

More Than a Sort: The Business Logic in Your Control Cards

What DFSORT, ICETOOL and Syncsort statements really encode, why inventories miss it, and how to find and grade it before migration

Bill Ahlbrandt

President and CEO, Ahlbrandt Software

Abstract

Application inventories count sort steps as utility calls. A lot of them are really programs. Between 2003 and 2010 IBM added conditional reformatting, find-and-replace, group processing, variable-field parsing, two-file joins and date arithmetic to DFSORT in a series of service updates, and Syncsort MFX has similar features. A deck of twenty control statements can now select, classify, compute, join, aggregate and report, with its business rules written as column positions and hex constants. In this paper I trace how the sort turned into a declarative data-transformation language, and read an invented (but syntactically faithful) example as the specification it really is. I also explain why this logic slips past code scanners, documentation and ownership, and lay out a scoring rubric and an inventory method, so a migration estimate, a test plan or an AI conversion can start from what the control statements actually do.

What I found

- The features that turn a sort into an application came in **service updates** between releases: arithmetic and SPLICE in 2003, IFTHEN and OVERLAY in 2004, PARSE in 2006, FINDREP and WHEN=GROUP in 2008, JOINKEYS in 2009, date arithmetic in 2010.^{1,2,3,4,5,6}
- One short deck can hold selection windows, classification thresholds, sign conventions, due-date rules, aggregation keys and report totals. **Calling that “one sort step” undercounts it.**
- This logic usually lives in control-card libraries, PROCs and PARMLIB members. Operations often maintains it, and it's **invisible to tools that only parse programs.**
- ICETOOL has 17 operators;⁷ one widely used migration runtime documents six.⁸ Check coverage **against your inventory.** Don't assume it.
- A simple rubric, applied to every statement set, sorts the plain sorts from the embedded applications and gives your estimates, test plans and conversion scope something to stand on.

1. The step everyone counts as a utility

Every mainframe application inventory has a line for sort. It's usually a count: so many steps execute PGM=SORT, ICEMAN, SYNCSORT or ICETOOL, and each one gets classed as a system utility along with the copies and deletes. The programs are measured in lines of COBOL, and there's plenty of that. One vendor-sponsored survey in 2022 put it at more than 800 billion lines in daily use.⁹ Migration estimates get built from those two numbers, and the second one dominates.

The assumption behind the first number is that a sort step sorts. A lot of them do: SORT FIELDS=(1,10,CH,A) and nothing else. But the language those steps are written in grew a great deal in the decade after 2000, and it was all documented in the open, in user guides showing how to join, reformat, summarize and report without writing a program. IBM's DFSORT team observed that the product's customers "seem to come in two varieties": those who use its features "often in ingenious ways", and those who use it in the most basic way.¹⁰ An inventory that counts steps treats the first kind as if it were the second.

I've spent more than thirty years supporting sort, and I'd argue that sort control statements are an application layer in their own right. The business rules in them can be found and graded without a huge effort, whichever way you migrate.

2. How a sort became a language

DFSORT's control statements have always done more than put records in order. INCLUDE and OMIT select, INREC and OUTREC reshape, SUM aggregates. The big growth came late, though, and in small packages. IBM's own summary of changes lists ICETOOL, the multi-operator front end, from Release 11.1 in February 1991, OUTFIL processing in the Release 13 base of May 1995, two-digit-year formats with a century window in Release 13 service, and symbols for fields and constants (SYMNAMES) in Release 14 in September 1998.¹¹ From 2003 on, the additions came as PTFs, each with its own user guide written by the DFSORT team (Exhibit 1).

Exhibit 1. When the transformation features arrived in DFSORT

Date	Release or PTF	Features added (selection)
Feb 1991	Release 11.1	ICETOOL multi-operator utility
May 1995	Release 13	OUTFIL processing; later, Y2x formats and the Y2PAST century window
Sep 1998	Release 14	Symbols for fields and constants (SYMNAMES)
Mar 2002	Release 14 PTFs	OUTFIL FTOV/VTOF, REMOVECC; date/time formats; SELECT FIRSTDUP/LASTDUP
Feb 2003	UQ90053	Arithmetic in reformatting; ICETOOL SPLICE; OUTFIL SAMPLE, REPEAT, SPLITBY
Dec 2004	UQ95214/UQ95213	IFTHEN (WHEN=INIT, WHEN=(logexp), ANY, NONE), OVERLAY, BUILD
Apr 2006	UK90007/UK90006	PARSE and %nn fields; JFY, SQZ; relative dates (DATEn±r); numeric tests; SPLIT1R
Jul 2008	UK90013	FINDREP; IFTHEN WHEN=GROUP (PUSH, ID, SEQ); sorting between headers and trailers
Nov 2009	UK51706/UK51707	JOINKEYS, JOIN, REFORMAT (paired and unpaired); TOJUL, TOGREG, WEEKDAY
Oct 2010	UK90025/UK90026	ADDDAYS, SUBDAYS, DATEDIFF, LASTDAYx date arithmetic; IFTRAIL; RESIZE
2013–19	z/OS V2R1–V2R4	1,000 parsed fields (2013); WEEKNUM, AGE (2015); regular expressions (2019)

Sources: IBM, DFSORT Summary of Changes by Release; IBM user guides for the PTFs listed.^{11, 1, 2, 3, 4, 5, 6} Where a PTF pair is shown, the first one applied to z/OS DFSORT V1R5 or V1R10 and the second to the other supported level.

That timing matters when you take inventory. A deck written in 2012 may use features that didn't exist when the people maintaining it were trained. IBM's companion publications, *Smart DFSORT Tricks* and *DFSORT: Beyond Sorting*, collected worked examples of joins, group processing, date arithmetic and reports built entirely from control statements.^{12, 10} People copied them a lot, which is one reason these techniques turn up in production decks.

Syncsort MFX, now a Precisely product, has similar features under mostly the same statement names, including a join that can keep unpaired records from either or both files,¹³ and SyncTool, its counterpart to ICETOOL.¹⁴ IBM's December 2004 user guide listed easier migration from competing sort products as one of the aims of those PTFs.² So shops on either product have had the same toolkit for well over a decade, and you can find the same statements in either kind of shop.

3. Reading a control deck as a specification

Exhibit 2 is an invented deck of 24 lines, written to IBM's documented syntax, of the kind that builds up around a nightly feed. It reads a fixed-length transaction file: record type in columns 1–2, account in 3–12, transaction date in 21–28, a debit or refund code in 40, a six-byte packed amount in minor units in 41–46 and a currency code in 47–49. It writes no ordinary sorted output at all. What it produces is an extract and a report.

The toolkit can do more than the example shows. There are lookups through CHANGE tables, propagation of header values to detail records with WHEN=GROUP, parsing of comma-separated input with PARSE, and matching of two files with JOINKEYS, where picking JOIN UNPAIRED, F1 or JOIN UNPAIRED, F1, ONLY decides whether you get an outer join or a list of exceptions.^{15, 12} ICETOOL adds operators that answer business questions directly: SELECT for first, last or duplicate occurrences, OCCUR for frequency counts, STATS and RANGE for totals and bounds, and SPLICE for joins by key.⁷

Exhibit 2. A 24-line deck and the business rules in it (invented example)

Lines	Control statements	Business rule encoded
1	* DAILY CARD FEE FEED (INVENTED EXAMPLE)	Just a comment. Often it's the only documentation a deck like this has.
2-4	INCLUDE COND=(1,2,CH,EQ,C'TX',AND, (40,1,CH,EQ,C'D',OR,40,1,CH,EQ,C'R'),AND, 21,8,CH,GE,DATE1-35)	Only transaction records; only debits and refunds (every other type is dropped without a trace); only the last 35 days, counted from the run date . The business date doesn't come into it, so a rerun next week selects different records.
5-7	INREC IFTHEN=(WHEN=INIT, OVERLAY=(201:21,8,Y4T,ADDDAYS,+30,TOGREG=Y4T, 209:C'S')),	Payment is due 30 calendar days after the transaction, with no business-day adjustment. Every record starts out as class S (standard).
8-10	IFTHEN=(WHEN=(41,6,PD,GT,+1000000,AND, 47,3,CH,NE,C'EUR'), OVERLAY=(209:C'H'),HIT=NEXT),	High value means more than 10,000.00 in two-decimal minor units, whatever the currency, except euro. The threshold and the exception are both hard-coded.
11-13	IFTHEN=(WHEN=(40,1,CH,EQ,C'R'), OVERLAY=(41:41,6,PD,MUL,-1, T0=PD,LENGTH=6))	Refunds are carried as negative amounts. Because of HIT=NEXT, a large refund also gets classed as high value, since the test runs before the sign is flipped.
14-15	SORT FIELDS=(3,10,CH,A,21,8,CH,A,209,1,CH,A) SUM FIELDS=(41,6,PD)	Net one record per account, day and class. Amounts in different currencies get added together, because the deck assumes one currency per account. SUM overflow is controlled by options set elsewhere.
16-19	OUTFIL FNames=HIGHVAL,INCLUDE=(209,1,CH,EQ,C'H'), BUILD=(3,10,C',',201,8,C',', 41,6,PD,EDIT=(SIIIIIIIT.TT), SIGNS=(+,-))	A comma-separated high-value extract for a downstream consumer: account, due date, signed net amount. That file layout is an interface contract.
20-24	OUTFIL FNames=FEERPT,REMOVECC,NODETAIL, SECTIONS=(3,10, TRAILER3=(3,10,2X, TOT=(41,6,PD,EDIT=(SII,III,III,IIT.TT), SIGNS=(+,-)))	A control report with one total line per account, net of refunds, and no detail lines. Somebody reconciles against this every morning.

Made up for illustration. It isn't taken from any shop. Syntax follows the z/OS DFSORT Application Programming Guide and IBM's published examples of relative dates, ADDDAYS, IFTHEN, SUM and OUTFIL reports.^{15, 10, 12}

Exhibit 2 has seven groups of rules in it, and none of them is sorting. Three are easy to get wrong in any re-implementation. First, DATE1-35 is evaluated against the date the job runs, so the output depends on the calendar as well as the data. A parallel run on a different day won't match. Second, IFTHEN clauses are tested in order, and

HIT=NEXT lets a record satisfy more than one, so the order of lines 8 to 13 is itself a rule. Third, what SUM does when a total overflows its field, and which of several equal-keyed records survives, are decided by run-time options and installation defaults. Nothing in the deck tells you.^{15, 16} If all you read is lines 14 and 15, you see a sort.

4. Why nobody sees this logic

To the tools, it's data. To a JCL parser, a control deck is an in-stream data set or a member of a partitioned library named on a SYSIN DD statement. Tools that parse programs and JCL will see the step, its program name and its data sets. They'll understand the statements only if they have a sort-statement parser built in, and whether a given analysis product does is a question to put to its vendor in writing.

It lives in several places. DFSORT reads statements from SYSIN, from SORTCNTL and from DFSPARM, which can override options for a step without touching its deck;¹⁵ Syncsort uses \$ORTPARM the same way. ICETOOL reads TOOLIN and a separate xxxxCNTL data set for each USING(xxxx). Installation defaults, including how equal keys and SUM overflow are handled, sit in ICEPRMxx PARMLIB members.¹⁶ COBOL programs that use the SORT verb can take statements from a control data set, by default IGZSRTCD.¹⁷ Some shops generate control statements in an earlier job step, so the deck that runs doesn't exist until the night it runs.

Operations owns it. This point and the next one come from practice. I haven't measured them. At a lot of shops the control-card libraries belong to production control or operations analysts, and the application teams whose programs surround them never touch them. Changes go through operational change processes and rarely make it into design documents. When the people who wrote the decks retire, the knowledge goes with them. In one 2025 vendor-sponsored survey, 70 percent of organizations said they had trouble finding mainframe skills.¹⁸

It looks finished. A declarative deck that has run for ten years rarely fails in a way anyone notices. It just keeps applying the rules of the year it was written.

5. Grading control statements: a rubric

Grading is for triage. It won't be precise, and it doesn't need to be. The idea is to separate the decks that sort from the decks that decide. Exhibit 3 gives points by feature family. The weights are mine and they're only illustrative, so calibrate them against a sample of decks whose logic you've already documented.

Exhibit 3. A scoring rubric for sort control statements (illustrative weights)

Feature family	Statements and operands that score	Points	Exh. 2
Selection	INCLUDE/OMIT: simple test 1; AND/OR or substring 2; relative dates, numeric tests or regular expressions 3	0-3	3
Reformatting	BUILD/OVERLAY 1; CHANGE, FINDREP, PARSE, JFY/SQZ, edit masks, conversions 2	0-2	2
Conditional logic	IFTHEN WHEN=(logexp) 2; +1 for HIT=NEXT, WHEN=GROUP or over three clauses	0-3	3
Computation	Arithmetic 1; date arithmetic (ADDDAYS, DATEDIFF, TOGREG and similar) 1	0-2	2
Aggregation	SUM 1; SECTIONS/TRAILERn with TOT or COUNT 1; STATS, RANGE, OCCUR 1	0-3	2
Multiple inputs	JOINKEYS 2, or 3 with UNPAIRED or FILL; ICETOOL SPLICE 2	0-3	0
Multiple outputs	Each further OUTFIL, SAVE, SPLIT/SPLITBY: 1 each	0-2	1
Environment	DATEn constants, symbols, DFSPARM/\$ORTPARM overrides, reliance on EQUALS or OVFLD defaults: 1 each	0-2	2
Total	Tiers: 0-2 mechanical; 3-6 transforming; 7-11 business logic; 12+ embedded application	0-20	15

My rubric. Points and tier boundaries are illustrative. Calibrate them for your own shop.

Scored this way, Exhibit 2 gets 15 out of a possible 20. That's an embedded application in 24 lines. A plain SORT FIELDS deck scores zero, and a copy with a simple INCLUDE scores one. What you really want is the spread across the whole shop. It tells you how many decks need a business analyst, how many only need a compatibility check, and where to put test effort first.

6. What this means for estimates, testing and conversion

Estimates. If every sort step is priced as a utility and some of them are applications, the estimate will be wrong, and you know which direction. The cost is in reproducing the rules. The sorting is the easy part. If your migration route rewrites sort steps as code, each high-scoring deck becomes a small program to specify, write and test. If the route keeps the statements and runs them on a compatible engine, the same deck becomes a compatibility requirement. Either way it has to be counted.

Testing. Each IFTHEN clause, INCLUDE branch, JOIN option and OUTFIL destination is a path your test data has to exercise. Exhibit 2 has four combinations of class and sign alone, before you even get to the date window. Relative date constants mean comparison runs have to be pinned to the same run date. The general case for proving identical output, including equal-key order, is in *Same Input, Same Output*, the second paper in this series.

AI-assisted conversion. When a converter runs into PGM=SORT it can do one of three things: call a sort engine on the target, translate the statements into generated code, or leave the step to the customer. The first is only as good as the engine's coverage. For example, one widely used runtime documents an ICETOOL with six operators (COPY, SORT, SELECT, SPLICE, COUNT and OCCUR)⁸ against IBM's 17.⁷ The second turns a reviewed declarative spec into custom code whose defaults for signs, collation and equal keys aren't the same as the original's. I go into both in *What the AI Converters Do With Your Sort Steps*. Whichever one you get, the conversion can only be as complete as the inventory that fed it.

7. Finding them: an inventory method

Every step here uses libraries and records your shop already has. You need a parser for sort statements (a pretty modest piece of software, since the grammar is well documented) and access to the libraries operations owns.

- 1 Gather the sources.** Collect JCL, PROCs, every library and data set listed in Exhibit 4, and the source of programs that call sort.
- 2 Work out what actually runs.** Expand PROCs, apply overrides and substitute JCL and system symbols, so each step is paired with the statements it really gets. Flag any step whose statements are generated by an earlier step.
- 3 Cast a wide net for sort steps.** Include SORT, ICEMAN, SYNC SORT, ICETOOL, SYNCTOOL and site aliases, JOINKEYS subtask data sets (JNF1CNTL, JNF2CNTL), and COBOL SORT and MERGE verbs with IGZSRTCD.
- 4 Parse, score and link.** Normalize continuations and comments, score each deck with the rubric, and link it to the record layouts of its input and output data sets so column positions can be read as field names.
- 5 Check it against what ran.** Match the inventory to DFSORT SMF type 16 records, which log each sort run,¹⁹ to separate the decks that run from the ones that just sit there, and to see how often each one runs.
- 6 Assign owners and write it down.** For every deck in the top two tiers, write the business rules in plain language, name an owner, and add test cases for each branch. That's what an estimate, a test plan or a converter needs.

Exhibit 4. Inventory checklist: where sort logic hides

Location	What to look for	Typical owner
JCL, PROCs and symbols	SYSIN DD * decks; PROC symbolics and SYMNAMES data sets that supply positions or constants	Development or production control
Control-card libraries	Members named on SYSIN, SORTCNTL, TOOLIN and xxxxCNTL DDs; shared decks reused by lots of jobs	Operations
Override data sets	DFSPARM (DFSORT) and \$ORTPARM (Syncsort) statements that change options per step	Operations or systems programming
PARMLIB	ICEPRMxx installation defaults such as EQUALS, OVFL0 and Y2PAST, and their Syncsort equivalents	Systems programming
Programs	COBOL SORT/MERGE with IGZSRCD; calls to sort with parameter lists built at run time	Development
Generated decks	Steps that write control statements a later step uses	Whoever wrote the generator

Sources: IBM DFSORT and Enterprise COBOL documentation; Precisely Syncsort MFX documentation.^{15, 16, 17, 13}

8. Bottom line

- **Count decks, and grade them.** A step's size depends on what its statements do, and they can do as much as a small program. Counting steps tells you very little.
- **The toolkit is mature.** Most of the transformation features have been in DFSORT since 2010 or earlier, and Syncsort MFX has similar ones. Expect to find them in production decks.
- **Ownership decides who sees it.** Logic kept in operations libraries, override data sets and PARMLIB members gets missed by analysis that only looks at programs, unless somebody goes looking for it.
- **Grade before you estimate.** Run a rubric across the whole shop and an unknown becomes a spread you can look at. It points analysts, testers and converters at the decks that make decisions. Whether you pick a replacement sort engine or a rewrite, those rules have to be reproduced, and the only way to check that is against an inventory that found them.

About the author

Bill Ahlbrandt is President and CEO of Ahlbrandt Software, which he founded in 1993. Before that he was a systems programmer, Director of Information Security and Director of R&D at Acxiom. He has been working on mainframe sort and batch for more than thirty years, and his company's sort engine runs mainframe sort jobs on Linux and other platforms for banks, insurers and government agencies.

Full disclosure: my company sells sort software for platforms other than the mainframe, and it runs mainframe sort control statements. The example in this paper is made up, and you can use the rubric whichever migration route you take.

References

1. F. L. Yaeger, *User Guide for DFSORT PTF UQ90053*, IBM, February 2003.
2. F. L. Yaeger, *User Guide for DFSORT PTFs UQ95214 and UQ95213*, IBM, December 2004.
3. F. L. Yaeger, *User Guide for DFSORT PTFs UK90007 and UK90006*, IBM, April 2006.
4. IBM, *User Guide for DFSORT PTF UK90013*, July 2008.
5. IBM, *User Guide for DFSORT PTFs UK51706 and UK51707*, November 2009.
6. IBM, *User Guide for DFSORT PTFs UK90025 and UK90026*, October 2010.
7. F. L. Yaeger, *DFSORT: ICETOOL Mini-User Guide*, IBM, October 2010.
8. AWS, *AWS Mainframe Modernization User Guide*, "Sort Utilities" (SORT/SYNCSORT/ICEMAN, ICETOOL, MFSORT), accessed September 2026.

9. Micro Focus, COBOL market survey conducted by Vanson Bourne, 4 February 2022. Vendor-sponsored.
10. F. L. Yaeger, *DFSORT: Beyond Sorting*, IBM DFSORT Team, October 2010.
11. IBM, “DFSORT: Summary of Changes by Release”, DFSORT web site (IBM Support), covering Release 6 (1984) to z/OS V2R4 (September 2019).
12. IBM DFSORT Team, *Smart DFSORT Tricks*, October 2010.
13. Precisely, *Syncsort MFX 3.1 Programmer's Guide*, “How to Use the MFX Data Utility Features” (join processing, retaining unpaired records), 2023.
14. Precisely, “Implementing SYNCTOOL in Syncsort MFX”, customer support article, undated.
15. IBM, *z/OS DFSORT Application Programming Guide*, SC23-6878 (INCLUDE/OMIT, INREC IFTHE, SUM and EQUALS, OUTFIL, JOINKEYS; SYSIN, SORTCNTL and DFSPARM sources).
16. IBM, *z/OS DFSORT Installation and Customization*, SC23-6881 (ICEPRMxx PARMLIB members and installation defaults).
17. IBM, *Enterprise COBOL for z/OS Programming Guide*, “Controlling sort behavior” (SORT-CONTROL special register; IGZSRTCD control data set).
18. Kyndryl, State of Mainframe Modernization Survey, 9 September 2025 (n = 500). Vendor-sponsored.
19. IBM, DFSORT SMF type 16 record documentation (record counts, data sets and options per sort run).