IRI CoSort
Sort, Transform & Report

V10.5 Product Overview

Technical Summary, Samples, and Specifications

IRI
Total Data Management

Table of Contents

An Introduction to CoSort

Since 1978, CoSort has been meeting the growing data manipulation needs of companies with high-volume flat-file, database and data warehouse installations. CoSort is also a favored solution for legacy sort and data migrations to Unix and Windows.

IRI has worked to make CoSort the most widely licensed commercial sort product on open systems, and is heavily focused on the development of related data manipulation technologies.

CoSort is now a performance-enhancing solution for many applications, the basis for IRI's structured data processing product line, and serves a single-pass platform for large-scale:

Data Processing	Transformation, Migration, Cleansing, etc.
Data Presentation	Data Wrangling and Reporting
Data Protection	Data Masking, Encryption, etc.
Data Prototyping	Test Data Generation

Data Processing

The CoSort Sort Control Language (SortCL) program can execute parallel data transformations to integrate, stage, and convert large data volumes. In just one I/O pass and job script, SortCL can:

> *select, sort/merge, join, lookup, convert data types and endian, file formats, re-map/reformat, pivot, sequence, calculate, aggregate, manage sub-strings, scrub, encrypt, de-identify, and perform complex transforms*

Sources and targets include compressed, flat and index files, pipes, tables via Open Database Connectivity (ODBC), and custom procedures.

Data Presentation

SortCL users can output the results of the above processes into one or more detail and summary reports. Users can combine joins, cross-calculations, hash lookups, and conditional selection to generate formatted reports and subsets for: billing operations, customer segmentation, change data capture, forensic data analysis, and business intelligence tools. Formatting may include special field and file layouts, headers and footers, page numbers, environment values, embedded HTML tags (for web posting), and the conversion of data into CSV or XML for hand-offs to BI and analytic tools.

Data Protection

SortCL, and the spin-off data masking product IRI FieldShield, can secure sensitive data at the field level, based on business rules. Functions include 256-bit AES format-preserving encryption and de-identification, and data masking techniques to anonymize, obfuscate, pseudonymize, or redact fields. Additional encryption or security functions are also available through custom field transforms. Securing data at the field level (during or after processing and presentation) is faster, and leaves non-sensitive file, disk and database data available.

Data Prototyping

SortCL, and the spin-off test data product IRI RowGen, can randomly create or select test field data and display it in real (production) file and report formats. You can create any number, type, and size of files, records and value ranges necessary to safely simulate reality and stress-test applications. Uses include database and ETL tool population, benchmarking, application development, and outsourcing.

Figure 1 *| CoSort Single-Pass Data Manipulation Flow Diagram*

CoSort Users & Uses

CoSort is a general-purpose, high-performance processor of sequential data in a variety of formats. It also serves as a migration platform for legacy data and sort programs, while also supporting business intelligence, ETL, and data governance operations. CoSort reduces runtimes, risks, and complexity for a variety of IT and business users.

Job Function	CoSort Deliverables and Benefits
IT Manager	• Legacy sort software migration and modernization tools • Universal flat-file format, and data type, conversion capabilities • Detail and summary batch reporting with optional dashboard • Codes and runs faster than Perl, shell, SQL, ETL, and COBOL jobs • Affordable price points and flexible licensing policies
DBA	• Parallel pre-sorts improve load, reorg, and query performance • Combined sort, join, and aggregate transformations offline • External batch and delta reports that are faster/easier than SQL • Flat-file lookups offer discrete, offline, one-to-many solutions • Shared metadata with Fast Extract (FACT), FieldShield, and RowGen
Data Warehouse (ETL) Designer	• Plug-in sort accelerators for DataStage and Informatica • Multi-threaded transformations in the file system for data staging • Complex selection and expression logic for data integration • Easy, open metadata and converters interface with existing tools • Integrated protection, custom transforms, and in/out procedures
BI Architect	• Fast aggregation and filtering wrangles (big) data for BI tools • Embedded reporting with many formatting functions, including PCRE • Field-level protections enable compliant segmentation reports • Web log and IPA data handling facilitate clickstream analysis • Change data capture (delta) reporting using joins and selection
CISO, Data Governance or Compliance Officer	• Field-level anonymization, de-ID, encryption, pseudonymization • Protection functions can run within transform and reporting jobs • Query-ready XML audit log of job details help verify compliance • Quality and safety improvements for Master Data held in flat files • Support for many COBIT control objectives
Application Developer (ISV)	• Thread-safe API libraries for embedded parallel sorts, transforms • Serial and parallel system calls to the SortCL program • Access to included encryption libraries protect real-time data flows • Built-in test data generation capabilities (RowGen functionality) • Affordable licensing, customized to individual business models

Compatible Applications

In addition to CoSort, many other IRI products create or leverage the metadata of the CoSort SortCL language to define the layouts and manipulation of data in the various fit-for-purpose contexts described below.

Note that CoSort, and all the listed products, also include and share the same free graphical integrated development environment (IDE) for (SortCL-based) job design and scheduled deployment called **IRI Workbench**™.

Built on Eclipse™, IRI Workbench facilitates the specification, execution, tuning, and maintenance of SortCL job scripts through job creation and metadata definitions wizards, workflow and transform mapping diagrams, and a syntax-aware editor with dialog-supporting outline for manual specification.

IRI Workbench also provides database access, viewing, and integration with SortCL and other jobs. It includes extensions for team contributions, job version control, and remote system data and execution.

IRI Voracity® Data Management	Consolidates the discovery, integration (ETL, CDC, SCD), migration, governance, and analytics of data in big and small structured data sources on-premise or in the cloud. A Hadoop edition of Voracity can also seamlessly convert and interchangeably run many CoSort SortCL jobs in MapReduce 2, Spark, Spark Stream, Storm or Tez. Voracity includes wizards for CoSort and all the products below..
IRI FACT™ Fast Extract	Unloads very large database (VLDB) tables to flat files in parallel for off-line archival, data transformation, reporting, migration, and reloads. FACT can work through metadata and pipes with the CoSort SortCL program to perform fast reorg, replication, encryption, ETL, and BI operations, all in one I/O pass.
IRI FieldShield® Data Masking	De-identifies personally identifiable information (PII) and other sensitive data in ODBC-connected database tables or popular file formats with multiple functions, including: encryption, blurring, hashing, pseudonymization, randomization, and tokenization.
IRI RowGen® Test Data	Uses SortCL syntax to create safe test data in real file, report, and table formats for prototypes, DevOps, benchmarking, etc.
IRI NextForm® Data Migration	Converts database tables, file formats, field data, and endian types for data, database, application, and platform migration projects.

Following are examples of third-party products with which IRI maintains various levels of compatibility to enhance their operational performance or data protection capabilities via CoSort SortCL functionality:

Hadoop	CoSort can read and write files to/from the Hadoop File System (HDFS) via a SortCL /INFILE URL specification. Many SortCL jobs can also run in Hadoop through the Voracity gateway for Hadoop.
Oracle	CoSort can source and target Oracle tables via ODBC or OCI to the file system (using FACT and SQL*Loader). On this data, CoSort can: ● Transform, cleanse, and integrate Oracle and other data ● Capture changed data (generate delta reports) ● Mask (column-level encryption, redaction, etc.) ● Perform index load pre-sorts (on the longest table key) By pre-sorting, CoSort can improve the speed and efficiency of SQL* Loader (and other RDB load utility) operations, and thus reorgs and queries on pre-CoSorted tables. CoSort and related jobs can also be triggered or invoked by Oracle Job Scheduler.
IBM DB2	Source and target DB2 tables via ODBC or the file system (using FACT and DB2 Load), to do the jobs above. Also, the CoSort Load Accelerator for DB2 (CLA4DB2) speeds bulk loads up to UDB Version 9.5.
MS SQL Server	Performs many of the same functions as SQL in SortCL, up to 10x faster.
MongoDB	Use SortCL to transform, migrate, mask or prototype MongoDB collections.
IBM InfoSphere DataStage	Run the CoSort SortCL program in a DataStage sequential file stage or as a before-job subroutine for larger sort, join, and aggregation jobs. SortCL transforms data in a one fast external pass which requires no partitioning or memory juggling, and can also include data cleansing, masking, reporting, etc.
Informatica PowerCenter	Push the largest data transformations down to CoSort SortCL programs in the file system as workflow command tasks. This approach is more efficient than Informatica-recommended “pushdown optimization” into Oracle, and far less expensive than Teradata, Ab Initio, DMExpress, etc.
Pentaho (PDI)	Similar to the above, calling CoSort into Kettle speed sorts jobs dramatically.
KNIME	Wrangle in, and display data from, CoSort SorCL jobs in IRI Workbench. Te. CoSort users with the Voracity Data (Job Source) Provider Node for KNIME can feed data mining, machine learning, and other data science nodes with CoSort-wrangled data in the same workflow.
Splunk	Index and analyze data in Splunk immediately with data prepared and protected by SortCL via the IRI add-on for Splunk, the Voracity app for Splunk, or through Splunk Universal Forwarder.
Proxy Coupling	Seamlessly offload z/OS JCL sort steps to SortCL on z/Linux or another ‘open system’ platform via Proximal Systems “PSCsort” technology.

CoSort Package Contents

The **CoSort** package contains standalone utilities, file layout metadata and sort parameter converters, third-party sort replacements, API libraries, and documentation. The core **utility programs** are:

- **SortCL** - the fourth-generation *Sort Control Language* program for defining data and manipulations with syntax and semantics familiar to both mainframe sort and SQL users. The most comprehensive interface in the CoSort package, SortCL combines multiple data transformation functions (sorting, joining, aggregation, filtering, remapping, etc.) with cleansing, reporting and masking for: file compare and change data capture, data type and file format conversions, data warehouse integration and staging (ETL), BI reporting and analytic data wrangling, delta and summary targets, plus data and database migration, replication, federation, and compliance with data privacy laws.
- **Sort** - a drop-in replacement for the Unix *sort* command that runs faster and scales linearly. It runs on all Unix and Windows (unixsort.exe) platforms.

SortCL recognizes environment variables and supports pipes and brokered data streams to allow data to flow between processes without additional I/O. SortCL may also be customized with user exit procedures for special input, output, or comparison criteria. SortCL job creation, modification, sharing and execution is supported in IRI Workbench.

These **metadata converters** leverage existing data source layout information:

- **cob2ddf** translates COBOL copybook layouts to SortCL data definitions
- **csv2ddf** translates Microsoft **.csv**. File headers to SortCL data definitions
- **ctl2ddf** translates Oracle SQL*Loader control file layouts to SortCL data definitions
- **elf2ddf** translates web logs in W3C " Extended Log Format" to SortCL data definitions
- **ldif2ddf** translates LDIF layouts to SortCL data definitions
- **xml2ddf** translates XML formats to SortCL data definitions
- **xls2ddf** translates XLS/X spreadsheet column headers to SortCL data definitions
- **odbc2ddf** converts database table layouts into a SortCL data definition file (.ddf)

The above utilities are also supported graphically in IRI Workbench, which also produces DDF for JSON and MongoDB sources (command line utility development is pending). In addition:

- **MIMB** (from MITI) translates many application file layouts to SortCL data definitions
- The **DataSwitch** no-code platform or **Mapping Manager** (from erwin) create SortCL data and job metadata from scratch, or convert to it from third-party ETL tool mappings.

These sort parameter conversion utilities facilitate legacy sort migrations:

- **mvs2scl** translates MVS JCL sort cards to SortCL job specifications
- **sorti2scl** translates SortI parameters to SortCL job specifications
- **vse2scl** translates VSE JCL sort cards to SortCL jobs specifications

The following third-party **sort replacements** are available with the CoSort package:

- **acu-cosort** - a drop-in replacement for the sort verbs supplied with ACUCOBOL-GT
- **cla4db2** - the " CoSort Load Accelerator for DB2" replaces IBM's sort routine within the UDB 5-8 loader, as much as doubling throughput on Unix
- **mf-cosort** - a drop-in replacement for the sort verbs supplied with Micro Focus Net Express, Server Express, and Workbench on Unix and Windows. COBOL users can link statically and dynamically to mfcosort and the CoSort engine to accelerate sort speed and reduce temporary sort space in new executables or a full RTS
- **nat-sort** - a drop-in replacement for the sort verb in Software AG natural
- **proc-sort** - SAS System 7-9 users can link dynamically to shared cosort() libraries to replace the sort function in SAS on Unix systems

CoSort provides similar drop-in sort replacement facilities for Tetrad OPX and the UniKix Mainframe Batch Manager. CoSort hooks are also available for the ETI Solution, Cincom Control:Manufacturing, Kalido's Dynamic Information Warehouse, and DataStreams (Korea) TeraStream ETL suite.

For developer and Independent Software Vendor (ISV) use, CoSort also includes two Application Programming Interface (API) libraries:

- **cosort_r()** - a thread-safe version of the original cosort() API that allows multiple coroutine sort/merge operations to occur in the same pass through the data. The coroutine engine allows in-memory record transfers between programs and the sort.
- **sortcl-routine()** - the thread-safe SortCL library that allows programmers to exploit the full range of CoSort Sort Control Language commands within their own programs.

CoSort APIs let you define any input (selection), compare (order sequence), or output (reporting) criteria, enabling applications to accomplish complex jobs in one I/O pass. You can write calls to either library in any language that can link to a C library, such as C++, COBOL, VB, Java, etc.

Finally, the CoSort package also includes the following **documentation**:

- **Installation Guide** - platform-specific loading, licensing, and configuration advice
- **Manual** - a full user and programmer documentation for all the above interfaces
- **Job Examples** - sample SortCL job scripts, metadata conversions, and API calls
- **Best Practices** - tips and tricks for job scripting and tuning

IRI Workbench also provides online material for CoSort users as follows:

- **Platform Overview** - a basic primer in the Welcome section for orientation purposes
- **Reference** - a set of hyperlinking content available under "Tools User Guides"
- **Cheat Sheet** - an interactive step-by-step 'Getting Started' wizard built on Eclipse
- **Dialog Help** - context-specific instructions for each job parameter on every page

Sort Control Language Program (SortCL)

Well beyond traditional sort/merge operations, **CoSort** provides a broad range of big data manipulation and management functions for data warehousing, legacy migration, and business intelligence projects through its fourth-generation Sort Control Language (SortCL) and program.

Functionality

SortCL allows end-users and developers to perform the following:

Function	Actions
Filter	At the byte, field, and record level, plus duplicate removal and saving
Segment	Conditional (include/omit) selection with if-then-else, else-if logic
Sort	Multiple keys, directions, sequences
Merge	Two or more pre-sorted files
Join (Match)	Two or more un/sorted files on many conditions for file compares and change data captures (deltas)
Aggregate	Parallel roll-up and drill-down sum, min, max, average, and count values. Accumulate (Running). Rank. Lead and Lag (Windowing).
Check	Verify source data is pre-sorted prior to sort or join operations
Re-Map	Resize, reposition, and realign fields
Convert	Change data types, file formats, endian states, and database (vendor/version) tables
Re-Format	Convert between file formats (e.g., Text <> XML, VS <> RS, ISAM <> Vision, LDIF <> CSV
Pivot / Unpivot	De-normalize and normalize dimensional layouts
Cleanse	De-duplicate, validate, homogenize, filter, find/replace, and re-structure
Enrich	Integrate and segment data enhance row and column detail. Create new data forms and layouts through conversions, calculations and expressions, and composites (templates).
Mask	De-identify PII at the field level via encryption, pseudonymization, redaction, etc.
Calculate	Math and trig functions across detail and summary rows

Function	Actions

Sub-string	Bit-level manipulations and PCRE for pattern matching and replacement, etc.
Validate	Verify characters and fields match their specifications (i.e. “isascii”, gap analysis)
Sequence	For custom indexing, reporting, and database load operations
Set Lookup	Discrete field substitutions, pseudonymization, etc. using “ set” file field dimensions
Fuzzy Search	For slowly changing dimension (SCD) reporting
Federate	Get discrete (lookup) values and virtualize results via reports and replicas
Test	Create randomly-generated or set-selected (safe) test data fields
Report	Custom-formatted, segmented detail, and summary targets
Replicate	Copy, manipulation, and move data from one or more sources to one or more targets
Custom	Complex field-level user functions (e.g., 3rd-party DQ libraries)

SortCL can support the combination of these functions all in one job script and I/O pass through multiple data sources and targets.[1] By running multiple data manipulations at once, SortCL helps you:

- Package, protect, and provision big data without Hadoop
- Filter, integrate, and stage data for DW, ODS, BI, data marts, and spreadmarts
- Replace slower 3GL, shell, Perl, and SQL procedures
- Transform high volumes outside BI, DB, and ETL tools
- Relieve applications and system overhead
- Generate custom detail, delta, and summary reports
- Accelerate bulk database reorgs and loads
- Detect, capture, and audit changed data
- Consolidate data privacy with transformation and reporting

SortCL uses a self-documenting Data Definition Language (DDL) and Data Manipulation Language (DML) syntax that is familiar to both mainframe sort and SQL users. SortCL DDL repositories or data definition files (DDF) can be centralized and re-used. By supporting the separation of data definition and manipulation statements, SortCL supports the use of shared metadata and the independence of data from applications.

[1] With the IRI Fast Extract (FACT) utility, bulk flat-file input can come from RDBMS table unloads (instead of ODBC).

Invocation

SortCL job scripts can be invoked from:

- The command line
- IRI Workbench
- A batch process
- Any job scheduler / workflow automation tool
- A thread-safe SortCL API (sortcl_routine) call

Application-level statistics can be output with each job, either to the screen or a file. In addition, the CoSort job log runs in a self-appending file and sends debugging information into a self-replacing file. On-screen monitoring options are available at various verbosity levels for runtime progress assessment. You can also enable and secure an XML audit log file to validate compliance and performance forensic application and data analysis.

Users converting from legacy sort products can leverage included metadata conversion tools and available IRI services to ease job script migrations to SortCL. See METADATA CONVERSION TOOLS.

Operation

One of the most basic SortCL scripts that you can write contains only an infile and an outfile, as shown in the following:

```
/infile=accts695
/outfile=accts695.new
```

This simply sorts the accounts695 file from left to right without reformatting.

SortCL processes data in three phases: input, action (processing), and output. In the input phase, source records are processed with selection. Actions are sort, merge, join, report, or check. In the output phase, selected records are remapped to one or more targets simultaneously. Derived fields, aggregates, additional filters, and multiple formats can be conditionally defined in the same target. A special “inrec” section is defined when a virtual record layout is needed for processing input sources that are formatted differently.

SortCL also has the ability to perform and combine many more data transformations, as well as protect data at risk and produce formatted reports, all at the same time. Through the use of metadata repositories -- SortCL DDFs -- you can define and share any structured data subset or relational view in the SortCL job specification files (.SCL) that reference the DDFs. Those file layouts can be re-used in many applications.

Application Sample #1

Sort and Reformat, Metadata Repository

This SortCL job script is a simple, two-key sort job. A single, flat input file is specified directly in the script. In addition to re-ordering the data, this script will convert the file layout from a pipe-delimited format to fixed position fields. Notice, however, that the fixed output field definitions are stored in this reusable SortCL "data definition file" metadata repository:

Data Definition File *chiefs.ddf*

```
/FIELD=(president,POS=1,SIZE=22)
/FIELD=(service,POS=25,SIZE=9)
/FIELD=(state,POS=40,SIZE=2)
/FIELD=(party,POS=45,SIZE=3)
```

Input File 1 *chiefs_10_sep*

```
Eisenhower, Dwight D.|134|1953-1961|REP|TX
Kennedy, John F.|135|1961-1963|DEM|MA
Johnson, Lyndon B.|136|1963-1969|DEM|TX
Nixon, Richard M.|137|1969-1973|REP|CA
Ford, Gerald R.|138|1973-1977|REP|NE
Carter, James E.|139|1977-1981|DEM|GA
Reagan, Ronald W.|140|1981-1989|REP|IL
Bush, George H.W.|141|1989-1993|REP|TX
Clinton, William J.|142|1993-2001|DEM|AR
Bush, George W.|143|2001-2009|REP|TX
```

The job script to the right uses explicit layouts for the input, but relies on the metadata file, ***chiefs.ddf***, for the output layout. By centralizing the metadata, it can be used in other SortCL job scripts.

```
### CoSort SortCL Job Specification ###

### Input Phase ###
/INFILE=chiefs_10_sep
       /FIELD=(president,POSITION=1,SEPARATOR='|')
       /FIELD=(votes,POSITION=2,SEPARATOR='|')
       /FIELD=(service,POSITION=3,SEPARATOR='|')
       /FIELD=(party,POSITION=4,SEPARATOR='|')
       /FIELD=(state,POSITION=5,SEPARATOR='|')


### Action Phase ###
/SORT
       /KEY=party
       /KEY=president

### Output Phase ###
/OUTFILE=chiefs.out
       /SPECIFICATION=chiefs.ddf    # metadata
```

SortCL job scripts are typically run from a batch script with a command entry similar to:

```
$COSORT_HOME/bin/sortcl  /SPECIFICATION=/path2/example1.scl
```

Output File 1 *chiefs.out*

```
Carter, James E.          1977-1981        GA     DEM
Clinton, William J.       1993-2001        AR     DEM
Johnson, Lyndon B.        1963-1969        TX     DEM
Kennedy, John F.          1961-1963        MA     DEM
Bush, George H.W.         1989-1993        TX     REP
Bush, George W.           2001-2009        TX     REP
Eisenhower, Dwight D.     1953-1961        TX     REP
Ford, Gerald R.           1973-1977        NE     REP
Nixon, Richard M.         1969-1973        CA     REP
Reagan, Ronald W.         1981-1989        IL     REP
```

The input file, **chiefs_10_sep**, is now in order by party and president and displayed according to the fixed position layout specified in **chiefs.ddf**. Notice that the second input field, *votes*, was not in the output file specification, and that the state and party fields were transposed. By mapping using symbolic field name references, SortCL gives you field-level control of all your output targets, as the following examples will further demonstrate.

Application Sample #2

Data Transformation and Masking

This SortCL job is an example of a single-key sort. The fields are defined in the input phase, the sort key in the action phase, and then, in the output phase, a series of target files are defined in different formats for different departmental purposes. Notice how individual fields are protected according to different business rules or role based access controls (RBAC).

The input file below was generated with IRI's RowGen tool to create realistic transaction data. Note that any number of sources can be input and that these input sources can have any number of formats. The input sources can files, pipes, and/or procedures. Data integration of this kind was not demonstrated for the sake of simplicity.

Input File *seqdata*

```
01  330170363  Stuart Clay         0056681.42  6   cT   101 B St              B
02  421901269  Taylor Guerrero     0015019.10  9   MD   1031 Park Ln Apt D     A
03  529433545  Charles Caldwell    0041116.71  3   NY   14 Main St            A
04  129737773  Robyn Puckett       0044558.62  3   ny   822 Hwy 76            B
05  594521240  Santiago Lindsey    0055836.11  0   TX   Star Rt Box 822       A
06  796569799  Charles Lindsey     0098525.58  2   TX   12746 Wolf Circle     A
07  384127387  Santiago Puckett    0059036.80  4   NY   321 Baltic Ct         B
08  711604065  Charles Williams    0018645.95  1   Tx   1103 Fresh Creek Ln   A
09  343054521  Jack Velazquez      0029205.44  2   NY   6780 Sand Dr Apt 3A   B
10  148354977  Donald Cooke        0044121.44  4   MA   35 La Palma Dr        A
```

The following job script products several output files in the same job script and I/O pass, which includes all 10 records from the above input files (though filters could have been applied:

```
### Sort Control Language (SortCL) Program ###
```

```
### Input Phase ###

/INFILE=(seqdata)                          # Reads 1 input file
       /FIELD=(idnum,POS=1,SIZE=2)        # Unique record identifier tag
       /FIELD=(ssno,POS=4,SIZE=9)
# Overdefines the social security number into parts for obfuscation
      /FIELD=(ssno_part1,POSITION=4,SIZE=1)
      /FIELD=(ssno_part2,POSITION=5,SIZE=4,NUMERIC)
      /FIELD=(ssno_part3,POSITION=9,SIZE=4,NUMERIC)
      /FIELD=(name,POSITION=14,SIZE=20)
       /FIELD=(ssno,POSITION=4,SIZE=9)
# Defines the last name letter field for conditional selection
      /FIELD=(last_name_letter,SEPARATOR=‘ ’,POSITION=4,SIZE=1)
      /FIELD=(salary,POSITION=36,SIZE=10,NUMERIC)
      /FIELD=(salary2,POSITION=36,SIZE=10)      # Redefined as ASCII for encryption
      /FIELD=(deduction_no,POSITION=47,SIZE=1)
      /FIELD=(state,POSITION=49,SIZE=2)
      /FIELD=(address,POSITION=52,SIZE=1)
      /FIELD=(group_code,POSITION=75,SIZE=1)
      /FIELD=(wholerec,POSITION=1,SIZE=71)
# Define a field to be the entire record


### Action Phase ###

/SORT
      /KEY=(group_code)           # Sort on code field for aggregation
      /KEY=(salary)               # Sort on salary field


### Output Phase ###

/OUTFILE=testdata          # First Output File. Obfuscates data, preserves format
      /FIELD=(idnum,POSITION=1,SIZE=2.0,FILL=‘0’,NUMERIC)
      /FIELD=(ssno_part1,POSITION=4,SIZE=1)
# Obfuscate the next 4 digits through conditional cross-calculation
      /FIELD=(ssno_part2_new,POSITION=5,SIZE=4.0,FILL=‘0’,NUMERIC, \
                IF ssno_part2 GT 4500 THEN ssno_part2 / 2 ELSE 2 * ssno_part2 - 55)

# Obfuscate the final 4 digits
      /FIELD=(ssno_part3_new,POS=4,SIZE=4.0,FILL=‘0’, NUMERIC, \
                IF ssno_part3 GT 4500 THEN ssno_part3 / 2 ELSE 2 * ssno_part3 - 54)

# Create pseudonym of the real name using the lookup table pseudo.set
      /FIELD=(name_fake,POSITION=14,SIZE=20,SET=pseudo.set[name])

# Make the salaries anonymous using conditional cross-calculation
      /FIELD=(salary_new,POSITION=35,SIZE=10.2,NUMERIC, \
             IF salary GT 50000.00 THEN 0.85 * salary ELSE 1.15 * salary)
      /FIELD=(deducation_no,POSITION=46,SIZE=1)
      /FIELD=(state,POSITION=46,SIZE=1)
      /DATA=“ ”
      /DATA={20}“*”        # Masks address data with asterisks
```

```
/OUTFILE=aggregate_salaries_by_group     # Summary record format
      /FIELD=(total_salary,POSITION=51,SIZE=12,CURRENCY)
      /SUM total_salary FROM salary BREAK group_code

/OUTFILE=aggregate_salaries_by_group     # Detail record format
      /FIELD=(group_code,POSITION=1,SIZE=1)
      /FIELD=(name,POSITION=3,SIZE=20)
      /FIELD=(address,POSITION=25,SIZE=20)
      /FIELD=(TOUPPER(state),POSITION=47,SIZE=2)    # Capitalization function
      /FIELD=(salary,POSITION=51,SIZE=12,CURRENCY)

OUTFILE=salaries_de_id
# De-identify fields using bit manipulation function (to preserve field size)
      /FIELD=(idnum,POSITION=1,SIZE=2.0,FILL='0',NUMERIC)
      /FIELD=(de_identify(salary,"abc"),POSITION=6,SIZE=10,NUMERIC)
      /FIELD=(TOUPPER(state),POSITION=20,SIZE=2)

/OUTFILE=encrypt_all
      /FIELD=(encryptAES256(wholerec),POSITION=1)   # Default key phrase used

/OUTFILE=encrypt_2fields
          # Encrypt 2 fields, each with a different key phrase
          # To determine the size of the encrypted output field:
          # Increase field size to next multiple of 16 and divide by 3
          # Round up to the next whole number and multiply by 4
      /FIELD=(encryptAES256(ssno,"passphr1"),POSITION=1,SIZE=24)
      /FIELD=(name,POSITION=26,SIZE=20)
      /FIELD=(encryptAES256(salary2,"passphr2"),POSITION=47,SIZE=24)
      /FIELD=(TOUPPER(state),POSITION=73,SIZE=2)

/OUTFILE=report.csv
      /PROCESS=CSV              # Creates header from fieldnames
      /FIELD=(idnum,POSITION=1,SEPARATOR=',',FRAME='"')
      /FIELD=(ssno,POSITION=2,SEPARATOR=',',FRAME='"')
      /FIELD=(name,POSITION=3,SEPARATOR=',',FRAME='"')
      /FIELD(salary,POSITION=4,SEPARATOR=',',FRAME='"')
      /FIELD=(deduction_no,POSITION=5,SEPARATOR=',',FRAME='"')
      /FIELD=(state,POSITION=6,SEPARATOR=',',FRAME='"')
      /FIELD=(address,POSITION=7,SEPARATOR=',',FRAME='"')
      /FIELD=(group_code,POSITION=8,SEPARATOR=',',FRAME='"')
```

This job script turns the two input files into six output files, all in one I/O pass. Many more inputs or outputs, of any size and format, could have been specified, and conditional selection criteria could have been applied against any source or target. A number of additional field level transformation functions could also have been specified. For a more complete list of available data manipulation functions, see the TECHNICAL SPECIFICATIONS chapter.

Outputs from the sample above follow, along with explanations of each:

Output File 1, *testdata*

```
02  443252484  Clifton Jimenez      17271.97    9  MD   ********************
08  722658076  Jeffery Gomez        21442.84    1  Tx   ********************
03  558317036  Teddy Black          47284.22    3  NY   ********************
10  124182488  Julio Koch           50739.66    4  MA   ********************
05  547262426  Spencer Craig        47460.69    0  TX   ********************
06  748284900  Landen Sullivan      83746.74    2  TX   ********************
09  385552260  Francisco Duffy      33586.26    2  NY   ********************
04  158913886  Salvador Jacobson    51242.41    3  ny   ********************
01  359790672  Ramsey Flynn         48179.21    6  cT   ********************
07  342063694  ALvaro Mcleod        50181.28    4  NY   ********************
```

This file shows safe, protected results in the form of similarly formatted test data. The data are sorted on group_code and then salary prior to the salary being protected; therefore, the salary order will not appear to be ordered. Note that the social security numbers were obfuscated with expression logic defined in 2 lines of the SortCL script Par 2 is shown below:

```
/FIELD=(ssno_part2_new,POSITION=5,SIZE=4.0,FILL='0',NUMERIC, \
      IF ssno_part2 GT 4500 THEN ssno_part2 / 2              \
      ELSE 2 * ssno_part2 - 55)
```

and that names were de-identified with pseudonyms in a lookup table called *pseudo.set*:

```
/FIELD=(name_fake,POSITION=14,SIZE=20,SET=pseudo.set[name])
```

SET File, *pseudo.set*

```
Charles Caldwell        Teddy Black
Charles Lindsey         Landen Sullivan
Charles Williams        Jeffrey Gomez
Donald Cook             Julio Koch
Jack Velazquez          Francisco Duffy
Robyn Puckett           Salvador Jacobson
Santiago Lindsey        Spencer Craig
Santiago Puckett        Alvaro Mcleod
Stuart Clay             Ramsey Flynn
Taylor Guerrero         Clifton Jimenez
/default/
```

In the above set file, the 'real name' Charles Caldwell is identified with the 'fake name' Teddy Black, and so on. A tab character separates the two names in the lookup table.

Output salaries were anonymized with math functions (which caused the appearance of disorder in that field). The TOUPPER syntax applied in the state field converted the input values to all uppercase letters on output; this is one of several built-in data quality functions. Finally, the last three input fields were redacted and masked with asterisks.

At the same time, this output file (with protected real data) was also produced:

Output File 2, *aggregate_salaries_by_group*

```
A  Taylor Guerrero      1031 Park Ln Apt D     MD    $15,019.10
A  Charles Williams     1103 Fresh Creek Ln    TX    $18,645.95
A  Charles Caldwell     14 Main St             NY    $41,116.71
A  Donald Cooke         35 La Palma Dr         MA    $44,121.44
A  Santiago Lindsey     Star Rt Box 822        TX    $55,836.11
A  Charles Lindsey      12746 Wolf Circle      TX    $98,525.58
                                                   $273,264.89
B  Jack Velazquez       6780 Sand Dr Apt 3A    NY    $29,205.44
B  Robyn Puckett        822 Hwy 76             NY    $44,558.62
B  Stuart Clay          101 B St               CT    $56,036.80
B  Santiago Puckett     321 Baltic Ct          NY    $59,036.80
                                                   $189,482.28
```

This is a detail and summary report, grouping records by their group code (A or B). Two same-named /OUTFILE sections were used: one to define the format of the detail records, and the other for the summary information. The group totals were derived with this syntax:

```
/SUM total_salary FROM salary BREAK group_code
```

Output File 3, *salaries_de_id*

```
02   335735<253     MD
08   335:9872<7     TX
03   33855592;5     NY
10   3388545288     MA
05   3377:69255     TX
06   33<:74727:     TX
09   334<437288     NY
04   338877:294     NY
01   33799:5284     CT
07   337<3692:3     NY
```

This third output file contains the same information, but without the header record. Salaries have been de-identified with an internal bit manipulation function using the pass code ‘abc’ (which creates the bit manipulation parameters). This method is akin to, but less secure than, encryption, but preserves the field size.

Output File 4, *encrypt_all*

```
lS7t5h+/s3sUbOs+42pYtkTau2K4kzZMOB4CrvE5QSJes2kvxrYIiWtg3y4VWYT7qIoc/rYwELAux4Gh3Cg9EoWf68OdPh+ATqOJ4xE6/T4=
Xh2nImJY4hBLfwWIDGeR1d0hNCC4Fbtz1UICR31wgr7rQo7byO1fVFGw5mwh+GSbh5OM7icopQ84lNVnODVuw6QpqLJCIpwkrYyjO0wvDFg=
twoxGCkGxWn76wrnHlBMX7V1AdRAqXne1l23i30ljkMhbWkf+K9E6JZ/iJtHElPi90bgeqGJyaXcMRMRHlcQkdV/wLAXJJ2b1Nl6Sz/A8uQ=
O1v4mpIxb+D+egm3MbovHPi1hnlMI67Kmdza38JvfIjEw5/tXLbFC4GllUK1eaaNDYdRuJw8CcV1wf05nJ1wQU5UmXmP8uIqeHW41tFK2Xo=
b7B4w8SWnGaM7gXFC6pGAu1qOwp3aX8vFTkRSmUF57hFLQ9IxmLjd7nqDQKMgPg9SVBvlcB4Jsk13TIL6NKM5J0TCKYIH38aVfcw6cG+LuA=
usnCrehsH109zkzDdCYJnfkTp35IjdgA+p4JXZAF3W4uPlqhieH4fKg5zLxYZPtfA1ThT4oWxNCV9Yb/Ey4m/oTM8CaqFderHAqRQJiqj5c=
e0DUXvL1/axHuLzgxPaCG0w/IT1QKcypks3al9hXCCHGh29yegi3Tp+9/nVGaWxVn/UFBYvuiVOJB1hw9RQuzmJMDsuEnr6zGbpl4gvzrCA=
ysO8dAWoGb2soHkp+W1pML7B2MXR8i5kKjzcHjCnxEqoU5Dtd/GXToy1fuM0CrJfmz3F/oRnlw6YhVlvAc46X+Cr033y0y5eCIrATGFqGkI=
Pe8gYs5l/3z9eQrxobmMo/FHI0c1QboyNy1mtWdWBa4/UekwCssPRnvYh8J17y4PrmE9V94WPTZ3RrL7rijQdz8WzS8ES1taZ/MpVaUJ0Gk=
LR5H66hXx7YtaaNSJRCiwmC2Hssc6YA3VAhSRtUQDWTkF/+vxonCrpR6tP87LphMB/k3hKUD5rxuaJkCLrugfZhD3XiLGDxnuryfOXhPeJo=
```

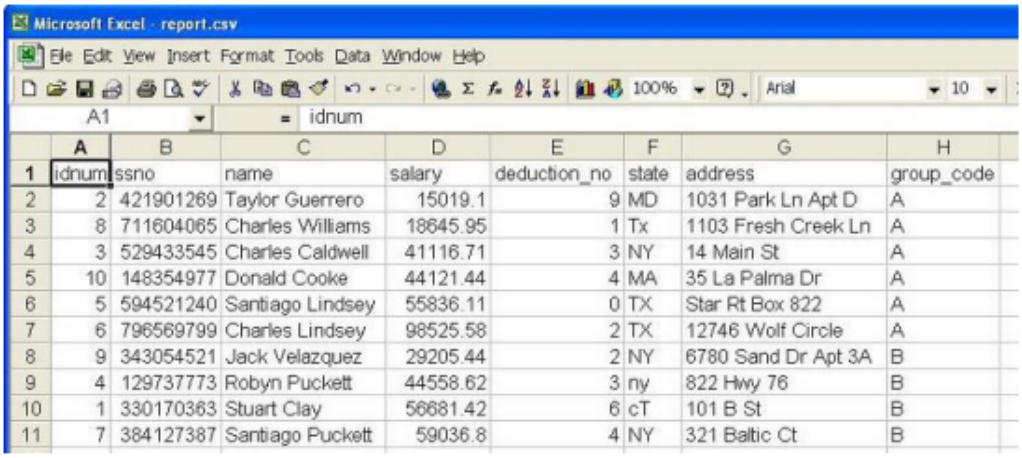
Here, the entire record was declared as a field and encrypted. Notice how the encrypted 'ciphertext' results contain only printable ASCII characters for printing and inclusion in text files.

Output File 5, *encrypt_2fields*

```
bUWn/CrhAql4MXNxoy7OkA==  Taylor Guerrero     2iyseYtWEd3ESFwE91433Q==  MD
8XvhrNe7ecF2DKFZyP4cNw==  Charles Williams   SK8auRS56YxRWMtLTXEGtg==  TX
pHxUwHhkcXW06JEhoDIlJg==  Charles Caldwell   bCz/W5bEpDpA2KJYDi3xvQ==  NY
sAol8DlsqS2viXEMPHullQ==  Donald Cooke       kpCZLODnoWFht50aH7u2LQ==  MA
eGoUczVoZEm95/9eBD8iOQ==  Santiago Lindsey   8inKXdYDU4AH9tJx8xlPAg==  TX
AJJEKjJuJQErVR726ytkZg==  Charles Lindsey     Vzm4/kcz/ypUNfWJjBBrcA==  TX
2KcftKVKDv+OCaG/FFdJ6w==  Jack Velazquez      m8MIpOmfPqxtNsO69cV0Mg==  NY
QMCKCxi32a9ZL3cYuBwHew==  Robyn Puckett       qwn2T7pUF/Hu+YPQBThcwg==  NY
CcxB9LuMkdAJ0E3rhwDIuA==  Stuart Clay         Q/bc3I756EPqn3TAYcqOUA==  CT
RQVeipLI4xlzvwzskHTE0Q==  Santiago Puckett    9i1fjr3CQs0yjuKV/tVCEw==  NY
```

The fifth output file, shown above, encrypts the social security number (SSN) and salary fields, but with two different passphrases so the data is protected for different disclosures.

Output File 6, *report.csv*



	A	B	C	D	E	F	G	H
1	idnum	ssno	name	salary	deduction_no	state	address	group_code
2	2	421901269	Taylor Guerrero	15019.1	9	MD	1031 Park Ln Apt D	A
3	8	711604065	Charles Williams	18645.95	1	Tx	1103 Fresh Creek Ln	A
4	3	529433545	Charles Caldwell	41116.71	3	NY	14 Main St	A
5	10	148354977	Donald Cooke	44121.44	4	MA	35 La Palma Dr	A
6	5	594521240	Santiago Lindsey	55836.11	0	TX	Star Rt Box 822	A
7	6	796569799	Charles Lindsey	98525.58	2	TX	12746 Wolf Circle	A
8	9	343054521	Jack Velazquez	29205.44	2	NY	6780 Sand Dr Apt 3A	B
9	4	129737773	Robyn Puckett	44558.62	3	ny	822 Hwy 76	B
10	1	330170363	Stuart Clay	56681.42	6	cT	101 B St	B
11	7	384127387	Santiago Puckett	59036.8	4	NY	321 Baltic Ct	B

The last output file above, viewed in a spreadsheet, shows how specifying /PROCESS=CSV on output automatically adds a header record with the output field names. The conversion of the source records to a comma-separated framework (including the addition of the header record) allowed the target to be read by Excel without additional processing.

Application Sample #3

Alphanumeric Format-Preserving Encryption

This example uses personal information data, **personal_info**, including credit card numbers, driver license numbers, and names.

```
9654-4338-8732-8128    W389-324-33-473-Q    Jessica Steffani
2312-7218-4829-0111    H583-832-87-178-P    Cody Blagg
8940-8391-9147-8291    E372-273-92-893-G    Jacob Blagg
6438-8932-2284-6262    L556-731-91-842-J    Just Rushlo
8291-7381-8291-7489    G803-389-53-934-J    Maria Sheldon
7828-8391-7737-0822    K991-892-02-578-O    Keenan Ross
7834-5445-7823-7843    F894-895-10-215-N    Francesca Leonie
8383-9745-1230-4820    M352-811-49-765-N    Nadia Elyse
3129-3648-3589-0848    S891-915-48-653-E    Gordon Cade
0583-7290-7492-8375    Z538-482-61-543-M    Hanna Fay
```

The following SortCL script encrypts the credit card and driver license number fields while preserving the field formats.

```
### Sort Control Language (SortCL) Program ###

### Input Phase ###
/INFILE=personal_info      # Reads 1 input file
       /FIELD=(credit_card,POS=1,SEP='\t')
       /FIELD=(driv_lic,POS=2,SEP='\t')
       /FIELD=(name,POS=3,SEP='\t')

### Action Phase ###
/REPORT

### Output Phase ###
/OUTFILE=personal_info_encrypted
      /FIELD=(credit_card1=FPT_ALPHANUM(credit_card,"pass",POS=1,SEP='\t')
      /FIELD=(driv_lic1=FPE_ALPHANUM(driv_lic,"pass",POS=2,SEP='\t')
      /FIELD=(name,POS=3,SEP='\t')
```

This produces **personal_info_encrypted**:

```
0832-9678-1911-0645    R784-107-86-619-Q    Jessica Steffani
0835-7171-0577-5699    G156-454-45-303-O    Cody Blagg
0789-2128-0461-5374    Q305-118-71-384-Q    Jacob Blagg
1591-0561-0417-5772    D344-156-20-555-G    Just Rushlo
9296-9613-4710-5436    U751-860-67-075-Y    Maria Sheldon
9881-4436-0773-0973    X878-716-85-252-C    Keenan Ross
4594-9802-2566-4840    T273-579-67-063-M    Francesca Leonie
6514-3079-6147-6828    A617-849-83-864-X    Nadia Elyse
9221-6125-6496-9606    S039-406-12-369-U    Gordon Cade
1404-8512-8389-2619    K379-587-05-591-C    Hanna Fay
```

Application Sample #4

Data Transformation and Reporting

This example uses stock trading data. It performs a full outer join of two differently formatted input files to produce three differently formatted targets. The script includes cross-calculation, aggregation, selection, and markup tags for BI, data interchange, and web posting purposes.

It is important to remember that the samples shown are small in order to illustrate combinable functionality, not speed in volume. The major benefit of SortCL, in addition to task consolidation, is its fast processing of big data sources together.

In this example, the input files to be joined are unsorted. The first, **nyse-a**, is in a tab-delimited format, and export from a database. The second input file, **buys.csv**, is in CSV format, typical in spreadsheet applications.

Input File 1, *nyse-a*

A. O. Smith Corporation	AOS	42.40	142900	0.04	0.09
A.G. Edwards Inc.	AGE	52.81	251800	0.48	0.91
A.O. Tatn EFT First	TNT	103.01	136000	1.01	0.99
AAG Holding Company1	GFZ	24.71	1900	0.06	0.24
AAG Holding Company2	GFW	25.05	4200	0.05	0.20
Aames Investment Corp.	AIC	4.84	145500	0.04	0.82
Aaron Rents, Inc.	RNT	24.05	1706300	2.53	9.51
ABB LTD.	ABB	12.55	3456100	0.13	1.04
Abbey National plc	SXA	25.00	24700	0.15	0.60
Abbott Laboratories	ABT	47.25	4210700	0.15	0.31
Abercrombie & Fitch	ANF	50.86	1973000	1.32	2.53
Abitibi-Consolidated	ABY	2.61	240600	0.00	0.00
ABM Industries Inc.	ABM	16.44	102600	0.363	2.14
ABN AMRO Holding N.V.	ABN	27.47	195900	0.31	1.14

Input File 2, *buys.csv*

```
Shares,Symbol,Client
"1000","DIS","Bill Gates"
"950","EDS","Ben Graham"
"25000","WMT","Warren Buffett"
"3250","AMR","Jeff Bezos"
"775","TSG","Wendi Deng"
"400","HBC","Stephen Covey"
"2100","HIG","Richard Branson"
"950","TEM","Sergey Brin"
"1500","AGE","Michael Bloomberg
"5000","BAC","Donald Trump"
"3333","PRU","Steve Wynn"
"2000","ABN","Jack Welch"
"8500","RNT","George Soros"
"1000","MCK","Kerry Packer"
"4300","UNH","Rupert Murdoch"
"9000","SDS","Jesse Livermore"
```

```
"3500","SNE","Alan Greenspan"
"825","ABT","Lakshmi Mittal"
"9000","ABY","Robert Kiyosaki"
"855","ADS","Lisa Mangino"
"50","IBM","Rick Haines"
"90","SUN","Amrita Thakur"
```

The goal of the following SortCL job script is to order and match these files by the ticker symbol in their second columns so that a meaningful set of output reports can be created. Field-level protections could certainly have been applied if desired.

```
### Job Controls Phase ###
/WARNINGSON
/MONITOR=4

### Input Phase ###
/INFILE=nyse-a        # 1st input, tab-delimited
       /FIELD=(Issue,POSITION=1,SEPARATOR='\t')
       /FIELD=(Symbol,POSITION=2,SIZE=3,SEPARATOR='\t')
       /FIELD=(LastTrade,POSITION=3,SIZE=5.2,SEPARATOR='\t',NUMERIC)
       /FIELD=(Volume,POSITION=4,SEPARATOR='\t',NUMERIC)
       /FIELD=(Change,POSITION=5,SIZE=4.2,SEPARATOR='\t', NUMERIC)
       /FIELD=(Percent,POSITION=6,SIZE=4.2,SEPARATOR='\t', NUMERIC)

/INFILE=buys.csv      # 2nd input, CSV format
/ALIAS=buys
/INSKIP=1             # skip header record
      /FIELD=(Shares,POSITION=1,SEPARATOR=‘,’,FRAME=‘“’)
      /FIELD=(Symbol,POSITION=2,SEPARATOR=‘,’,FRAME=‘“’)
      /FIELD=(Client,POSITION=3,SEPARATOR=‘,’,FRAME=‘“’)

### Action Phase ###
/JOIN FULL_OUTER NOT_SORTED nyse-a NOT_SORTED buys \
       WHERE nyse-a.Symbol EQ buys.Symbol


### Output File 1 - 2D BI Report with Selection ###
/OUTFILE=TradingA           # Summary record format
      /HEADREC="                          ----------\n"
      /FIELD=(New_balance,POSITION=50,SIZE=14.2,currency)
      /SUM New_balance from (nyse-a.LastTrade * buys.Shares) # Expression logic

/OUTFILE=TradingA           # Detail record format
      /HEADREC= “Client    Symbol     Shares   LastTrade   Shares*LT   Ln.\n\n”
      /FIELD=(buys.Client,POSITION=1,SIZE=17)
      /FIELD=(buys.Symbol,POSITION=20,SIZE=5)
      /FIELD=(nyse-a.Symbol,POSITION=28,SIZE=5)
      /FIELD=(buys.Shares,POSITION=35,SIZE=5)
      /FIELD=(nyse-a.LastTrade,POSITION=45,SIZE=5.2,NUMERIC),
      /FIELD=(product,POSITION=54,NUMERIC, IF nyse-a.Symbol NE buys.Symbol \
                THEN “” ELSE nyse-a.LastTrade * buys.Shares)
      /FIELD=(Sequencer,POSITION=66,SIZE=4)      # Creates sliding index column
```

```
### Output File 2 - Data Interchange Format ###
/OUTFILE=TradingA.xml
      /PROCESS=XML
      /FIELD=(Client,POSITION=1,SEPARATOR='|',XDEF="/Trades/Buy@Client")
      /FIELD=(Symbol,POSITION=2,SEPARATOR='|',XDEF="/Trades/Buy/Symbol")
      /FIELD=(Shares,POSITION=3,SEPARATOR='|',XDEF="/Trades/Buy/Shares")
      /INCLUDE WHERE nyse-a.LastTrade GT 10 AND buys.Shares GT 0


### Output File 3 - Web-ready Summary Report ###
/OUTFILE=TradingA.htm      # Summary record format
      /DATA="</FONT><B></TD>\N<TD align=right><B><U><FONT SIZE=+2>  \
                <FONT COLOR='GREEN'>"
      /FIELD=(New_balance,POSITION=3,SIZE=15,SEPARATOR='',TYPE=CURRENCY)    # Derived
in prior output spec!
      /DATA="</FONT></U></B></TD>\n</TR>\n"
      /SUM New_balance from (nyse-a.LastTrade * buys.Shares) WHERE symbol NE "RNT"
      /FOOTREC="</TABLE><BR>\nCreated on </B>%s. \
             <HR></BODY>\n</HTML>",AMERICAN_DATE

/OUTFILE=TradingA.htm       # Detail record format
      /HEADREC= "<HTML><HEAD>\n<TITLE>HTML produced by SORTCL \
                 </TITLE>\n</HEAD>\n<BODY><H2><FONT COLOR='RED'>Trading Summary \
                 </H2>\n<TABLE CELLPADDING=4 CELLSPACING=1 BORDER COLS=5>\n"
      /OMIT WHERE Symbol EQ "RNT" OR Symbol EQ ""    # Selection applied in display
      /DATA="<TR>\n<TD><I>"
      /FIELD=(buys.Client,POSITION=1,SEPARATOR='')
      /DATA="</B></TD>\n<TD align=right><FONT COLOR='BLUE'>"
      /FIELD=(buys.Symbol,POSITION=2,SEPARATOR='')
      /DATA="</TD>\n</TR>\n"
```

The following command ran the entire job above, producing all targets and job statistics at once:

```
Sortcl /SPECIFICATION=stockjoin.scl
```

And as the outputs were created, requested warning and monitor messages were displayed:

```
   +16 stockjoin.scl: warning (101): "PRECISION" ambiguous reference
   +18 stockjoin.scl: warning (101): "PRECISION" ambiguous reference
   +19 stockjoin.scl: warning (101): "PRECISION" ambiguous reference
   +38 stockjoin.scl: warning (101): "PRECISION" ambiguous reference
   +47 stockjoin.scl: warning (101): "PRECISION" ambiguous reference
       warning TradingA
     gap [1 -> 49]
       warning TradingA
     gap [18 -> 19]
     gap [25 -> 27]
     gap [33 -> 34]
     gap [40 -> 44]
       warning TradingA.htm
           overlap [1 -> 14]
       warning TradingA.xml
           missing field before field 175
           missing field before field 177

CoSort Version 10.5.1 D10570608-1306 Copyright 1978-2024 IRI, Inc. www.iri.com
EDT 09:25:53 AM Monday, July 17 2023. #22162.TEST 2 CPUs
```

```
Expires Dec 31, 2023       Monitor Level 4
   <00:00:00.00> event (57): /spec=stockjoin.scl initiated
   <00:00:00.00> event (57): /infile=nyse-a initiated
   <00:00:00.03> 2 CPUs
   <00:00:00.14> event (59): P5a5b88 infile opened
   <00:00:00.14> event (59): P5a5c08 infile opened
   <00:00:00.00> event (57): /infile=buys.csv initiated
   <00:00:00.06> event (66): cosort() process begins
   <00:00:00.14> event (59): d:\CS_JOIN_7d8.6fc infile opened
   <00:00:00.14> event (66): cosort() process begins
   <00:00:00.22> event (67): cosort() process ends
   <00:00:00.22> event (58): /infile=buys.csv completed
   <00:00:00.33> event (61): TradingA outfile opened
   <00:00:00.33> event (61): TradingA.xml outfile opened
   <00:00:00.33> event (61): TradingA.htm outfile opened
   <00:00:00.30> event (67): cosort() process ends
   <00:00:00.30> event (58): /infile=nyse-a completed
   <00:00:00.34> event (60): P5a5b88 infile closed0
   <00:00:00.34> event (60): P5a5c08 infile closed0
   <00:00:00.34> event (68): left 0 right 0
   <00:00:00.34> event (62): TradingA outfile closed
   <00:00:00.34> event (62): TradingA outfile closed
   <00:00:00.34> event (62): TradingA.xml outfile closed
   <00:00:00.34> event (62): TradingA.htm outfile closed
   <00:00:00.34> event (62): TradingA.htm outfile closed
   <00:00:00.39> event (58): /spec=stockjoin.scl completed
EDT 09:25:53 CoSort Serial # 22162.TEST 2 CPUs Expires Dec 31, 2024
```

In **TradingA**, both matches and non-matches are shown in order by ticker symbol. The cross and down-row calculations support business intelligence and billing operations.

Output File 1, *TradingA*

Client	Symbol		Shares	LastTrade	Shares*LT	Ln.
		ABB		12.55		1
		ABM		16.44		2
		ABN		27.47		3
Lakshmi Mittal	ABT	ABT	825	47.25	38981.25	4
Robert Kioysaki	ABY	ABY	9000	2.61	23490.00	5
Lisa Mangino	ADS		855			6
Michael Bloomberg	AGE	AGE	1500	52.81	79215.00	7
		AIC		4.84		8
Jeff Bezos	AMR		3250			9
		ANF		50.86		10
		AOS		42.40		11
Donald Trump	BAC		5000			12
Bill Gates	DIS		1000			13
Ben Graham	EDS		950			14
		GFW		25.05		15
		GFZ		24.71		16
Stephen Covey	HBC		400			17
Richard Branson	HIG		2100			18
Rick Haines	IBM		50			19
Kerry Packer	MCK		1000			20
Steve Wynn	PRU		3333			21
George Soros	RNT	RNT	8500	24.05	204425.00	22
Jesse Livermore	SDS		9000			23

```
Alan Greenspan          SNE             3500                                    24
Amrita Thakur           SUN             90                                      25
                                SXA             25.00                           26
Sergey Brin             TEM             950                                     27
                                TNT             103.0                           28
Wendi Deng              TSG             775                                     29
Rupert Murdoch          UNH             4300                                    30
Warren Buffett          WMT             25000                                   31
                                                                  -----------
                                                                  $346,111.25
```

The next target, in valid XML format, contains only the selected fields and condition results:

Output File 2, *TradingA.xml*

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <Trades>
  - <Buy Client="Lakshmi Mittal">
      <Symbol>ABT</Symbol>
      <Shares>825</Shares>
    </Buy>
  - <Buy Client="Michael Bloomberg">
      <Symbol>AGE</Symbol>
      <Shares>1500</Shares>
    </Buy>
  - <Buy Client="George Soros">
      <Symbol>RNT</Symbol>
      <Shares>8500</Shares>
    </Buy>
</Trades>
```

Output File 3, *TradingA.htm*

IRI Workbench

CoSort ships with a free Graphical User Interface (GUI) for Sort Control Language (SortCL) job design and management. IRI Workbench is a plug-in to the familiar Eclipse™ Integrated Development Environment (IDE) to help users create, maintain, run, and share SortCL -- and other IRI and non-IRI -- jobs. It provides rich, ergonomic functionality and future extensibility.

Job Wizards, Dialogs, Diagrams and Enhanced Script Editing

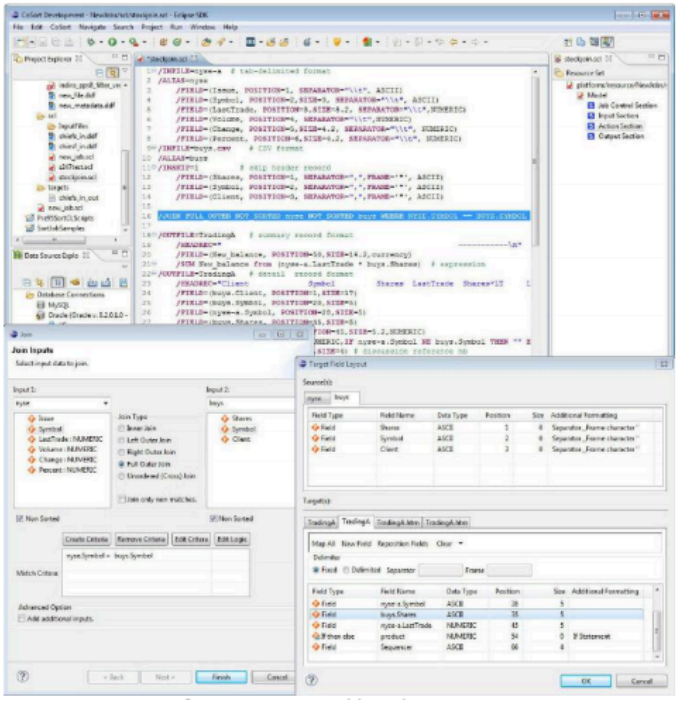
IRI Workbench can use multiple presentation facilities in Eclipse to improve the SortCL job design experience. The wizards take you from source and metadata specification through the action phase of a job, and finally to the specification of one or more targets and formats. Wizard and form dialog results help populate and modify SortCL job scripts. Workflow and transformation mapping diagrams can also be built from scripts (or vice versa).

For those who prefer direct coding in the 4GL, a syntax-aware editor facilitates valid script creation and modification. Changes to DDF or .SCL code made in the editor or the other interfaces dynamically update each other; i.e, everything is model-driven and re-entrant.

The top window in the figure below shows the editor and outline view of a join job script. You can invoke the join wizard (shown in the bottom left) from the main menu and within the script editor view.

The join wizard allows you to specify all details of a multi-table join action. The bottom right window shows the target field layout of this job. Note that multiple input sources (shown in upper tabs) and output targets (shown in lower tabs) are displayed to help you specify file and table target layouts quickly and easily.

Figure 3 *IRI Workbench Overview*



ODBC-Connected (DB Table) Data Sources and Targets

In addition to their flat file inputs, CoSort users can integrate, stage, transform, protect, and report against data stored in Oracle, DB2, SQL Server, SAP, MySQL, Sybase, Excel, and other ODBC-connected sources. Workbench uses the JDBC-ODBC bridge in the Eclipse Data Tools Platform (DTP) for viewing and selecting table data.

In the graphic below, the left panel shows the DTP view of a database. The middle panel displays the contents of the selected table, and the right panel is the Data Definition File (DDF) form editor that is used to modify the table metadata specifications for a SortCL job.

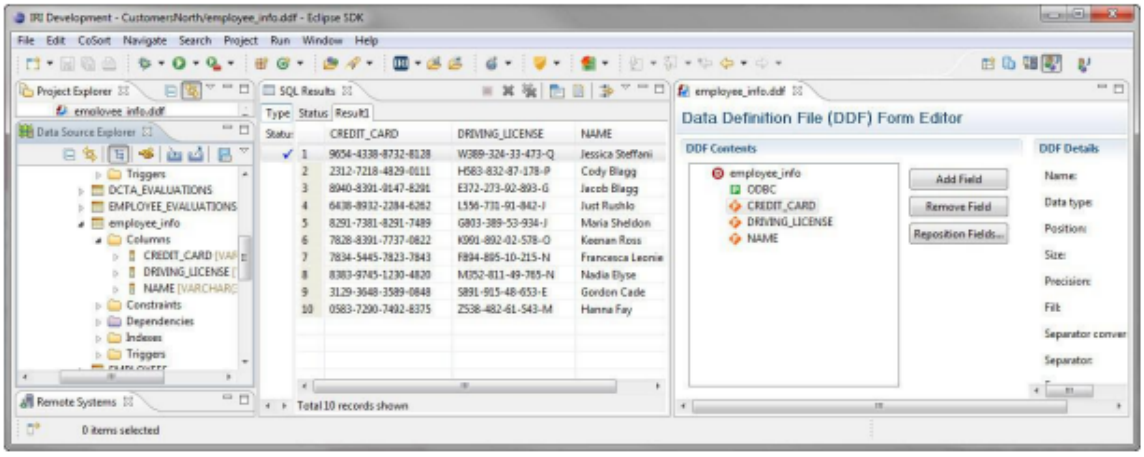


Figure 4 *DTP and DDF Editor*

Metadata Discovery

Those familiar with CoSort know that you must define the structure of all input and output files in SortCL DDF syntax. This has traditionally been a manual field-by-field editing process, or the result of a command line conversion program like cob2ddf (for COBOL copybooks). When pre-existing metadata is not available, IRI Workbench helps users to visually define their field layouts and populate file and table metadata for use in SortCL jobs.

The following screenshot shows how users can define fixed-position fields by moving sliders to the start and end of each field in a source data preview window. Once you add a field, the bottom spreadsheet view reflects the DDF specifications which you can then modify.

For records with delimited fields, a column-based editor is provided instead, as sliders are unnecessary. A HEX view option facilitates the definition of binary data.

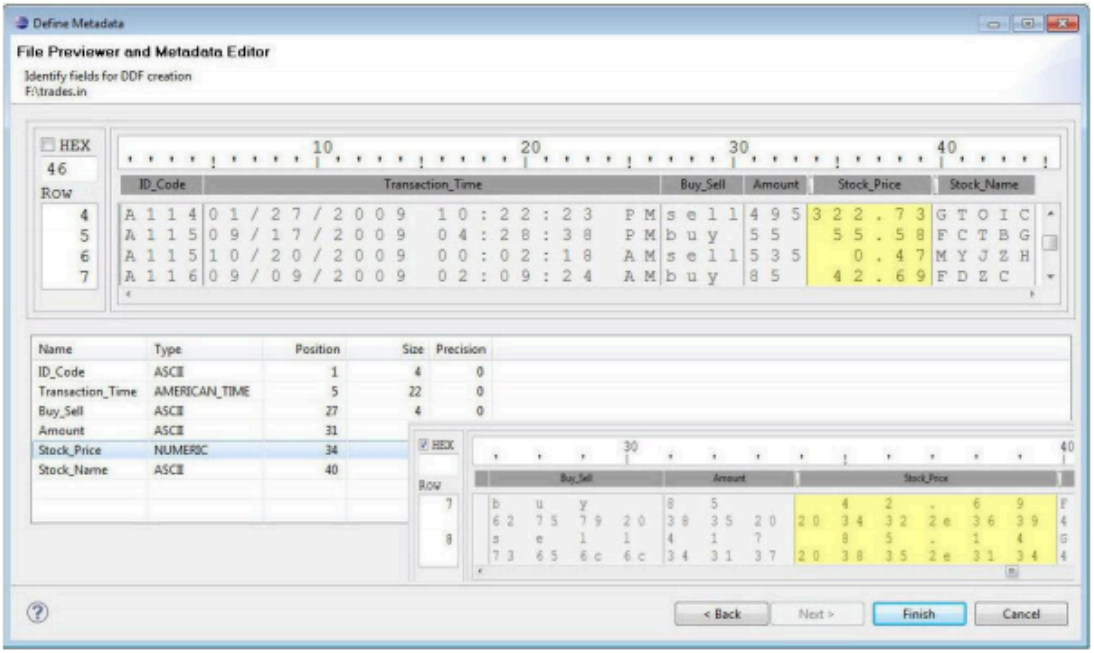


Figure 5 *Define Metadata Wizard*

Metadata Conversion

The process of bulk metadata and third-party sort script migration to CoSort SortCL syntax has been modernized and automated in the Workbench. Wizards exist to translate third-party data layouts into SortCL DDFs and job scripts.

An example of the former would be bulk COBOL copybook conversions to SortCL .ddf targets. There is a wizard to convert one or more metadata repositories or file headers into DDF. There is also a wizard to convert and import third-party sort parms into SortCL job scripts.

The conversion wizard shown below demonstrates the conversion of one or more JCL sort decks to SortCL job scripts. You can browse to the location of the parms, and then automatically convert them for use in SortCL.

The middle window in the screenshot below shows advanced options available for these conversions. The bottom window shows from left to right: project explorer files (including the source and target job scripts), an editor displaying the source job script, a display of the target job script, and a tree view of the components of the target job script.

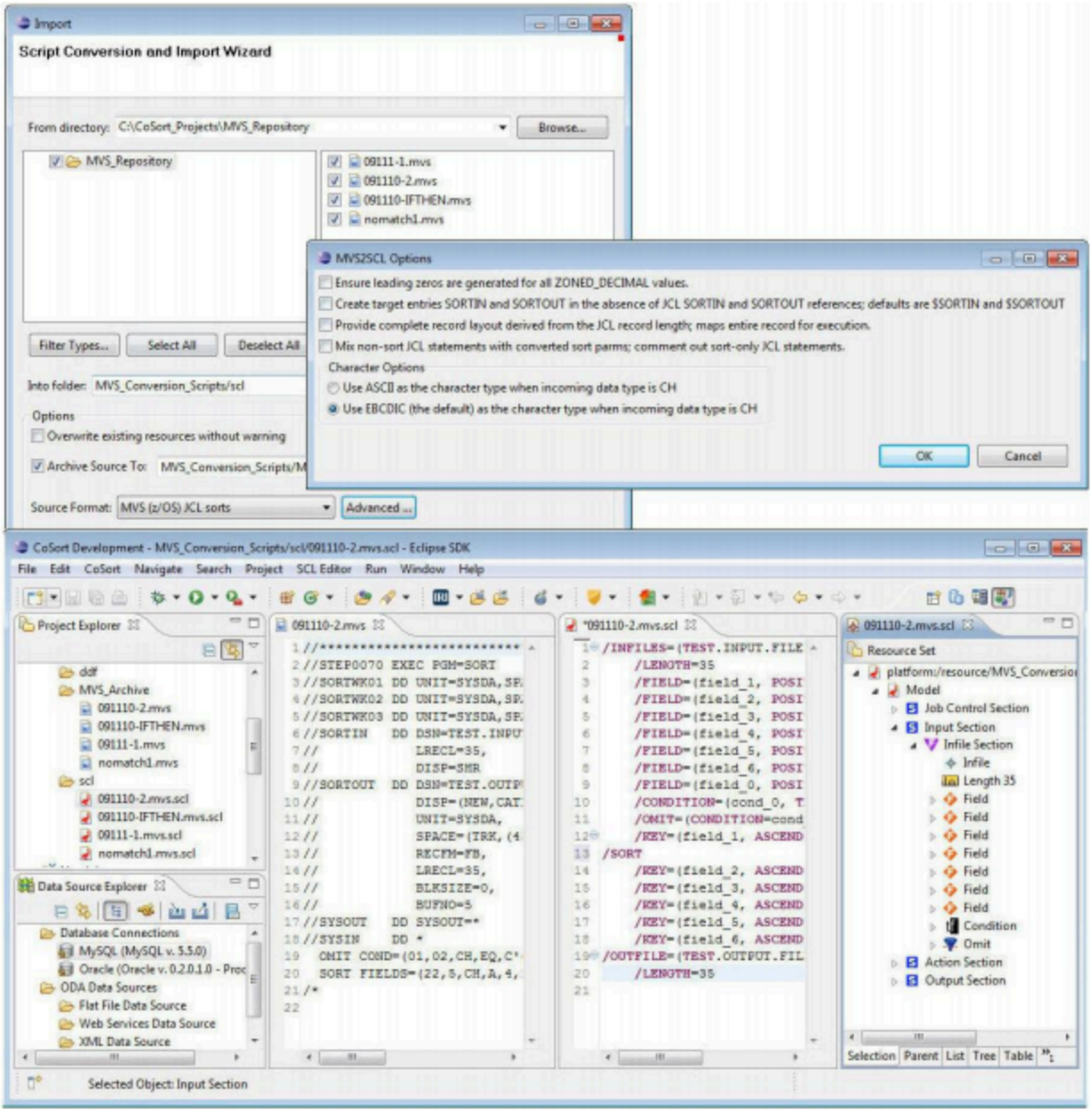


Figure 6 *Import Wizard and Imported Script*

SortCL Command Set

The following is a list of the core commands available in SortCL as of CoSort v10:

```
/ALIAS
/APPEND
/ALTSEQ
/AVERAGE
/CHARSET
/CHECK
/CONDITION
/COUNT
/CREATE
/DATA
/DEBUG
/DELETE
/DUPLICATESONLY
/ENDIAN
/EXECUTE
/FIELD
/FIELD_PREDICATE
/FILE
/FOOTREC
/HEADREAD
/HEADREC
/HEADSKIP
/HEADWRITE
/INCLUDE
/INCOLLECT
/INFILE
/INFILES
/INPROCEDURE
/INREC
/INSERT
/INSKIP
/JOBCOLLECT
/JOBSKIP
/JOIN
/KEY
/KEYPROCEDURE
/LENGTH
/LIBRARY
/LOCALE
/MAXIMUM
/MINIMUM
/MERGE
/MONITOR
/NODUPLICATES
/OMIT
/OUTCOLLECT
/OUTFILE
/OUTPROCEDURE
/OUTSKIP
/PROCESS
/QUERY
/RANDOM
/RC
/RECSPERPAGE
/REPORT
/ROUNDING
/SORT
/SPECIFICATION
/STABLE
/STATISTICS
/STREAM
/SUM
/TAILREAD
/TAILSKIP
/TAILWRITE
/TEMPLATE
/UPDATE
/WARNINGSOFF
/WARNINGSON
```

Note that each command may contain additional parameters that expand its functionality. For example, the `/JOIN ONLY` command will eliminate the inner join results of a full outer join. Note also that many external functions can be invoked beyond the command set. For example, encryption and other masking functions, as well as set file lookups, are specified as functions within `/FIELD` statements.

Metadata Conversion Tools

CoSort includes several tools to translate existing flat file layout (and sort job) metadata for use in Sort Control Language (SortCL) programs. IRI has also partnered with leading metadata generation and conversion vendors like DataSwitch, erwin, and Meta Integration Technology to auto-create SortCL-compatible data definitions from third-party BI, ETL and RDB metadata.

About SortCL Metadata

The explicit text-based metadata of SortCL is straightforward and self-documenting -- making it easy to learn, use, and audit. You can store your files' field (column) layouts in reusable Data Definition File (.ddf) repositories. You can paste or reference these definitions in your SortCL job script. SortCL uses `/FIELD=` statements to identify column:

- Names or aliases
- Sizes or ranges
- Positions or delimiters
- Data types
- Conditions and expressions
- Security and other functions

Third-Party Metadata

File layout translators included in the CoSort package can migrate your existing field/column descriptions into SortCL (and RowGen) .ddf repositories. These tools can reduce or eliminate the overhead associated with converting existing metadata into .ddf from:

- COBOL copybooks (cob2ddf)
- Comma-separated values (csv2ddf)
- Excel (XLS and XLSX spreadsheets, xls2ddf)
- LDIF (ldif2ddf)
- MongoDB collections (Workbench)
- XML (xml2ddf)
- W3C extended log format (elf2ddf)
- Oracle SQL*Loader control files (ctl2ddf)
- ODBC table data

Therefore, if you already have file layouts defined in the above applications, you can automatically reproduce that metadata for use in any SortCL manipulations. If your file layouts exist in other formats (including CWM, DSX, UML, XMI and XML), the Meta Integration Model Bridge (MIMB) will convert these repositories into SortCL .ddf syntax. This precludes the need for manually re-defining flat file field layouts for reference in SortCL applications.

Sort Program Conversions

The included mvs2scl, and vse2scl utilities convert legacy MVS JCL, and VSE JCL, respectively, to functionally-equivalent SortCL job specifications. Conversion tools cannot translate every script. However, field re-casting usually works, and manual translation of source scripts to SortCL equivalents is usually not difficult. For more information on, and examples of, these tools, see IRI's Legacy Sort Migration white paper.

Another available utility, called sorti2scl, translates legacy CoSort Sort Interactive (SortI) program specifications (.spc) into their SortCL script (.scl) equivalents. SortI is not provided in CoSort V10 and above.

Unix Sort Replacement (bin/sort)

The /bin/sort utility provided with the Unix operating system is designed for ordering small collections of alphanumeric data. It does not work well when the size of the data increases beyond available memory. For those with an investment in, or familiarity with, existing system sort commands, IRI provides a drop-in /bin/sort replacement that improves high-volume sort performance through the CoSort engine.

Unix users are provided with a new /bin/sort, and Windows users get a unixsort.exe program. Upon installation, the object file can be moved into a system directory to provide sort services with the same syntax as the original sort verb, but at much higher performance levels.

An example of the Unix sort replacement in CoSort follows, using the following data sample:

Input File 1, *chicago*

```
5180   On Top 15.95 Harper-Row
3391   Married Yount 24.95 Prentice-Hall
8835   Beginnings 8.50 Prentice-Hall
2272   Still There 13.05 Dell
1139   Greater Than 34.75 Valley Kill
3928   Not On Call 9.99 Harper-Row
4877   Going Nowhere 17.95 Valley Kill
```

We first define the way to specify fields for the sort key. White space denotes the end of a field unless a field separator character is defined.

To sort **chicago** starting with the second field, use the command:

```
/path2/cosort95/bin/sort -k 2 chicago -o out1
```

Output File, *out1*

```
8853   Beginnings 8.50 Prentice-Hall
4877   Going Nowhere 17.95 Valley Kill
1139   Greater Than 34.75 Valley Kill
3391   Married Young 24.95 Prentice-Hall
3928   Not On Call 9.99 Harper-Row
5180   On Top 15.95 Harper-Row
2272   Still There 13.05 Dell
```

To sort starting at the second character of the first field, use the following command:

```
Sort -k 1.2 chicago -o out2
```

***Output File,* out2**

```
1139   Greater Than 34.75 Valley Kill
5180   On Top 15.95 Harper-Row
2272   Still There 13.05 Dell
3391   Married Young 24.95 Prentice-Hall
8835   Beginnings 8.50 Prentice-Hall
4877   Going Nowhere 17.95 Valley Kill
3928   Not On Call 9.99 Harper-Row
```

The CoSort /bin/sort replacement also supports legacy Unix sort options. For example:

```
Sort -t, +3 +2nr -3 chicago -o out3
```

Output File, *out3*

```
2272   Still There 13.05 Dell
5180   On Top 15.95 Harper-Row
3928   Not On Call 9.99 Harper-Row
3391   Married Young 24.95 Prentice-Hall
8835   Beginnings 8.50 Prentice-Hall
1139   Greater Than 34.75 Valley Kill
4877   Going Nowhere 17.95 Valley Kill
```

The CoSort Unix sort replacement supports these command-line flags:

```
[-c] [-d] [-m] [-f] [-u] [-i] [-o] [-M] [-T] [-b] [-n] [-t] [-r] [-z]
```

`[-y]` and `[-Kmem]` flags are supported indirectly via values defined in your cosortrc file. For more information, see the SYSTEM TUNING chapter.

COBOL Migration Tools

CoSort ships with the tools, libraries, and documented examples you need to achieve a wide range of COBOL data migration and processing goals on open systems:

Sorting and Migrating COBOL Data

All CoSort interfaces support MF and RM COBOL data type collation while SortCL also handles conversion. CoSort includes a COBOL copybook metadata translation tool to leverage your file layouts in SortCL applications, and JCL sort parm conversion tools to leverage your MVS and VSE cards. CoSort can also sort EBCDIC data and can sort ASCII data in EBCDIC order. Multiple conversions can be done while simultaneously sorting.

Accelerating Native Sort Calls

CoSort is faster than most compiler-included sort functions, and CoSort packages include several tools and methods to improve COBOL sort performance:

- Sort replacement for ACUCOBOL-GT
- Sort replacement for Micro Focus COBOL
- Serial and concurrent system calls to SortCL
- Static and dynamic API calls to CoSort libraries

Sorting and Converting Index, Variable & Blocked Files

The CoSort SortCL program can perform multiple manipulations and conversions on ACUCOBOL-GT Vision files, Micro Focus variable length records, and index files in MF-ISAM and VISION formats. In the SortCL (4GL) job script, you define your input and output file formats and record layouts, along with your data filtering and transformation (sort, join, aggregate, etc.). You can integrate these formats with files in other formats all at the same time, and write output files in the same or different file format.

Generating Reports

The CoSort SortCL program includes a wide range of reporting features you can exploit to customize detail and summary targets for presentation and hand-offs to other tools.

Protecting Sensitive Data

Either as a separate process, or in combination with the above SortCL activities, you can also engage field-level protection functions like encryption and de-identification.

Creating Safe COBOL Test Data

Both CoSort and the fit-for-purpose IRI RowGen test data product uses SortCL syntax to define the layout of COBOL files that can contain randomly-generated and/or randomly-selected test data.

Auditing Data and Applications

For data governance and other tracking purposes, SortCL offers several logging options, including a query-ready XML file with entire scripts (containing field names and masking function specifications).

Application Programming Interfaces (APIs)

In addition to the many standalone utilities and third-party sort replacements in each CoSort package, there are two high-performance, thread-safe sorting libraries you can call into your software. Each API satisfies a different class of requirements.

You can link these C routines statically or dynamically, and the same calling code runs across all Unix, Linux and Windows platforms. Both libraries leverage the same multi-threaded CoSort sorting routine against any volume of input. Inputs and outputs can be in the form of files, pipes, records and record buffers (blocks) streaming to and from multiple calls simultaneously from your applications.

Integrating Sort/Merge Operations Only

The traditional CoSort API is now thread-safe, and is documented as cosort_r(). You can call cosort_r() to speed operations that sort or merge high volumes of data. Because your programs configure the input, compare, and output processes into the CoSort engine, you also can apply your own selection and comparison criteria.

The 'r' in cosort_r() refers to the re-entrant nature of the call; you can call the function recursively from multiple processes. This means you can specify multiple sort orderings on the same input, and in the same pass. Flexible architecture also allows you to manage several sort jobs from within a single process, and from within as many processes as you like.

Integrating Multiple Transformation Functions

The multi-purpose SortCL program is also available for thread-safe application calls and can enable the simpler execution of scripts. Embed sortcl_routine() library to speed and combine functions, including:

- Sort/Merge
- Match/Join
- Aggregate/Calculate
- Filter/Scrub
- Type-Conversions
- Encrypt/De-ID
- Reporting
- Random Data Generation

This API gives you access to all the data transformation, business intelligence, protection, and prototyping functions available in SortCL data definition and manipulation syntax. Integrating sortcl_routine() into an ETL environment allows you to source and target database tables, as well as files, pipes, and custom input/output procedures.

The following C program demonstrates the simplicity of calling SortCL scripts via the sortcl_routine(api):

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*  Copyright 1978 - 2023 CoSort / Innovative Routines International (IRI), Inc.
*  All Rights Reserved.
*
*  SortCL API example to run a script file
*  ScriptFile: keyproc.scl
*  Input File: chiefs
*  Output File: chiefs.out
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* cosort header file */

#if defined (_WIN32)
#define __DDL_IMPORT__ /* IMPORTANT: needed to work with Windows DDL */
#pragma pack(1) /* or compile with /Zpl option */
#endif /* (_WIN32) */
#include "cosort.h
#include "sortcl_routine.h"

int main()
{
int iRetVal; /* return value */
cs_sortcl_t* sortcl; /* main sortcl context variable */
/* allocate the main sortcl context variable */
sortcl = sortcl_alloc();
if (sortcl) {
/* call sortcl_routine api */
iRetVal = sortcl_routine(sortcl, "/spec=keyproc.scl");
/* free the sortcl variable */
sortcl_free(sortcl);
}
/* if return value != 0 it is an error */
return (iRetVal);
}
```

You can make calls to the sortcl_routine library from any language that links to C. For more information about the content of SortCL job specification (.scl) files, see the functional description and scripting examples in the SORTCL PROGRAM chapter.

COBOL programmers may also wish to consider the ability to call SortCL scripts as system calls; for example:

```
CALL “system” USING “sortcl /SPECIFICATION=keyproc.scl &”
```

where **keyproc.scl** starts in the background while other COBOL program functions run.

System Tuning

High-volume sorting and related data manipulations can be very resource-intensive. CoSort can achieve scalability that is nearly linear through its proprietary processing techniques and the proper application of manual and automatic system tuning. At the same time, however, CoSort is designed to be a good neighbor in a multi-user computing mix.

System administrators can combine kernel and CoSort tuning to optimize the performance of large data transformations without unduly impacting concurrent jobs. To prioritize CoSort operations at the system (global), user, or job level, adjust the values associated with the parameters shown below in your resource control file(s) (on Unix) or Windows registry:

Threads

On multi-core platforms, CoSort users can manually set the number of threads that sorts and related transformations will spawn and use. In most cases, the closer this value is set to the actual number of CPU cores and disk drives on the system, the better the performance.

Memory Allocation

By assigning, or limiting the amount of random access memory available for sorting, system administrators can maximize the efficiency of jobs according to their desired system priority. Like other CoSort resource controls, automatic and manual memory tuning variables allow users to determine how much impact CoSort will have on concurrently running applications.

Block Size

Blocks of memory are used as buffers to hold data temporarily. The data move through input, sort and output phases of the CoSort utilities (or a customized front-end), and between an application (calling) program and the cosort() engine. The size of these blocks determines how often disk I/O will be performed, and is a function of how much memory is available.

Workfile Compression and Storage

Overflow occurs in sorting when there are more input records than can be held in memory. When all the overflow data are distributed to temporary files, these files and the internal data are merged to produce output. Users can control the compression ratio and location of these temporary files and distribute them across multiple file systems. Where multiple drives and threads are specified, speed can increase in the later stages of the sort as the data is read back (into the output file/s) in parallel. You can also use input selection to reduce processing volume and temporary space requirements at runtime.

CoSort resource controls also include these application-specific values:

Record Terminator

Where variable length output is specified, any record terminating character can be specified. By default the same terminator present on input is used on output. You can also select a Windows (CRLF), Unix (LF), or your own special terminator, including a NULL character ("").

Century Window

Users with non-Y2K-compliant (2-digit date) data can specify the minimum year for CoSort to sort after 1999. This “sliding” feature allows custom collation for any century-bordering dates.

Pause / Resume

During large sort operations that require the creation of temporary work files, CoSort can warn users when temporary disk space is exhausted, and allow for ad hoc re-allocation so that the job can continue without having to be re-started.

Runtime Monitoring

By specifying various levels of verbosity, CoSort users can view on-screen job progress reports with timestamps and event messaging. Events include the opening and closing of input, work, and output files and the number of records accepted, rejected or processed. Displays range from off (level 0) to showing every single record (number) being processed (level 9).

Execution Log

Basic runtime information can be directed to one or more files to archive CoSort jobs and their performance. This log file is a self-appending text file. A self-replacing text file called *.cserrlog* is created during each run to record tuning and version information, and any error messages, in the event of an abnormal termination (for debugging purposes).

Audit Log

Detailed runtime information can be optionally directed to a self-appending, query-ready XML log file that contains environmental and performance details, and the entire SortCL job script, to record every aspect of the application (data definitions, manipulations and protections) in order to verify compliance with procedures and data privacy regulations.

A complete description of all parameters, and how to tune them, can be found in Appendix D of the **CoSort User Manual**.

Technical Specifications

CoSort is a general-purpose data manipulation package for big data transformation, reporting, protection and conversion. Its utilities and API functions exploit the same coroutine engine to sort and otherwise manipulate all data sources and record types specified below. CoSort performs in-memory, record-level processing and leverages multiple threads, task consolidation, automatic tuning, and file system (not database) resources to optimize application without slowing down concurrent jobs.

Installation

- Distributed via internet or user-specified media
- Loads in under two minutes
- Menu-driven setup and configuration utility

Invocation

- Command line (including pipe sequences), shell commands and batch scripts
- IRI Workbench - Eclipse™ IDE for Windows, Linux, and MacOS (Sierra)
- Application calling programs as a standalone executable, subroutine or coroutine call, with or without additional exit routines
- IRI Workbench task scheduler or third-party schedulers (e.g., cron, Autosys, Stonebranch UC)

Ease of Use

- Processes data using record layouts and SQL-like field descriptions within applications or centralized data definition files (DDF)
- Converts and processes native COBOL copybook, Oracle SQL*Loader control file, CSV, and W3C extended log format (ELF) file layouts
- SortCL DDF is a supported AnalytiX DS and MITI MIMB metadata format
- Provides online help, pre-runtime application validation, and runtime errors
- Leverages centralized application and file layout definitions (metadata repositories)
- Reports problems to standard error when invoked from a program, or to an error log
- Runs silently or with verbose messaging without user intervention
- Allows user control over the amount of informational output produced
- Generates a query-ready XML audit log for data forensics and privacy compliance
- Describes commands and options through man pages and online documentation
- Fully supports job design, execution, modification, and tuning in familiar Eclipse™ GUI
- Easy-to-use interfaces and seamless third-party sort replacements preclude the need for training classes; however, advanced training is available in Florida or at user sites
- Phone, web, and email support available directly from the product developers
- Local language support is available from more than 40 international offices

Resource Control (See System Tuning above)

- Sets and allows user modification of the maximum and minimum number of concurrent threads for sorting on multi-CPU and multi-core systems
- Uses a specified directory or combination of directories for temporary files
- Supports automatic and manual compression (and ratios) for temporary files
- Automatically sets ceilings and floors for main and virtual memory used during sort operations
- Sets the size of the memory blocks used as physical I/O buffers

Input and Output

- Processes any number of supported data sources of any size, and any number of records, fixed or variable length (to 65,535 bytes)
- Sources can be: flat files (including CSV, LDIF, COBOL and XML), semi-structured JSON files, ODBC-connected tables or Excel spreadsheets, ASN.1 CDRs, an input procedure, **stdin** or a named pipe, a brokered Kafka or MQTT topic, HTTP/S streams, and supported file formats in Amazon S3 buckets, Azure Blob & GCP storage, or HDFS
- Supports the use of environmental variables and wildc/ards in the specification of input and output files, as well as absolute path names and aliases
- Accepts and outputs fixed- or variable-length records with delimited fields
- Generates one or more detail or summary targets, plus ODA output for BIRT
- Returns sorted, merged, or joined records one (or more) at a time to an output procedure, to **stdout**, a named pipe, a table in memory, one or more new or existing structured (or JSON) files, a Kafka or MQTT broker, S/FTP, or to an application program
- Outputs optionally with GUID, UUID, or sequence numbers for each record, at any starting value, for indexed loads and/or reports
- Supports inline .gzip (compressed file) read/write operations
- Synthesizes randomly generated or randomly selected test data values

Record Selection and Grouping

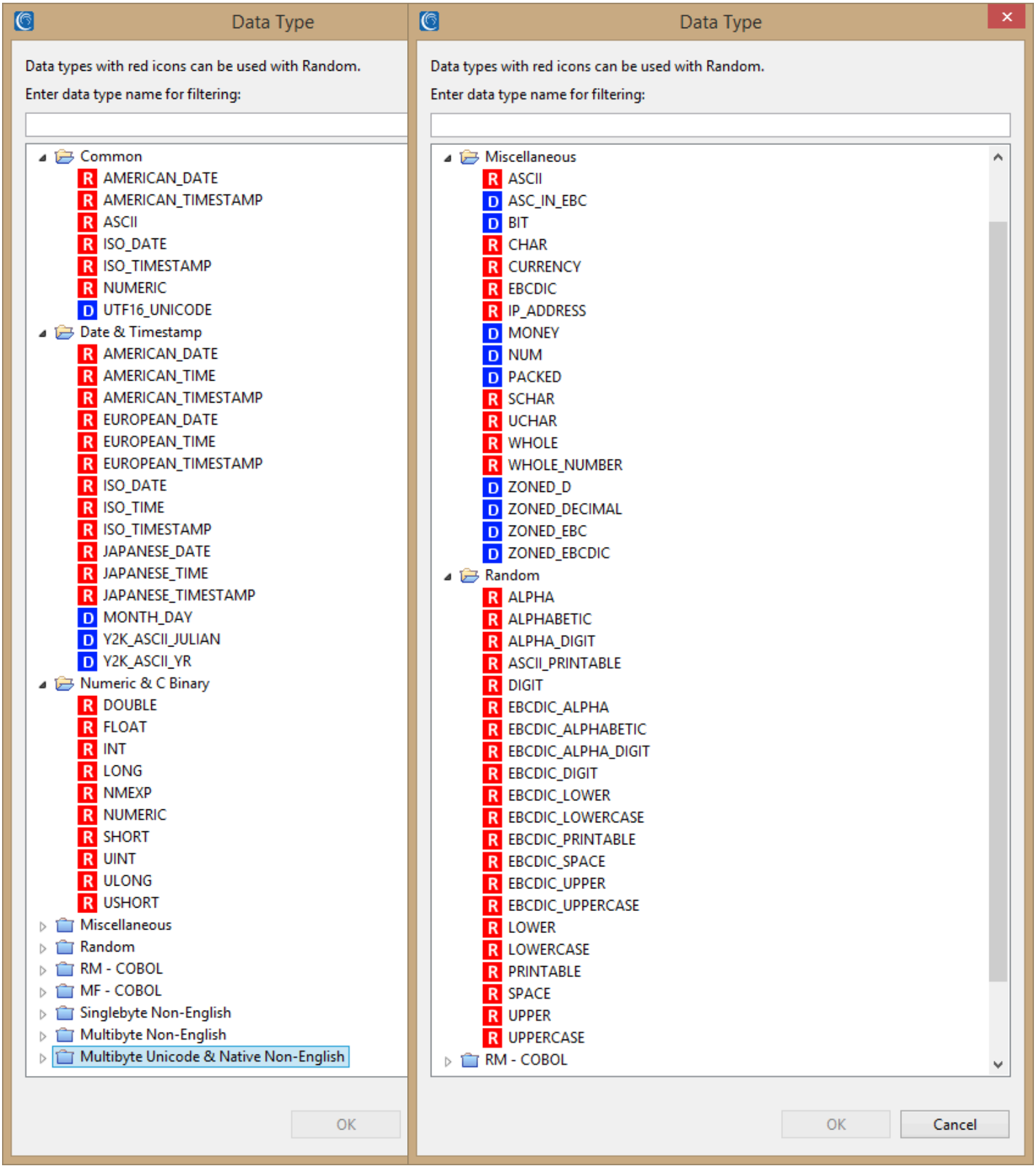
- Includes or omits input or output records using field-to-field or field-to-constant comparisons
- Compares on any number of data fields, using standard and alternate collating sequences
- Sorts and/or reformats groups of selected records
- Matches two or more sorted or unsorted files on inner and outer join criteria using SQL-based condition syntax
- Skips a specified number of records, bytes, or a record header
- Processes a specified number of records or bytes, including a saved header
- Eliminates or saves records with duplicate keys

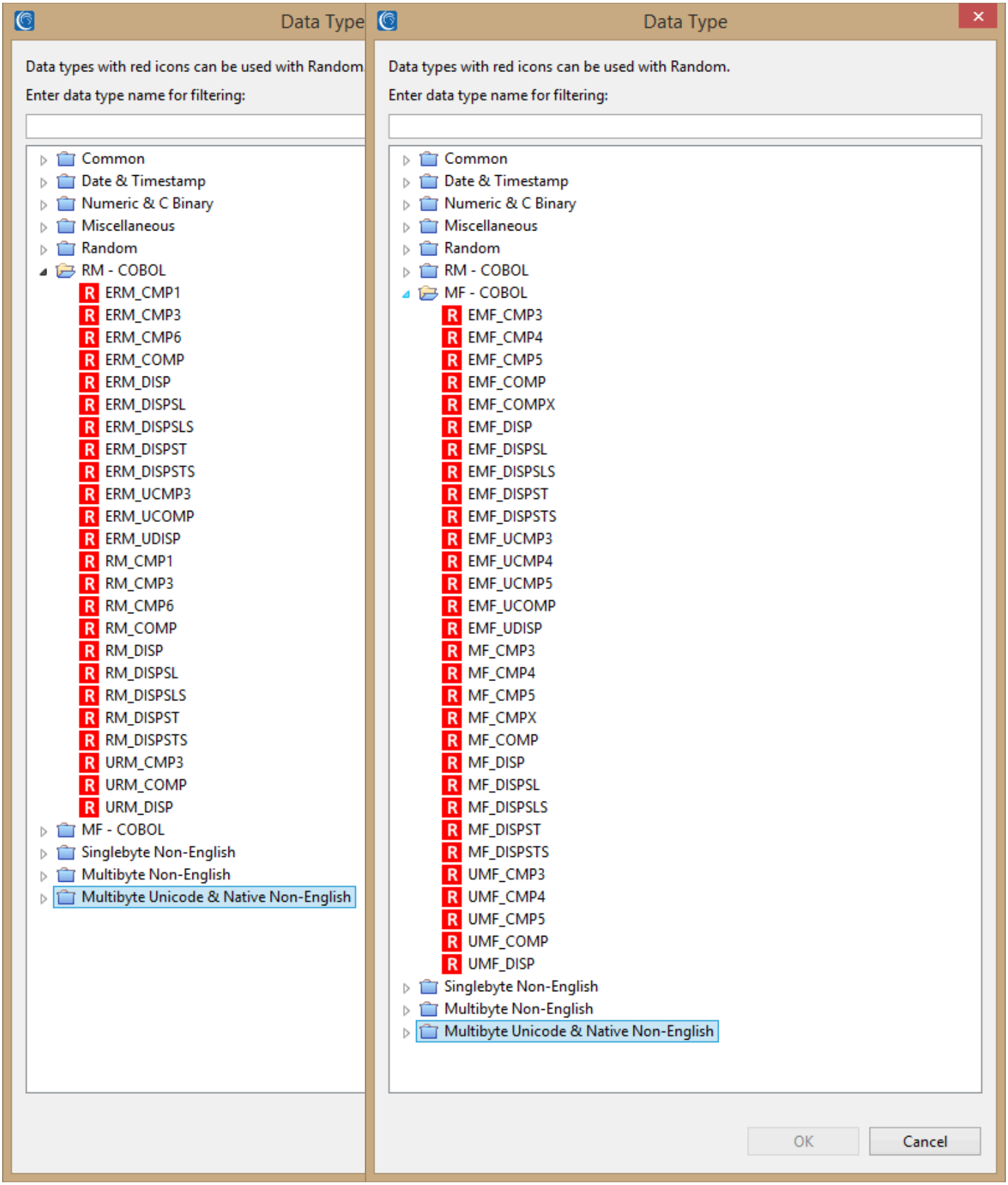
Sort Key Processing

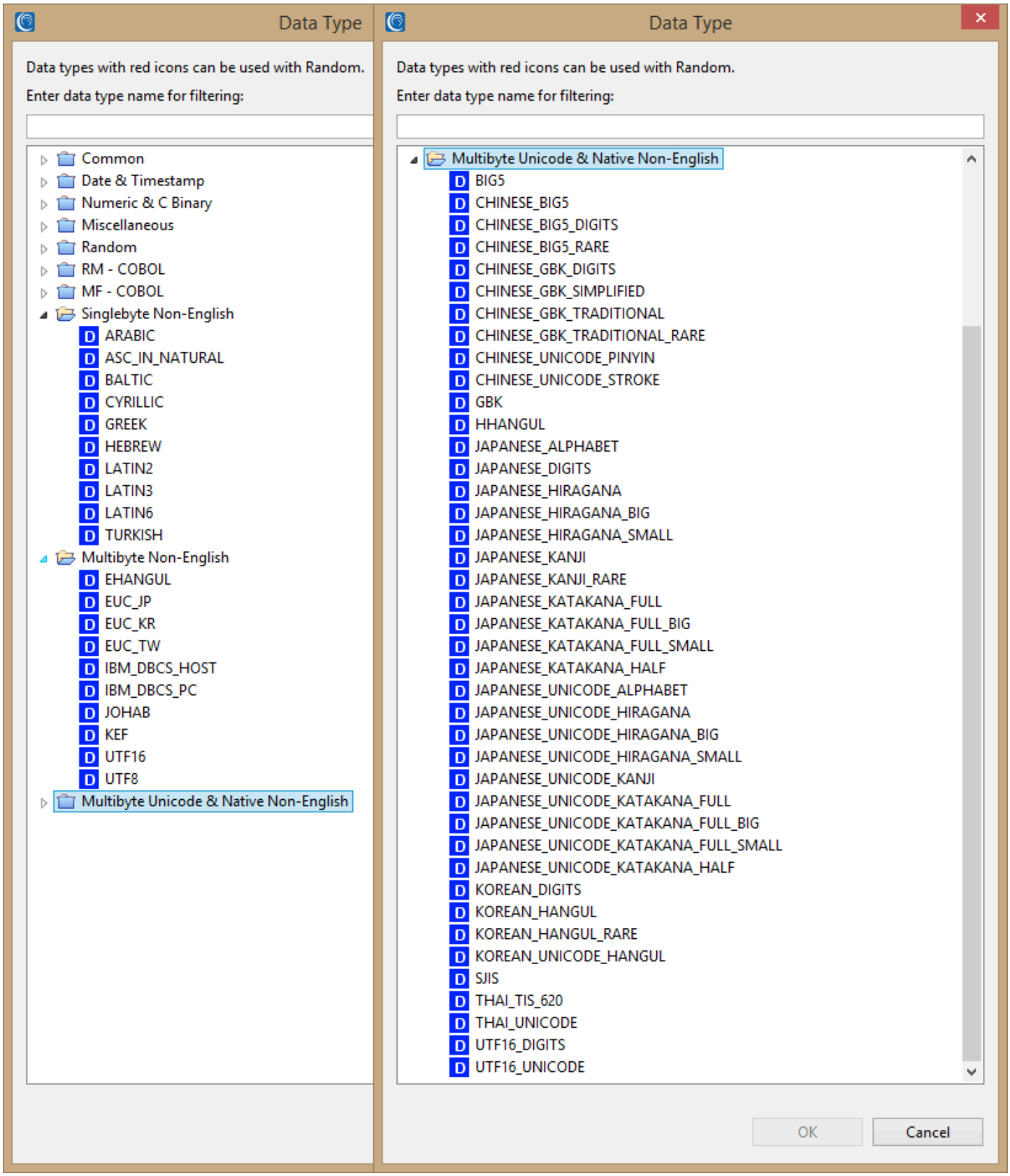
- Allows any number of key fields to be specified in ascending or descending order
- Supports any number of fields from 1 to 65,535 bytes in length
- Orders fields in fixed position or floating (on one or more delimiters)
- Supports numeric keys, including all C, FORTRAN, and COBOL data types
- Supports single and multi-byte character keys, including ASCII, EBCDIC, ASCII in EBCDIC sequence, Thai characters, and natural (locale-dependent) values
- Supports American, European, ISO and Japanese timestamps
- Supports Unicode and double-byte characters like Big5, EUC-TW, UTF16, and SJIS
- Allows left or right alignment and case shifting of character keys
- Accepts user compare procedures for multi-byte, encrypted and other special data
- Performs record sequence checking
- Maintains input record order (stability) on duplicate keys
- Controls treatment of null fields when specifying floating (character separated) keys
- Collates (and converts between many of) the following data type format groups

Form Group	Form Type
0	Alphabetic
1	Numeric
2	Date
3	Time
4	Timestamp
5	Reserved
6	Unicode

- Collates, converts, masks, formats, cleanses and otherwise operates on all the data types below. In many cases, the types can also be randomly generated for use as test data.







Record Reformatting

- Inserts, removes, resizes, and reorders fields within records
- Defines new fields through the use of various field-level functions
- Converts data in fields from one format to another using internal conversion
- Maps common fields from differently formatted input files to a uniform sort record
- Join any fields from several files into an output record, usually based on a condition
- Changes record layouts within, and converts between, different file formats, including: Delimited, Fixed, Line Sequential, Record Sequential, Variable Sequential, Blocked, Microsoft Comma Separated Values (CSV), ACUCOBOL Vision, MF I-SAM, MFVL, Unisys VBF, VSAM (via UniKix MBM), Extended Log Format, JSON, LDIF, and XML
- Maps processed records to many differently formatted output file, table and report targets
- Transposes (pivots) rows to columns and columns to rows
- Writes multiple record formats to the same file for complex report requirements
- Performs mathematical operations and functions on field data (including aggregate data) to generate new output field values

Field Reformatting & Validation

- Retrieves and re-maps values from multi-dimensional, tab-delimited lookup files on the basis of equal or conditional matches (suitable for Slowly Changing Dimensions)
- Creates and processes sub-strings of original field contents, where you can specify a positive or negative offset (from the left or right, respectively, of the source field) and a number of bytes to be contained in the sub-string
- Finds a user-specified text string in a given field, replaces all occurrences of it with a different user-specified string on output, and displays a set number of repeating strings
- Recognizes byte order marks and supports field-level big and little endian conversions
- Analyzes fields to display the offset number for the specified occurrence of a string
- Manipulates and displays literal values with input data inside field statements for use in value derivations, functions, conditions, cross calculations, and reporting
- Aligns desired field contents to either the left or right of the target inrec or output field, where any leading or trailing fill characters from the source are moved to the opposite side of the string
- Supports Perl Compatible Regular Expressions (PCRE), and pattern matching for find/replace operations
- Uses C-style “iscompare” functions to validate contents at the field level (for example, to determine if all field characters are printable or whether a value “ismissing”), which can also be used for record-filtering via /INCLUDE and /OMIT statements
- Supports custom-defined data masks and field layout templates to structure composite elements in new (e.g., master data format) field formats for mapping and validating data
- Supports Boost date and time libraries for data format masking and interval calculations
- Supports custom, user-written field-level transformation libraries, and documents an examples of data cleansing routines from Melissa Data and Trillium

Field Masking

- Encrypts with user-specified functions or these built-in libraries: 3DES, AES-128 and 256 bit (including format preserving), GPG, and FIPS-compliant OpenSSL
- De- and re-identifies with built-in ASCII bit scrambling, binary encoding, or substitute tokens
- Hashes with 128 and 256-bit functions
- Redacts with character masking or trimming, or conditional removal of columns, values or rows
- Pseudonymizes with scrambled substitutes of original values or randomly selected set lookups
- Randomly generates new field values for a given data type or randomly selects set file values
- Adds random noise (blur) to date and age values
- Anonymizes or generalizes quasi-identifying values by bucketing them into larger categories
- Manipulates field values with string functions or logical expressions
- Supports development and invocation of custom (user-created) field masking functions

Record Aggregation

- Consolidates records with equal keys into unique records, while totaling, averaging, or counting values in specified fields, including derived (cross-calculated) fields
- Produces maximum, minimum, average, standard deviation, sum, and count fields
- Displays running summary value(s) up to a break (accumulating aggregates)
- Breaks on compound conditions
- Allows multiple levels of summary fields in the same report
- Re-maps summary fields into a new format, allowing relational tables
- Ranks data through a running count of descending numeric values
- Writes detail and summary records to the same output file for structured reports
- Supports multiplication and ranking
- Calculates aggregate values within sliding range windows
- Uses fuzzy lookup logic for trend reporting and predictive analytics
- Supports Boost quick_stats functions for linear regression (trend line) analysis

Licensing Information

IRI and its expert agents around the world license **CoSort** for perpetual use on individual computer systems. Maintenance (technical support and site-specific software updates) services are provided free of charge during the first year after installation. Subsequent annual maintenance is usually offered at or below 20% of the base license fee, and 24/7 technical support is available as a premium option.

CoSort license fees for Unix systems are based on specific machine make and model numbers. CoSort license fees for x86 Linux and Windows are based on RAM. In either case, additional charges for multiple CPUs and cores are assessed to reflect the performance gains from multi-threaded operations.

License fee discounts apply for multiple copies of CoSort at the same installation, and for runtime integration and redistribution (for ISVs only). IRI is generous with credit for hardware upgrades and migrations, and provides for no- or low-cost failover (disaster recovery) licenses.

CoSort is also included in, and available on a subscription basis with, the IRI Voracity data management platform.

U.S. educational and 501c(3) nonprofit institutions qualify for a 10% license fee discount and government agencies will find CoSort on the GSA schedule.

A confidential license fee quotation and a free 30-day trial are available from your IRI agent, pursuant to a non-disclosure agreement.

A free, 30-day trial period is offered prior to licensing. A non-disclosure agreement must be completed online or sent signed to an IRI representative.

Professional Services

An IRI Professional services engagement allows you to leverage more than 100 collective years of IT and integrate data handling experience. Examples of IRI professional services engagements involve:

- Big data preparation - packaging, protection, and provisioning structured and unstructured data sets for BI within existing DBs and files systems (and without the skills gap, high cost, and failure rate associated with Hadoop or ELT appliances)
- Data masking - profiling, de-identification, re-ID risk scoring, and other services to aid data loss prevention, data governance, and data privacy law compliance efforts
- Data replication and federation - acquiring, re-mapping, and creating virtualized views
- Database migration - mapping table data and relationships to new versions on platforms
- Legacy data conversion - reformatting LDIF, XML, and COBOL index files (e.g., Vision, MF-ISAM), multi-byte character sets, most mainframe data types, and endian conditions
- Master data management - value and format definition, quality validation, and security
- Program replacement - translating cryptic and inefficient SQL, 3GL, ETL, and shell procedures into IRI's simple, portable, open-text 4GL scripts
- Test data management - end-to-end services from needs definition through data generation and target persistence (without using production data)

Company Background

Innovative Routines International (IRI), Inc. is an independent software vendor (ISV) specializing in data management and protection. Better known as "The CoSort Company," IRI was founded in 1978 as a pioneer and market leader in the "open systems" sort industry.

As data volumes have grown, so has IRI. More than 40 years later, IRI and its clientele continue to partner in the creation, improvement, and expansion of an integrated product line focused on:

Data Management - Data Movement, Data Integration, Migration, and BI / Wrangling
and
Data Protection - PII/PHI Data Discovery and Masking, Risk Scoring, and Safe Test Data

IRI continues to satisfy its users worldwide through an ongoing dedication to speed, ease, versatility, and value in the product line, and to agility, innovation, responsiveness, and quality in its services.

INNOVATIVE ROUTINES INTERNATIONAL (IRI), INC.

Suite 303, Atlantis Center
2194 Highway A1A
Melbourne, FL 323937-4932 USA
Phone | +1.321.777.8889
www.iri.com/cosort