIMESTAMP types directly out of the database. Because there is no conversion from string to these fields if you must access these fields as DATE, TIME and/or TIMESTAMP you will have to export your data to a file and then import it as a date field.

When attempting an orawrite I get the following error: "ORA-03106: fatal two-task communication protocol error".

Chances are you don't have the correct version of Oracle client installed to access the database that you are trying to access. For example, you might have the Oracle 9.0.1 client trying to access a remote 9.2 database. Verify that you've installed the correct level of client software and try again.

When attempting an orawrite I get the following error: SQL*Loader-951: Error calling once/load initialization

When you use ORAWRITE operator with the "method" property set to "direct", you may get the following error message in your SQL*Loader log file:

```
SQL*Loader-951: Error calling once/load initialization
ORA-26028: index name.name initially in unusable state
```

It is because SQL*Loader cannot maintain the index which is in IU state prior to the beginning of a direct path load. Fix this problem by recreating the table index.

[IBM][CLI Driver] SQL0290N Table space access is not allowed.

You are using a DB2WRITE but the table you are loading to is not setup for multiple nodes. See Direct mode loading with db2write to correct this problem.

[IBM][CLI Driver][DB2/SUN] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code "2". SQLSTATE=40001

This error can occur if you have a deadlock condition in your flows. Specifically, check for cases where the same table is accessed/locked for writing by multiple operators within a flow or a set of flows that are running concurrently.

In the case of deadlocks, you may want to create an event monitor for deadlocks to gather more information for diagnosis. You might also try increasing the number of "maxlocks" to avoid lock escalation, which can often result in a deadlock scenario.

[IBM][CLI Driver] SQL1224N A database agent could not be started to service a request, or was terminated as a result of a database system shutdown or a force command.

See DB2 on AIX to resolve this problem.

How do I request help when all else fails?

Contact support@retek..com. When requesting help with a flow follow these guidelines to get a more rapid response:

1. Minimize the scope of the problem. Submit the minimal flow with the minimal set of data and minimal schema that still reproduces the problem. This helps to ensure that the support team does not waste time trying to understand a long complicated flow and/or the data involved – allowing us to focus on the true problem.
2. If you have a database operator, (e.g. ORAREAD, ORAWRITE) separate the data from the database. Usually the database is not the problem; therefore removing the database operator from the equation makes our job easier. Replace the DB operator (e.g. ORAREAD, ORAWRITE) with an IMPORT or EXPORT operator. Use the exact same schema as is in the database. Extract a sample set of data into a flat file for the IMPORT operator. This should allow the support team to reproduce your problem here and debug it if necessary. This drastically reduces the amount of time it would take us to look into the problem otherwise.
3. If removing the database operator removes the problem then look very carefully to make sure that there is not a syntax error in the database operator. If you cannot find anything there, look at the IMPORT schema and make sure that it matches the database schema exactly (including null fields, etc). Finally, look at the flat file that the IMPORT operator is using. Make sure that you are using the correct delimiter, that the delimiter is not within the data and that the data is valid. Try setting the "rejectfile" property to see if there are any rejected records. If all this looks good then there may be a problem with the DB operator. Contact us and let us know what you've found.
4. Submit the flow and all data files associated with the flow so that the support team can run "rfx –f <yourflowname>" to reproduce the problem along with the log file from "verify_retl" so they can reproduce your environment.

I've upgraded and now I get WARNINGS when I never did before! What's wrong!?

"WARNING" messages are generated if the property names and / or values don't match the spelling / case noted in the Programmer's Guide. It is very important to run 10.3 and correct the warnings since they indicate that parameters are incorrect. If incorrect, they can produce unexpected results and/or corrupt data. In the future, RETL will stop processing and display error messages rather than just providing warnings to stop the flow until the problem is fixed. Warnings are being used as an intermediate step to minimize the impact to existing product

Appendix C – Database configuration and Troubleshooting Guide

Since RETL database operators work with database severs and utilities, validating RETL requires some background in database setup and administration. Specifically this involves database setup, RETL database login id privileges, and database connectivity, etc.

RETL Database configuration and maintenance notes

RETL uses very specific tools in order to access the different databases. RETL developers should be sure that their Database Administrator is aware of what RETL does, and how the database needs to be setup to support RETL activities. Below are notes particular to certain databases and platforms.

TeraData ODBC Drivers

Current versions of RETL require that the ODBC driver be located at /usr/odbc/drivers/tdata.so. Additionally /usr/odbc/lib is included in your library path.

DB2 on AIX

For users on AIX version 4.2.1 or newer, the environment variable EXTSHM can be set to ON to increase the number of shared memory segments to which a single process can be attached.

AIX and shared memory on DB2 32bit EXTSHM is an environment variable recognized by versions 4.2.1 and later of the AIX operating system. EXTSHM applies only to 32-bit processes. It affects how shared memory segments work, and affects I/O semantics, as well. DB2 makes extensive use of shared memory segments. On 32-bit AIX, there are only eleven shared memory segments available for each process.

The documentation for AIX 4.3 states "if an environment variable EXTSHM=ON is defined then processes executing in that environment will be able to create and attach more than eleven shared memory segments."

Attaching to more than eleven shared memory segments is attractive to DB2 users.

This error message is a symptom of this problem:

sqlstate[08001], sqlcode [-1224]: [IBM][CLI Driver] SQL1224N A database agent could not be started to service a request, or was terminated as a result of a database system shutdown or a force command. SQLSTATE=55032

The workaround solution is to implement the following steps.

1 Set the DB2 registry

```
db2set DB2ENVLIST=EXTSHM
```

2 Update the sqllib/db2profile file adding:

```
EXTSHM=ON

export EXTSHM
```

3 Stop DB2, log out of UNIX (or re-execute the db2profile file) to pick up the environment changes and restart DB2.

For more information how the EXTSHM environment variable affects DB2 shared memory on AIX, see the documentation on IBM's APAR IY09243

Direct mode loading with db2write

Direct mode loading with db2write works only with the tablespace setup for multiple nodes. You need the LOAD privilege and also the tablespace that you are using should be created for multiple nodes, or else you can't use the direct load mode in DB2. Only one db2write with direct loading works for a given tablespace at a time. Since some flows attempt to do multiple db2writes simultaneously, this problem may occur while using rfx with an improperly configured database.

If you see “"SQL0290N Table space access is not allowed" you need contact to your DB2 DBA to unlock the tablespace you use and make sure the tablespace that you are using is created for multiple nodes

Avoiding deadlock while using DBWRITE operators.

A deadlock may occur if a flow attempts to do a DBWRITE at the same time as other DBWRITE or DBREAD operations to the same table are being executed within the same flow.

Debugging database ETL utilities

The DBWRITE operators use the database specific import/export utilities. If you run into problems specific to the database and/or database operators, take a look at the log files and/or control files generated by the database operators (see How do I tell what commands are being sent to the database?). You may be able to better diagnose these problems by checking the utility's limitations, restrictions, environment, and privileges to see if there is a misconfiguration either with the database or within the rfx flow.

Database semaphore problems

Semaphores from database utilities and ODBC instances invoked by rfx may not be properly cleaned up if an rfx process or job is killed from Unix. In order find and clean up semaphores use “ipcs –s” and “ipcrm –s” respectively.

Runaway loader processes

Sometimes when rfx is killed some ancillary database utilities are left running. These can be killed via the normal Unix kill command.

For Oracle the executable is called “sqlldr”.

For DB2 the executable is called “db2atldr”.

TeraData has several executables: “mload”, “fastload”, and “fexp”.

Troubleshooting RETL with your database

The following RETL/ database setup validation steps will help you work with your DBA and System Administrator to verify and troubleshoot the RETL/database installation.

Note below that we include commands that should be executed from the command line. These are written in `Courier` font. Additionally, variables such as machine names and IP addresses that should be replaced by values particular to your environment are surrounded in <> like this: `<variable name>`.

1 Run verify_retl.

   If you have not already done so run the verify_retl script and correct any problems as directed by the script. When this is running properly proceed to step 2.

2 Verify that your databases are setup properly.

   For all databases go through RETL Database Configuration Notes to make sure that the database is setup properly.

3 Verify database connectivity

   Check connection between rfx local machine to remote database machine if the database and rfx are not in the same machine. We recommend that RETL is located on the database machine for better performance for Oracle and DB2 – Teradata is always located on a remote machine.

   ```
   ping <database machine name>
   ```

   If this fails, try pinging the IP address directly:

   ```
   ping <database machine IP address>
   ```

   If this works then your DNS isn't set up properly. Contact your Network Administrator to fix the problem by adding the IP address to the DNS.

   If this second ping command fails then you need to contact your Network Administrator to determine why you cannot contact the machine. Until this problem is resolved RETL will not work.

4 Check database server connectivity.

   For Oracle:

   ```
   tnsping <Oracle Server Name>
   ```

   If this fails contact to your DBA for TNS name setup.

5 Verify database login id/password.

Login the database by using the appropriate database utilities.

For Oracle:

```
sqlplus userid/password@Oracle_database_server
```

For DB2:

```
db2 connect to database_name user id using password
```

For Teradata:

```
Bteq database_server_name
logon
user id
password
```

Contact to your DBA if this step fails – your userid/password is not setup correctly.

6 Check database user id privileges.

Contact your DBA to ensure that the RETL user has privileges to: select, insert, update, create table, etc. Since RETL database operators invoke database utilities to extract and load (see Chapter 5 Database Operators for details), your RETL DB user needs privileges to meet all of the utilities requirements.

These are the privileges that each of the operators require:

Operators	Privilege
ORAREAD, DB2READ, TERAFEREAD	select
ORAWRITE, DB2WRITE, TERAWRITE	select, insert, update, create table, drop table, load

We recommend setting up the RETL/database login id with create/drop table privileges during RETL/database setup phase so that we can run the RETL/database test flows to verify that the database is setup properly for RETL.

7 Run RETL/database testing scripts.

Note these scripts require that the retl userid has full privileges to run. If you don't have the appropriate privileges to change tables you can still run the read scripts to verify that you can read from the dabase without problems go to step 10.

For Oracle:

```
$RFX_HOME/samples/oracle.ksh <database name> <userid>
<password>
```

For DB2:

```
$RFX_HOME/samples/db2.ksh <database name> <userid>
<password>
```

For Teradata:

```
$RFX_HOME/samples/tera.ksh <database name> <userid>
<password>
```

These scripts will:

- Create a "RETL_test" table with two columns and two rows on your database owned by the given userid.
- Read the two rows from the database and verify that the rows contain the correct information.
- Compare the results from read and verify what was supposed to be written.

In short, this test verifies both RETL read operator and RETL write operator appropriate to the database type.

If the scripts are working properly you see something like the following:

```
Testing a write to etlsun9i
Reading data back out of etlsun9i
Comparing data received from etlsun9i and expected
output...
Test Passed!
```

If both tests succeed - Congratulations your database seems to be properly configured! Skip the following steps.

If the scripts are not working properly you will see the following:

```
Testing a write to etlsun9i
RFX Unexpected Error: Operator.cpp:385: 'Operator: orawrite
is not a registered operator'
Reading data back out of etlsun9i
RFX Unexpected Error: Operator.cpp:385: 'Operator: oraread
is not a registered operator'
Comparing data received from etlsun9i and expected
output...
diff: oraread.out: No such file or directory
Changes were detected! Test failed!
```

If either test fails you need to move onto the following steps to try and diagnose the problem. If the DB write operator passed but the read operator did not, move onto step 9. If both failed then move onto step 8.

8 Debugging a DBWRITE failure.

This step verifies that the database write operator is working properly.

For Oracle:

```
orawrite.ksh <database_name> <userid> <password>
```

For DB2:

```
db2write.ksh <database_name> <userid> <password>
```

For Teradata:

terawrite.ksh <database_name> <userid> <password>

This flow works if the last line of output shows "Flow ran successfully". Go to step 9.

Here are some common failures and their causes:

'Operator: db2write is not a registered operator.'

RETL does not recognize the database operator (see When you try to run rfx you get an error like: 'Operator: db2write is not a registered operator.'). First double-check that you are using the correct shell script for this test – this error will happen if you use the wrong script for your binary. If you are using the correct script this error means that you've installed the wrong RETL for the database you are trying to access – this version of RETL does not support the operator necessary for accessing the database you are trying to read from. You need to reinstall RETL selecting the appropriate database for your environment.

Note: RETL binaries currently only support 1 database type each. Multiple binaries would have to be installed in order to access different types of databases.

```
'Error Connecting to the database()'
```

You've probably typed an invalid database name, userid, or password (see When running rfx I get the following error: 'Error Connecting to the database()'). Verify that these parameters are correct (See step 4 above) and try again.

If you still cannot find the problems you can try running step 10 on a table that you know already exists.

9 Debugging a DBREAD failure.

This step will verify that the database read operator is working properly.

For Oracle:

```
oraread.ksh <database name> <userid> <password>
```

For DB2

```
db2read.ksh <database name> <userid> <password>
```

For Teradata

```
teraferead.ksh <database name> <userid> <password>
```

If the last line of output shows "Flow ran successfully" your RETL/database setup is working for reads – Congratulations! You can run step 10 if you'd like to try extracting data from your own tables.

If it fails there are several possible reasons (generally the same as those for the write operator) – here are some common errors:

```
'Operator: db2read is not a registered operator.'
```

RETL does not recognize the database operator (see When you try to run rfx you get an error like: 'Operator: db2write is not a registered operator.'). First double-check that you are using the correct shell script for this test. This error happens if you use the wrong script for your binary. If you are using the correct script this error means that you've installed the wrong RETL for the database you are trying to access. This version of RETL does not support the operator necessary for accessing the database you are trying to read from. You need to reinstall RETL selecting the appropriate database for your environment.

Note: RETL binaries currently only support 1 database type each. Multiple binaries would have to be installed in order to access different types of databases.

```
'Error Connecting to the database()'
```

You've probably typed an invalid database name, userid, or password (see When running rfx I get the following error: 'Error Connecting to the database()'). Verify that these parameters are correct (See step 4 above) and try again.

If you are still running into problems and there are valid tables that you know work, try step 10. Otherwise review the RETL Database Configuration Notes, to verify that everything is set up correctly and try again.

10 Running the DBREAD script against user tables.

The database read scripts could be used to access any table to which you have read privileges by simply specifying the table name as the last parameter.

For Oracle:

```
oraread.ksh <database> <userid> <password> <table>
```

For DB2:

```
db2read.ksh <database> <userid> <password> <table>
```

For Teradata:

```
teraferead.ksh <database> <userid> <password> <table>
```

If the last line of output shows "Flow ran successfully" your RETL/database setup is working for reads on this table.

If you still cannot find out the problems, contact to Retek Support (support@retek.com).

Appendix D – FAQs

Does RETL call stored procedures?

As of version 10.2, RETL can call stored procedures only within the ORAWRITE operator via the sp_prequery and sp_postquery, which run before and after the ORAWRITE operation respectively. These stored procedures can take static variables (not data from the flow) as parameters. We anticipate expanding on this feature in the future to allow lookups based upon flow data and support for other database operators.

Can RETL handle text translations (e.g. between EBSDIC and ASCII)?

Currently RETL does not do textual translation of this type.

Can RETL handle data translations (e.g. between numbers and strings, etc)?

Data translations between different fields can be done via the LOOKUP and JOIN operators.

Are clients able to obtain RETL performance metrics on RETL?

Generally clients are interested in how a particular flow that use RETL runs – rather than how RETL itself performs at a fundamental level. These are generally handled on a case-by-case basis for the different product groups since flows are very different from product to product and the environment differs depending hardware and configuration. Contact your product group to determine if they have benchmarking number specific to your group.

Is it possible to do joins from 2 different databases?

Yes. This is generally done via two DBREAD operators and then a JOIN operator.

What error handling capabilities does RETL have?

There are many other areas where RETL does error handling and this capability has been expanding especially over the last several releases (since 1.7.0).

Error Handling – flow related problems. These are problems that an RETL flow developer might run into while developing flows using RETL. There are many different areas that RETL detects – because these are generally found only during development RETL usually exits with an error message describing one of the scenarios below:

- Schemas: mismatched input schemas (e.g. for FUNNEL), invalid schema specified (e.g. record length does not equal sum of field lengths).
- Mismatched datasets – RETL enforces 1-to-1 correspondence between inputs and outputs.
- Validation of operators, properties and property values.

Error Handling – data related problems. These are found after the flow is parsed and execution begins. In these cases invalid or non-conforming data may be siphoned into a separate file for later processing:

- Improperly formatted fields via the IMPORT operator (invalid types, lengths, etc).
- Improperly formatted records via the IMPORT operator (missing fields, etc).
- Invalid schemas.

Additionally, flow developers can insert data validation to verify that various database constraints are upheld via a combination of LOOKUP, JOIN and FILTER operators.

What temp files does RETL create and how much space will they take up?

RETL generates temporary files dynamically in order to sort and store intermediate data. These files are generally prefixed with "rfx", but this is not always the case. Therefore, separate rfx files into it's own tmp (e.g. /tmp/rfx) directory so that they can be easily purged on a regular basis (as per the Programmer's Guide).

How much space rfx will consume depends upon the complexity of the flows and how large the source system is. A good starting point is twice the size of the data being manipulated. A more conservative number is 3 times the size of the data being moved. Many people start with 10GB and move up from there if needed.