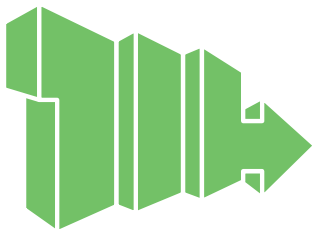


MAINFRAME MIGRATION ALLIANCE

Batch Applications—The Hidden Asset

Published: June 2006

For the latest information, please see www.microsoft.com/mainframe

Microsoft

FUJITSU

Contents

Introduction

A major part of the workload on mainframe computers consists of batch processing. A large mainframe often will run several thousand batch jobs every day. This "network" of jobs represents a business workflow with complex interrelations requiring careful scheduling and prioritizing to ensure that all batch jobs run in the correct order and meet strict deadlines. Consequently, sophisticated systems have evolved to manage and run batch workloads—systems that "feel" mainframe-specific, with no obvious parallels in Windows®-based computing. Approached with the idea of migrating batch applications to the Windows operating system, most IT managers react with, "How can that be possible?" From another angle, many other people assume modern applications no longer require batch processing—an opinion that is completely wrong.

The Windows operating system is quite capable of supporting the complex needs of batch applications as has been demonstrated by the many organizations that have migrated batch systems to the Windows platform. The continuing need for batch processing is illustrated by key Microsoft® products, such as BizTalk® Server, which support workflows very similar to those of mainframe batch systems.

There is tremendous value invested in batch applications that may seem locked in mainframe architecture. Taking advantage of new technologies on the mainframe, such as Web services, requires considerable effort and investment—more than most companies can afford. If this value can be placed in a more flexible, agile environment, such as the Windows operating system and the Microsoft .NET Framework, organizations have the opportunity to unlock the value to enhance business operations and provide new or better services to customers.

This white paper explores what is needed to migrate mainframe batch applications to a Windows platform by reviewing the different components that make up batch applications, considering the issues involved in migrating each component, and determining how those issues can be resolved with products available today.

This paper does not attempt to review all migration solutions and compare their strengths. The objective, rather, is to examine the primary aspects of migrating batch jobs to the Windows operating system and the .NET Framework and to consider whether adequate solutions from vendors such as Microsoft, Fujitsu, or a third-party group are available to support that migration. When you start seriously investigating the migration option, other vendors' offerings can be assessed and compared; the issues to be resolved will be the same.

The Critical Elements of Batch Systems

Overview

Batch systems are collections of programs that run without interaction with users and with minimal interaction with a computer operator, where “minimal” means no more than the computer operator setting parameters at the start of a batch job and changing tapes or other data storage devices when prompted. Many of today’s batch systems automate these tasks so there is no interaction with humans unless something goes wrong. Most mature mainframe systems rely on batch jobs to perform significant portions of the total application logic. The types of tasks undertaken include:

- Merging the day’s transactions into master files
- Sorting data files for optimal processing the following day
- Merging data from multiple locations
- Providing daily, weekly, monthly, and annual reports
- Issuing daily, weekly, and monthly bills or invoices to customers
- Performing daily, weekly, biweekly, and monthly payroll calculations
- Consolidating multiple orders into single shipments and invoices
- Printing checks
- Performing special mailings
- Applying interest to financial accounts
- Batching orders for transmission to another company
- Performing backups
- Archiving data
- Auditing transactions and systems

Some companies may be able to survive for a day or two without running batch systems, but most will find business operations degrading rapidly if the batch applications are not run.

Batch Components

The main components of the IBM mainframe batch ecosystem are:

- Job entry subsystem (JES)—queues and assigns jobs to initiators
- Initiator—responsible for running a job
- Job—one or more executable programs
- JCL—describes the job, including file name cross-references, executable commands, and control flow, and provides control parameters to the JES
- Scheduler—schedules job start times, codifies dependencies between jobs and their run-time requirements, and provides reporting and alerting capabilities
- Data files and databases

Longtime users of batch processing tend to under-appreciate the sophistication of the typical mainframe batch environment. System programmers spend considerable time deciding the number of initiators that can be created for batch processing without affecting the performance of online workloads. Job schedulers have thousands of stored parameters for documenting the event sequence requirements for thousands of jobs. The overall combination of the JCL, schedulers, and the jobs themselves represent a massive investment in constructed workflow by the organization’s IT staff. Re-creating this environment from scratch could represent hundreds of staff-years. Figure 1 gives a simplified view of these components.

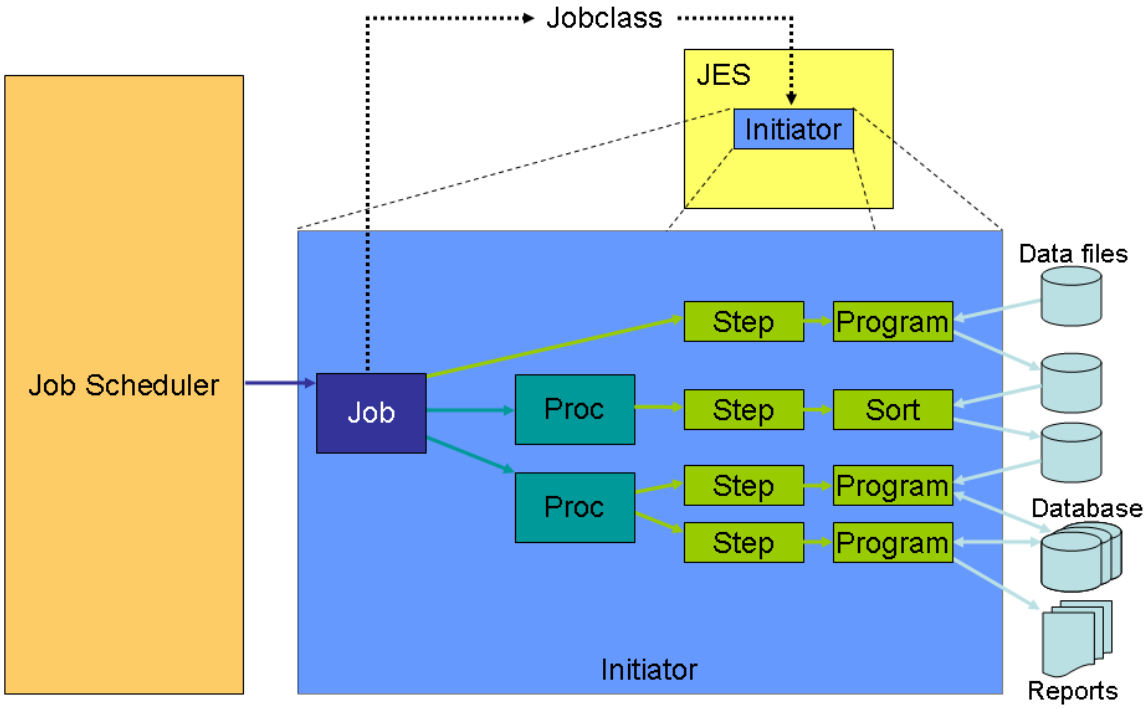


Figure 1. Schematic diagram of batch components

The processes illustrated in Figure 1 can be described as follows:

- The job scheduler determines when a job will run and starts the job.
- The job runs within an initiator—a set of machine resources such as memory and processor time. The mainframe operating system is configured for a limited number of initiators.
- Initiator assignment is determined by the JES (JES2 or JES3).
- Jobs consist of one or more steps, with one step for each main program to be run.
- The steps may be coded directly in the JCL that defines the job or in procedures (PROCs) invoked from the job, which is also written in JCL and stored in the libraries.
- Typically the program steps read and update files or databases and create reports.

Each of these elements is described in more detail in Appendix A. The following sections assume an understanding of these elements.

Exploring the Move to Windows

This paper discusses the key areas to be considered when migrating batch applications to the Windows operating system. Although the list of areas might appear daunting, when each area is divided into issues and these issues are tackled methodically, solutions become evident to organizations seeking the short-term and long-term benefits of migrating from mainframe computers. Here is a summary of the key areas:

1. Infrastructure—job classes, initiators, and schedulers
2. Application code—executable programs, such as COBOL programs or sort utilities
3. Application data—data files and databases

4. Script control—the JCL that drives the batch jobs

For the migration of batch applications from the mainframe to a Windows platform to be viable, the new environment must be able to offer equivalent solutions for each element of the mainframe batch components. Establishing the batch ecosystem on a Windows platform must also be sufficiently straightforward that the transition is not prohibitively expensive in time, money, or learning.

To make the move worthwhile, the migrated applications should be able to take advantage of features available in the new environment. For most organizations, emulating the mainframe environment will not be enough; access to the productivity features of Microsoft's line of server software and the Microsoft .NET Framework will be required also.

In this section we examine the issues involved in the migration of each component of batch applications and consider ways those issues can be addressed on the Windows platform.

1. Infrastructure

The mainframe operating structure is a collection of subsystems, each with a specific mission. The CICS or IMS-TM subsystems handle online transactions, the file subsystem manages data access, and the job entry subsystem (JES) manages batch processing. JES works cooperatively with the operating system to provide and manage hardware and other software services to the executing application commonly referred to as a "job." The schematic in Figure 2 provides a high-level view of this relationship.

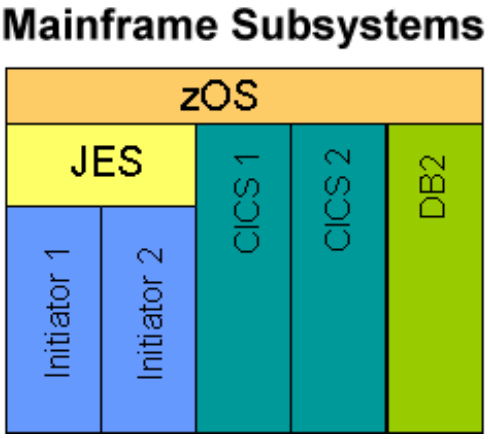


Figure 2. Mainframe subsystems

Main Issues

- Creating an equivalent to the mainframe batch ecosystem
- Determining whether the mainframe scheduling package also runs on Windows
- Preserving the prioritization and balancing functions provided by job classes and initiators

Create an Equivalent to the Mainframe Batch Ecosystem

Figure 3 illustrates how the Windows ecosystem can be viewed to have a structure similar to the mainframe system, with various subsystems spread across one or more servers.

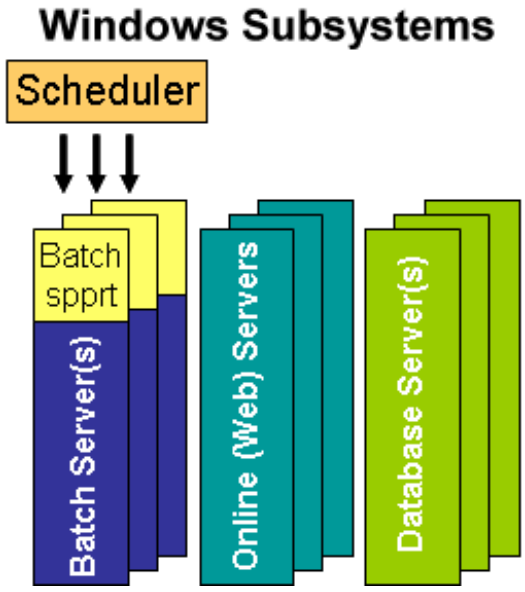


Figure 3. Windows subsystems can be spread over one or more servers

Here, the Windows ecosystem is represented as several independent servers. Windows also provides the option to run many processes on a single large server, in a manner similar to the mainframe. The combination of a scheduler and multiple servers provides a similar structure for queuing, resource allocation, and multiple-job sequencing. While this concept provides for a native environment for batch processing on the Windows platform, it does not satisfy many requirements for processing batch jobs that used to run on the mainframe. The following sections describe how add-on components can provide solutions for those requirements.

Determine Whether the Mainframe Scheduling Package Also Runs on Windows

Another key component of the batch processing infrastructure is job scheduling. Figure 4 illustrates the often complex scheduling relationships between batch jobs. This includes coordinating not only when particular jobs should be run but also includes what happens when jobs running on different systems succeed or fail. Most organizations use an IBM or third-party job scheduler to manage these processes.

These complex relationships are driven by business rules developed and refined over many years. The job scheduler database contains much valuable information that must be preserved when a migration is performed.

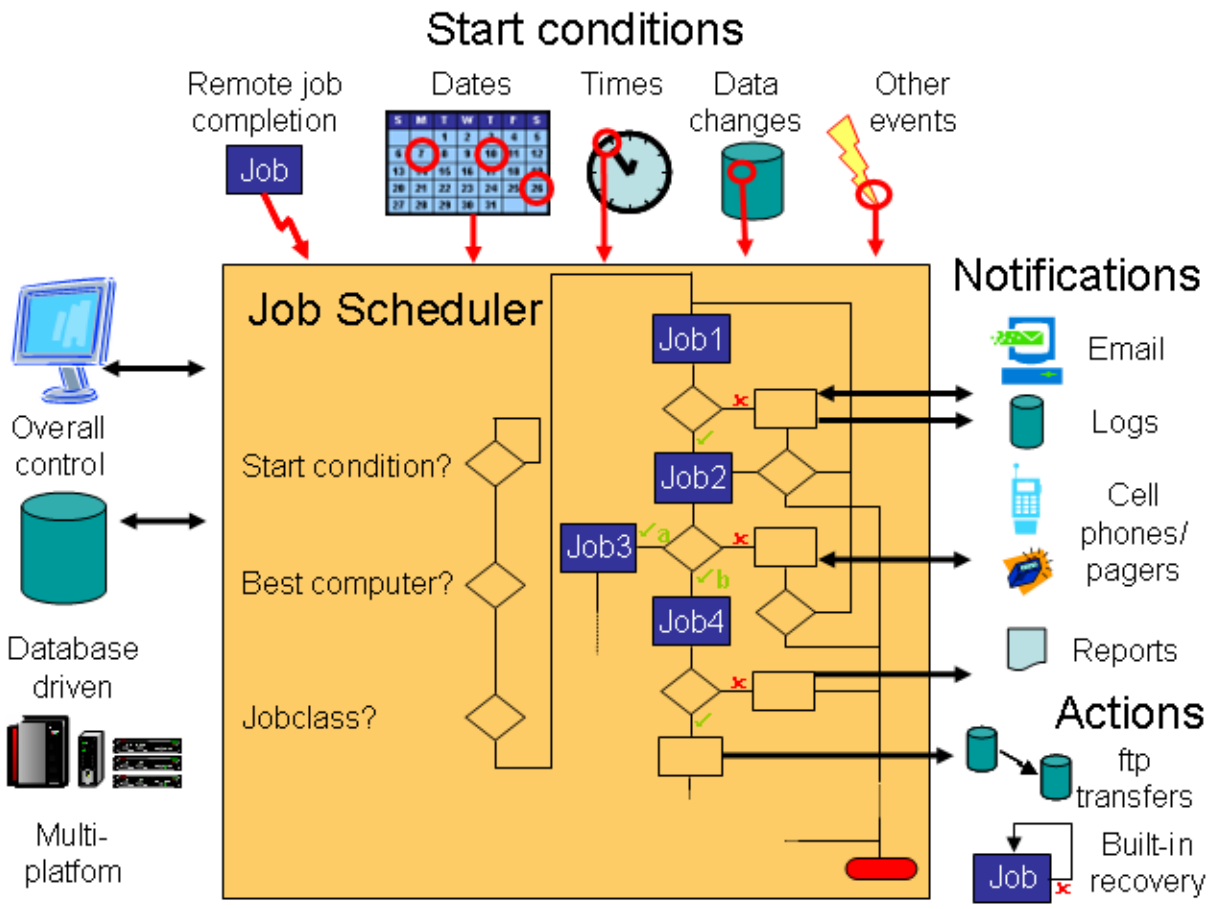


Figure 4. Overview of job scheduler functions

The main issue to consider before migration will be whether the mainframe scheduling package will run on Windows. Fortunately, most popular mainframe schedulers have a version for Windows, and these generally employ the same features as those employed on the mainframe.

Enterprise-grade schedulers are available from several vendors, including CA, BMC, Tidal Software, and Cybermation (now part of CA). These vendor products provide multiple-platform scheduling hosted on a Windows platform. Through the use of agents, schedulers can manage multiple-platform, multiple-server, and multiple-event combinations for the most demanding job schedules. If no Windows version exists for the current mainframe scheduler, Windows schedulers often have utilities to inherit the scheduling attributes stored on the mainframe.

To migrate scheduling information:

1. Select the job scheduler that provides the desired functionality in the Windows environment.
2. If possible, migrate the job schedule files from the mainframe to Windows or, if necessary, obtain printouts of the schedules and job success or failure information to reproduce in the new tool.

3. Perform test runs and review whether adjustments must be made due to changes in the execution times of the batch jobs.

Preserve Prioritization and Balancing Provided by Job Classes and Initiators

Organizations that run thousands of batch jobs every day have usually invested considerable effort in assigning job classes to batch jobs and determining how the job classes are distributed across initiators. The number of initiators set up and the number of machine resources assigned to each initiator are key ways of determining the amount of processing power allotted to different jobs, job classes, and the entire batch workload. Mapping job classes to initiators may have required considerable effort to achieve a balance that enables the organization's batch processing to be executed within the available time frame and with everything run in the correct order.

The knowledge embedded in job class structures can and should be carried forward to the Windows platform. Redoing this work in a new environment that uses different methods to prioritize batch jobs would be a daunting task. Therefore, the main issue to consider when migrating initiators and job classes is how to preserve the prioritization and balancing provided by job class and initiator relationships.

Windows does not have a construct that is equivalent to "initiator." Consequently, the choices are to work out how to use Windows utilities to prioritize and balance batch job execution or to find a JCL support package that provides a mechanism similar to initiators for prioritizing batch job execution.

Provided that it can be done efficiently, the best solution is to employ a JCL support package that will prioritize job classes similarly to the way the mainframe does.

For example, NeoBatch, a product developed by Microsoft Partner Fujitsu, enables system engineers to set up initiators and assign job classes just as they would on the mainframe. The only difference in the arrangement is that NeoBatch does not restrict the number of initiators that can be created and allows initiators to be set up on different processors in multiprocessor systems. The principles of using initiators to prioritize batch job execution is much the same, but system engineers now have additional options such as specifying the number of jobs in a given class that can run concurrently. These options are illustrated in Figure 5. (To compare this process with how JES uses initiators to manage batch job execution, see Figure 13. How JES uses initiators to manage job execution.)

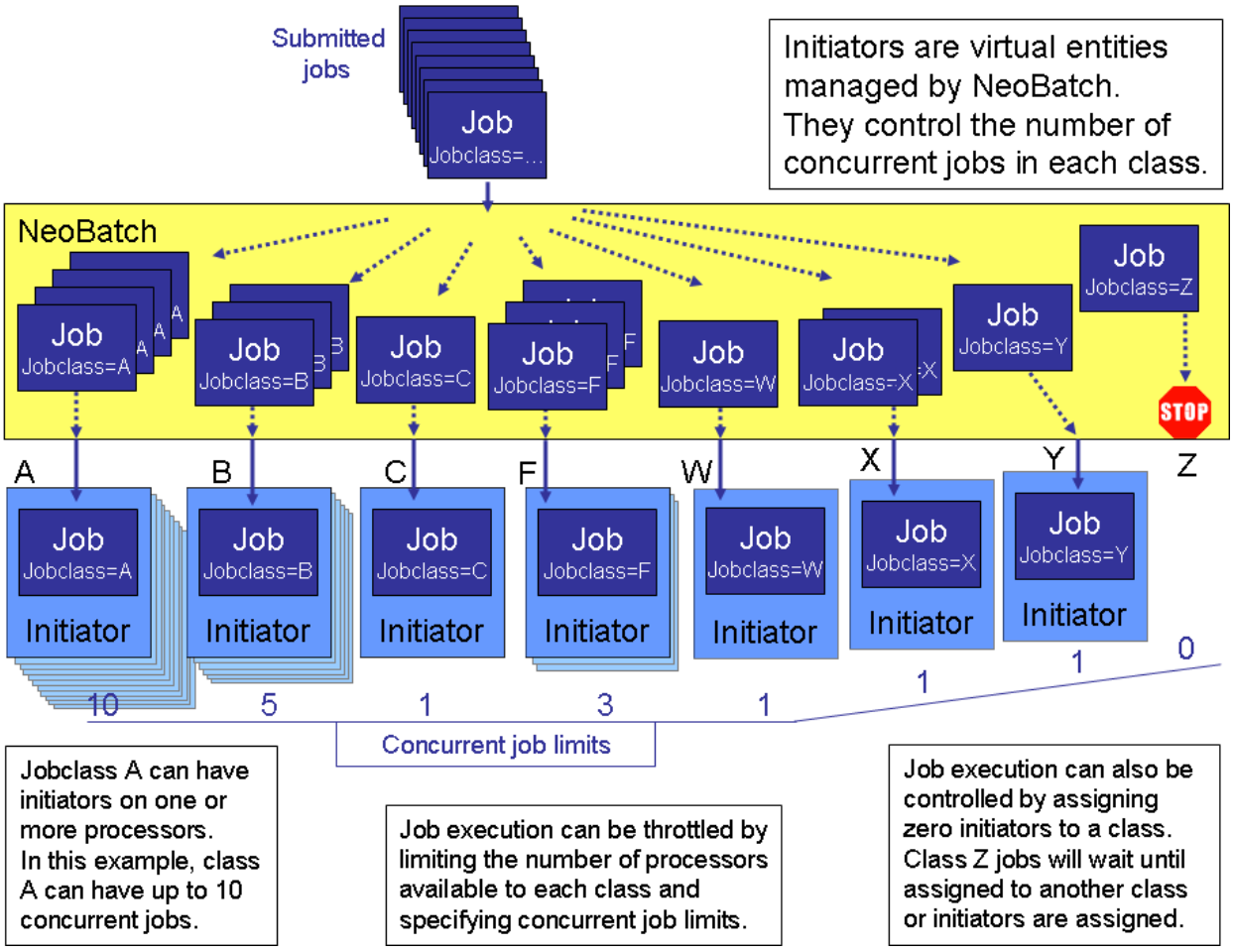


Figure 5. NeoBatch use of initiators

Because the JCL support package preserves the job classes that were defined within the JCL, it is able to prioritize jobs using the best practices established on the mainframe.

Customers who have migrated mainframe batch applications have reported general performance gains due to a combination of flexible hardware and operating system resources and products like NeoBatch.

2. Application Code—Programs and Utilities

Each step of the batch job runs a program. The program may be a business application program, an IBM utility, or a third-party utility.

Main Issues

- COBOL syntax compatibility
- SQL syntax compatibility
- Semantic compatibility
- EBCDIC to ASCII coding differences
- Other languages such as assembler and PL/I
- Utilities

- Sort syntax compatibility
- Performance

Program and Utility Issues

As with the issues involved in migrating batch application infrastructure, how well issues with batch job steps and programs are handled depends on the vendor solutions chosen to support the migration. Because it is best to use a consistent set of solutions, this paper describes example solutions from Microsoft Partner Fujitsu, or solutions that work with Fujitsu's migration suite. The Fujitsu NetCOBOL compiler and NeoSort utility are tools for migrating mainframe application code to the Windows platform. Microsoft SQL Server™ is the recommended solution for migrating from SQL databases.

COBOL Syntax Compatibility

When migrating from one platform to another, one obvious question is whether the source code will compile without errors. IBM mainframe COBOL programs benefit from two major standardization forces:

1. Because a large proportion of the world's COBOL programs reside on mainframes, much of the IBM COBOL syntax has become a de facto standard for COBOL.
2. The ANSI COBOL standard has been widely accepted and followed.

The easiest programs to migrate have syntax limited to ANSI standards. That subset of programs is very portable. Programs in which the syntax utilizes a number of IBM extensions usually migrate without problems because products such as NetCOBOL have implemented most IBM extensions or have well-defined, easy-to-implement conversions from IBM syntax to the NetCOBOL equivalents.

As is the case with other COBOL vendors, Fujitsu provides tools to analyze COBOL code and to report on the code's compatibility with the new environment. Vendors also can perform a proof-of-concept project to demonstrate how well the code migrates.

Table 1 provides examples of IBM COBOL syntax (not ANSI-standard) and how that syntax is handled during migration.

Table 1. IBM COBOL Syntax Examples

Difficulty	IBM COBOL syntax	Handled by
Automatic	USAGE COMPUTATIONAL-1	NetCOBOL compiler

Easy	> THAN, < THAN	"Change all" or self-automated process:
		IBM COBOL code
		`IF A = TO B`
		`  OR A > THAN C`
		`  OR A < THAN D`
		Change to
		`IF A = B`
		`  OR A > C`
		`  OR A < D`

Moderate	TRANSFORM	Straightforward manual change to INSPECT:
		IBM COBOL code
		`TRANSFORM DATA-T`
		`FROM "ABC"`
		`TO "CAT"`
		Change to
		`INSPECT DATA-T`
		`CONVERTING "ABC"`
		`TO "CAT"`

Hard	Channel names—defined in SPECIAL NAMES and used in: `WRITE … AFTER ADVANCING channel-name` Channel-name can be configured to advance the print position to a particular point on the page.	Inspection of code and analysis of the channel-name positions used. For simple channel use, such as channel 1 being top-of-page, the statement can be changed to the equivalent statement supported by NetCOBOL (e.g., WRITE … AFTER ADVANCING PAGE). For more complex channel use, such as advancing to a particular position on a page, additional coding changes may be required.

SQL Syntax Compatibility

The ease with which SQL can be migrated from the mainframe depends on two factors:

- Whether a compiler or preprocessor supports the syntax used.
- Whether the underlying database supports the SQL syntax or functionality. For example, some SQL statements pass somewhat directly to the database, while others might be converted into a compatible call to the database.

When migrating mainframe DB2 application code to SQL Server, SQL syntax will migrate about as easily as COBOL syntax does. Most syntax migrates with no change, some syntax requires minor changes, and some constructs require more effort to handle. As with COBOL, tools are available to analyze SQL code and to determine the number of items that require change.

Table 2. DB2 SQL Syntax Examples provides examples of SQL syntax and how that syntax is handled during migration.

Table 2. DB2 SQL Syntax Examples

Difficulty	IBM DB2 SQL syntax	Handled by:
Easy	DB2 allows a right angle character ¬ before an equal sign = to indicate “not equal to” ¬=.	Change all ¬= to the syntax supported by NetCOBOL and SQL Server: <>
Moderate	IBM COBOL allows host variables (data items used within EXEC SQL statements) to be defined in WORKING-STORAGE outside the statements EXEC SQL BEGIN DECLARE SECTION and EXEC SQL END DECLARE SECTION. NetCOBOL does not allow this.	Move the items so that they are between the BEGIN DECLARE SECTION and END DECLARE SECTION statements.
Difficult	DB2 supports separate date and time data types as well as a combined date-time type. SQL Server only supports a combined date-time type.	Analyze the use of the date or time items, determine the functionality achieved, and devise techniques to obtain the same functionality with SQL Server.

Semantic Compatibility

Having checked that the COBOL syntax and SQL syntax can be migrated, it is still necessary to make sure that the code functions as intended. Where syntax standards have not been defined, different COBOL and SQL vendors have often implemented different conventions. However, because products such as NetCOBOL were created to act like IBM COBOL, semantic differences are usually quite rare and generally only occur in situations where data has not been initialized and where a data item contains values that are invalid for its data type.

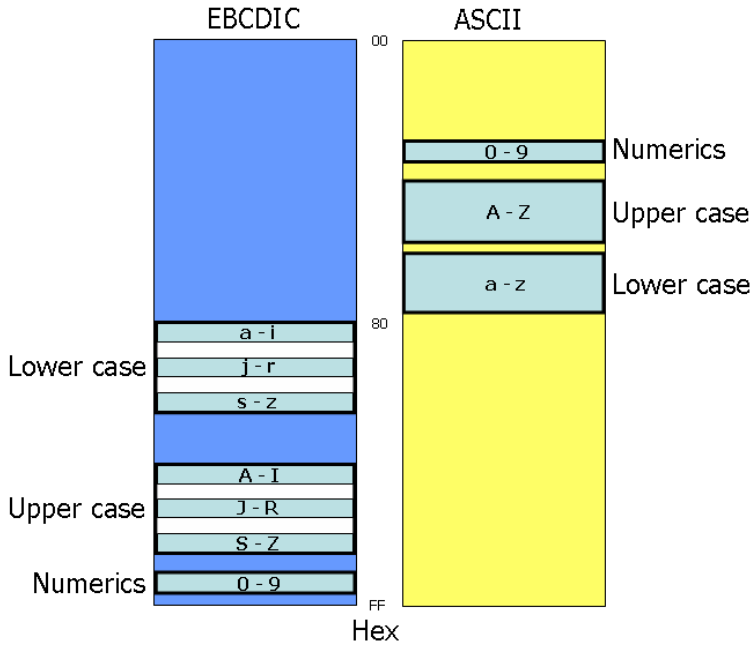
Such issues can be identified through thorough testing or with a source code analysis tool such as EZSource that will identify situations where uninitialized data items are used as source items in statements. Identifying these situations and ensuring the data is initialized appropriately will help yield more resilient code.

EBCDIC to ASCII Coding Differences

A later section discusses the issues of migrating data files from EBCDIC to ASCII. This section is about how EBCDIC and ASCII differences can affect the procedural code.

The main point to note about the EBCDIC and ASCII character sets is that they order lowercase, uppercase, and numeric characters differently, as illustrated in Figure 6. In EBCDIC, the order is lowercase, uppercase, and numeric; in ASCII the order is numeric, uppercase, and then lowercase. Obviously the values used to represent the characters are

different too but, as explained in the bullets below, the difference in values only affects the code if it uses rather rare or unusual coding constructs.



Note: Diagram assumes character values are arranged from lowest, X″00″, in the top left of the EBCDIC and ASCII blocks, to highest, X″FF″, at the bottom right. The rectangles within the blocks show the relative positions of the numeric and alphabetical characters.

Figure 6. Comparing EBCDIC and ASCII collating sequences

Changing EBCDIC to ASCII when migrating from the mainframe to the Windows platform requires changes to code in the following situations:

- When COBOL code uses the hexadecimal value corresponding to a particular character. For example:

  ```
  MOVE X″81″ TO CHARACTER-ITEM
  ```

  To address this issue, inspect the code to confirm that a hexadecimal value is being used where an alphanumeric literal should be used. Then make the replacement.

- When a comparison or decision point in the logic depends on the order of numeric, lowercase, and uppercase characters. The following statement is true in EBCDIC but not true in ASCII.

  ```
  IF SIX-BYTE-ITEM >= “aaaaaa” OR SIX-BYTE-ITEM <= “999999”
    <then the character is in the range of valid characters>
  ```

  Any string with an uppercase character would fail this test in ASCII. To address this,

change comparisons so that they are correct for ASCII values.

- When the logic of the program assumes that characters are sorted in the EBCDIC collating sequence. In the following example, the program logic assumes that alphabetical characters will precede numeric characters. The order of the data in the file matches the intended order of the processing: Items with numeric IDs will be processed after the items with alphabetical IDs.

```
IF SIX-BYTE-RECORD-ID > "ZZZZZZ"
  < alphabetical IDs processed; proceed with numeric IDs (which
have a different purpose)>
```

  Migrating this type of code to an ASCII environment involves more work. It is insufficient to convert the data to ASCII and then change the test because that places items with numeric IDs before items with alphabetical IDs. This will result in the data being processed in the incorrect order. The solution to this issue may require changes to the program logic and data file structure.

The data could be kept in EBCDIC to avoid these problems, but this option has two significant disadvantages:

1. Because the EBCDIC data is processed in an ASCII environment, EBCDIC-ASCII differences will eventually emerge, possibly after migration has completed. For example, ASCII encoding will be expected by most utilities and when printing data.
2. Enhancements using the .NET Framework class library or other programming languages within the .NET Framework will be limited or difficult because the Framework assumes ASCII or Unicode data values. EBCDIC-ASCII differences will need to be identified and resolved each time enhancements are considered.

Adding an intermediate step in the change from EBCDIC to ASCII may not be worth the extra effort if migrating to ASCII is still the long-term target. An intermediate step may double the testing effort—one round of testing to check that the migration of EBCDIC on the mainframe to EBCDIC on Windows gives the same results and another to ensure that the change from EBCDIC on Windows to ASCII on Windows gives the same results.

Other Languages such as Assembler and PL/I

Applications that contain modules written in languages that do not have compilers in the Windows operating system or the .NET Framework usually need to be rewritten.

If a small amount of code is written in other languages, rewriting can be a straightforward process with long-term benefits. Modules can be rewritten in the same language as the rest of the application (likely COBOL) or in another language that best supports the application's functionality. Often the rewrite is appropriate because the other language was used to perform an operation that COBOL could not perform in years past but that it now does, or to optimize performance on the mainframe in ways that are irrelevant to the new environment.

If a large amount of code is written in other languages, careful consideration is needed to determine the correct route. Language conversion tools or services can be used, or it may be that migration is not the correct solution for the applications at this time. It might be better to wait until code converters for the other language improve, or to undertake a gradual rewrite-as-you-update approach.

Utilities

Job steps often invoke third-party utilities to perform particular functions useful to the batch process. Creating and printing reports is a common function performed by such utilities.

The appropriate solution depends on the utility: whether its function continues to be important in the Windows environment and whether the function it performs can easily be replaced by newer technology.

Most often, Windows-based equivalents to the utilities allow existing setups to be quickly migrated or for functionality to be quickly recreated. For example, the main effort needed to create reports can be in determining the content and layout of the information so that reproducing the reports with another tool is a straightforward process.

Sort Syntax Compatibility

Because sort is used so frequently in batch processing, it is important that a migration solution supports the syntax used by the mainframe sort utilities. DFSORT and SyncSort are commonly used utilities and employ a similar syntax. Without a syntax-compatible solution, a large proportion of batch jobs would need to be edited to change the sort parameters from the mainframe format to one that will work with a Windows-based sort program.

The Fujitsu migration suite uses NeoBatch to process the sort syntax and to drive a specially packaged version of Nsort, a sort program that is optimized for multiprocessor Windows environments. The specially packaged version of Nsort is called NeoSort.

Performance

Most organizations need to run batch programs within a given time window, so it is critical that migrated batch applications run faster in the new environment than on the mainframe.

Companies that have benchmarked performance before and after migration report that migrated batch applications run faster on the Windows platform than on the mainframe. These results are contrary to public perception because mainframes have long been regarded as the best option for raw computing power. However, these performance results are well documented.

This white paper will not compare the performance of applications on the Windows platform with the performance of applications on mainframes, as that topic is a white paper in itself. However, here are some figures reported by organizations that have migrated batch applications to the Windows platform:

- As far back as 6 years ago, San Diego City Schools reported that batch applications ran 30 percent faster after migration and with much cheaper hardware.
- Euronext reported that some batch execution times reduced from 1.4 hours to 7 minutes after migration from a relatively small mainframe.
- Scottish & Southern Energy set records for the size of database supported by its applications after migrating to a system running Windows.
- A first-tier telecommunications company migrated a batch application from a 3000 MIP mainframe to a multiprocessor server running the Microsoft Windows Server® 2003 operating system as part of a proof-of-concept demonstration and found that batch processing times reduced from 2.5 hours to 27 minutes.
- In February 2006, NeoSort set a new world record for the MinuteSort benchmark.

To find out if these types of performance figures can apply to your batch applications, engage a proof-of-concept project on a representative batch application. Then you can be

confident that batch jobs will still run within existing time frames, or you may discover that you have the option of shortening your batch processing time frames.

3. Application Data—Files and Databases

The purpose of batch jobs is essentially to manipulate, update, and report on data contained in data files and databases. Therefore, it is obvious that the integrity of this data must be preserved during the migration.

Main Issues

- Data file compatibility
- Database compatibility
- EBCDIC to ASCII data conversion
- Database and file size
- Planning and testing
- Performance

Data File and Database Issues

For our example migration suite, the recommended solutions for data are:

- Sequential and relative files—use the NetCOBOL file system
- Indexed files (VSAM)—use the Pervasive Btrieve file system
- Relational databases—use Microsoft SQL Server
- Hierarchical and network databases (IMS, IDMS)—convert to a relational database and migrate to SQL Server
- Other databases—review the availability of Windows versions of database software or other options for migrating to SQL Server

The following sections discuss the main issues involved in migrating data and how the example solution set handles them.

Data File Compatibility

Most potential data file compatibility issues are handled by the file system, so data file incompatibilities do not need a lot of attention during the migration process. This is largely due to COBOL standardization—because data is central to applications, most COBOL vendors followed the same standards for supporting data files. Possible areas of incompatibility include:

- **Data organization**: Incompatibilities can occur if a particular data organization was not supported or was implemented in a manner that produced different results. For example, different vendor products employ different file formats. While the file formats function in the same way, the data may be arranged and managed differently. The only file format common across systems is the fixed-length sequential format. Consequently, migrating files other than fixed-length sequential files involves unloading the files into a sequential format on the mainframe and reloading them onto the Windows platform to return the data to the desired organization, such as indexed.
- **Variable-length records**: Files, other than line-sequential (text) files, that support variable-length records need special handling. This is because the way that variable length-records are handled is specific to the file system in use. For example, one system might store the record lengths as part of a separate file of indices, whereas another might store the record lengths in 4 bytes immediately preceding each record.

To migrate files with variable-length records:

a. Download the files as text files (assuming they contain no binary data) so that the downloaded file is an ASCII text file, which can then be loaded back into the COBOL format file using a suitable file utility.

Or:

b. Write a program or use a utility on the mainframe that creates a fixed-length sequential file (or your own variable-length format) of all the records with an item added that contains the record length. Then, from the Windows environment, a matching program or utility can read the data, using the record-length field to set the length of the records as they are written to their target format.

- **Data values**: When a file is migrated, the values in the new system must be interpreted in the same way they were in the old system, or part of the conversion process must map the values so that this is the case. For example, if a record contains a signed display numeric item, the Windows system should give that item the same value it had when it was written on the mainframe. If the Windows system uses a different convention for storing the sign, then the file migration process must convert the value from the mainframe convention to the Windows convention so that the stored value is the same.

  Once again, COBOL standards ensure that most data values are compatible between systems.

  For a conversion to NetCOBOL, the following data values need to be monitored:

  a. Converting from EBCDIC to ASCII (see "EDCDIC to ASCII Conversion" later in this white paper).

  b. Displaying numeric items with included signs—items defined as PIC S9(n) DISPLAY with no SIGN SEPARATE clause. Both IBM COBOL and NetCOBOL store the sign by setting a bit in the upper nibble of the sign-carrying digit, usually the trailing digit unless the SIGN LEADING clause is specified. For example, if the trailing digit is a 5, without a sign it would be stored as x"F5" with IBM COBOL and x"35" with NetCOBOL. If the 5 has a negative sign, these would become x"D5" and x"55". Looking at these values as characters, they would be N and U respectively. Converting such items as if they were regular display characters so that x"D5" is converted as if it were N to ASCII x"4E" would corrupt the data because x"4E" is an invalid value for a sign-carrying digit in NetCOBOL. This issue is handled by using a data conversion utility that can perform different conversions on different fields within records and can convert the sign-carrying digits of numeric display items, or by writing your own file conversion programs.

  c. Writing floating-point numbers to files. It is rare for floating-point numbers to be stored in files, but if they are, the numbers will need to be unloaded from their binary floating-point format on the mainframe and reloaded into the Windows floating-point format on the Windows environment. Although identical in length, binary formats for mainframe and Windows floating-point items are different. IBM COBOL supports an external floating-point format, in which floating-point numbers are stored in the form ±m.mmE±nn, where each character or digit in the string is stored as an EBCDIC character. NetCOBOL does not support this format, but a file containing floating-point numbers could be read with a small modification to the record definition and those values then used to set a floating-point item.

- **Multiple record types**: It is quite common for data files to contain records of several different types, with each type containing different data layouts. For example, customers in different countries may have different address formats or a file may contain both summary and detail records. If the records contain mixtures of binary and character data, this can cause challenges in the conversion from EBCDIC to ASCII (see "EDCDIC to ASCII Conversion" later in this white paper).
- **Indexes and alternate indexes**: Another area where incompatibilities can arise is again helped by the standardization of the COBOL language. Comparing indexed file support between IBM COBOL and NetCOBOL there is only one main area of incompatibility: the order in which records with duplicate alternate key values are read. IBM COBOL specifies that records written with duplicate alternate key values be read in the same order in which they were written (when they are being read sequentially). NetCOBOL does not specify the order in which such records will be read. There is therefore a chance that logic may depend on these duplicate records being read in a particular order, but NetCOBOL would not provide them in that order. Resolving this difference requires adjusting the logic so that it does not depend on the order of the duplicate items or adding a field to the index that ensures that items with duplicate keys are ordered as desired.
- **Print reports**: A special type of data file contains the reports that were created by batch jobs and that are to be printed. On the mainframe, these can be created in a manner that is as analogous to an ASCII text file in the Windows environment. Or, they can be created with the use of control codes written to the first byte of each line to communicate information such as the start of a new page or to advance to a channel, where the channel defines a configurable position on the page.

  When the latter type of print file is used, the program logic must be changed to write a more standard type of print file or utilities must be used that emulate the ability to interpret the control codes. NeoBatch's SYSOUT print spooling handles the basic (non-channel) controls. For more complex uses of control codes, companies such as Barr Systems offer specialty software to perform a variety of print management tasks, including the handling of several types of mainframe format techniques.

Database Compatibility

Whether database compatibility during migration will be a key issue depends on the type of database used. Different relational databases have similar features because their development has been guided by SQL standards. Consequently, migrating DB2 or Oracle databases to relational databases that run on the Windows platform, such as SQL Server, often is a matter of checking a few details. Other types of databases, such as the hierarchical IMS or networked IDMS databases, do not have popular Windows-based equivalents. Thus, research into available migration solutions for these databases is needed before migration can become a serious option. Following are some issues to consider for different types of database.

Relational Databases

- **Load and unload utilities**: Because database use of resources can be expensive, many applications perform standard processing or reporting by unloading the required data into sequential files that are then handled by the batch jobs. DB2 provides its own data loading and unloading utilities that are different from SQL Server utilities. Therefore, migrating applications that rely on these DB2 features requires creating utilities to do the

loading and unloading of data. This is a straightforward task that just needs to be included in the plans.

- **Stored procedures**: Code stored in the database and run when certain criteria are satisfied can be migrated to SQL Server assuming the language used to write the procedures is supported in the .NET Framework, as would be the case for procedures written in COBOL and migrated to NetCOBOL for .NET.
- **Error codes**: The SQL standards and SQL Server emphasize SQLSTATE values for communicating error conditions over SQLCODE, which has been the bearer of error codes on the mainframe. Applications that check SQLCODE for the common values of zero and 100 will generally work as expected except when SQL Server returns a zero for situations in which DB2 would return a non-zero value. Applications that rely on these and other values will need to be tested carefully, and possibly recoded, to ensure that the applications continue to respond appropriately to all error situations.
- **Date and Time data types:** As described in Table 2, SQL Server does not support DB2's date and time data types, having a single date-time data type. Tables that use these data types must be converted when migrating the databases.
- **Limits**: Another possible area of concern might be whether the database running on Windows can support the size of the mainframe database. Table 3 compares various size limits between DB2 and SQL Server. As shown, it is unlikely that many applications will encounter challenges with limits in either system.

Table 3. Relational Database Size Limits

Features	DB2, pre-V8	DB2 V8	SQL Server
Table name size	18	128	128
VIEW and ALIAS names	18	128	128
Column name size	18	30	128
SQL statement length	32 KB	2 MB	65,536 * Network packet size
Tables in a join	15	225	256
Character literals	255	32,704	$2^{(31-1)}$ varchar(max)
Index key	255	2000	800
Virtual storage (2GB)	2^{31}	2^{64}	2^{64}
Partitions	254	4096	1000

IMS, IDMS, and Other Databases

The popularity of relational databases has tended to confine nonrelational databases to niche markets. Consequently, very few nonrelational databases have clearly viable equivalents that run in the Windows environment. Options available for the migration of these databases include:

- **Before migration, craft the data and the code to use a relational database**. Some companies have recognized that a move to a relational database is important for long-term goals. They have therefore approached batch application migration in two

stages. The first stage converts the data to relational form, and the second stage moves the batch system to Windows. Stockholmshem[1] is one such company that decided this was the best option.

- **Work with a system integrator that specializes in data migrations**. The need for updating to modern database solutions has grown large enough that a number of system integrators, such as Mainframe Migration Alliance partner Sophisticated Business Systems[2], provide automated solutions.
- **Find a vendor with a good migration solution for your database**. It is also worth checking to see if there is a vendor with a good solution that makes it easy to migrate the database and support applications. With increasing interest in migration and rapidly advancing technologies, it is possible that someone has recently produced a solution that is just right for your needs.

EBCDIC to ASCII Data Conversion

Windows is an ASCII or Unicode environment, with the majority of characters in common English usage overlapping between the standard 128 ASCII characters and the single byte values of the UTF-8 form of Unicode. To take full advantage of the tools available in Windows environment, data must be encoded in ASCII or Unicode.

A major step in the migration process is to convert data from EBCDIC to ASCII or Unicode. Moving to Unicode can be a major step, with the possibility of many data fields lengthening. Although this might be a good move for international organizations to consider for the long-term (so that the source language of the data does not require special coding), the move to Unicode is best considered as a separate step from migration. We will therefore focus on the issues to be considered when converting data from EBCDIC to ASCII.

Converting data from EBCDIC to ASCII is a straightforward process when special attention is given to the following details:

- **Mixed binary and alphanumeric data**: If a file contains mixed binary and alphanumeric data, the data conversion must be performed by a tool that can convert character data and leave binary data untouched. Ideally, the tool should process the COBOL copybook that describes the records in the file to determine what data to convert and what data to leave. Pervasive Data Integrator and Fujitsu Data Converter are examples of two tools that can perform this task.
- **Multiple record types or redefinitions**: Files that contain mixed binary and alphanumeric data and that also contain multiple record types add a level of complexity to the conversion, as illustrated by the following record definitions.

```
FD FILE-1.
01 RECORD-1.
 03 RECORD-TYPE          PIC 9.
 03 REC1-DATA-1          PIC X(5).
 03 REC1-DATA-2          PIC 9(5) USAGE BINARY.
01 RECORD-2.
 03 RECORD-TYPE2         PIC 9.
 03 REC2-DATA-1          PIC 9(5) USAGE BINARY.
```

---

[1] The Stockholmshem migration was performed by its service partner, Datarutin (now part of EDB). Details of this project can be found in the EDB area of the Mainframe Migration Alliance site (www.mainframemigration.org): www.mainframemigration.org/ShowPost.aspx?PostID=513&fm=cs

[2] Sophisticated Business Systems (now named ATERAS): www.ateras.com

```
03 REC2-DATA-2          PIC X(5).
```

Figure 7 illustrates how these two records would be laid out.

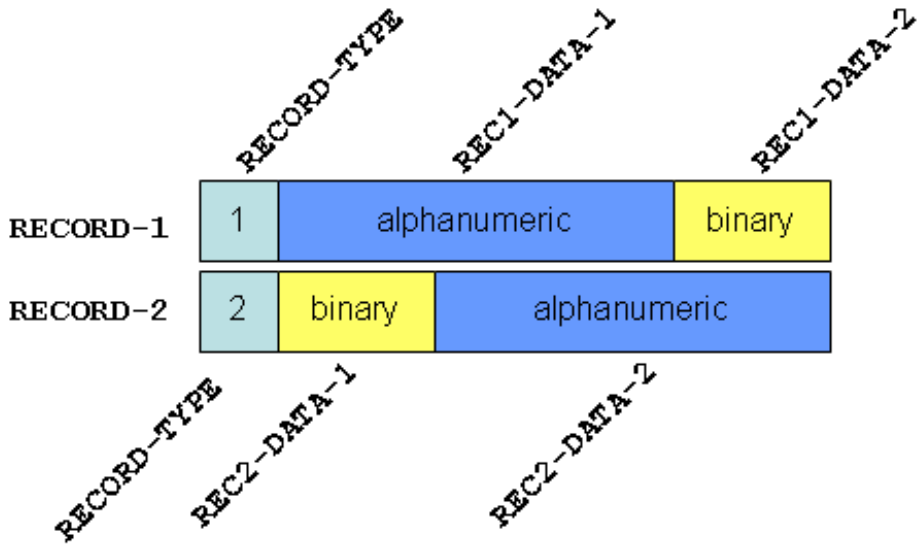


Figure 7. Diagram of a file containing two different record types

Applications that access this file would probably distinguish between the record types by the value in a particular field. In Figure 7, this is the RECORD-TYPE field, with RECORD-1 identified by the value 1 and RECORD-2 by the value 2.

Any utility or program converting a file containing these two different record types not only needs to be able to convert some fields and not others, but it also needs to be able to distinguish between record types. Once this situation has been identified and the means of distinguishing records has been specified to the tool or data conversion program, the data conversion can proceed in the same record-by-record manner it would do for less-complex record layouts.

- **Particular item types**: As explained under “Data Values” earlier in this white paper, some types of items must be converted during migration from the mainframe to the Windows platform. Although these conversions are not EBCDIC to ASCII conversions, it makes sense to perform them at the same time as the character set conversion. That way, only one conversion pass is performed on the data.

Special characters: Some applications may rely on special characters that are outside the standard ASCII set of characters defined by the ASCII values `X″20″` through `X″7F″`. Most often, these will be applications working in a language that extends the Latin alphabet or uses a different alphabet.

On the mainframe, support for these characters will have been enabled through a particular EBCDIC code page. The situation on the Windows platform is the same: Extended character sets are supported by different ASCII code pages. Those responsible for migrating the data need to make sure that equivalent code pages are used and that all relevant characters are mapped and converted correctly.

- **Collating sequence**: The section "EBCDIC to ASCII Coding Differences" explained how the collating sequence for ASCII data differs from the EBCDIC collating sequence. In terms of the EBCDIC to ASCII character set conversion, the collating sequence change is typically handled automatically as the data is unloaded from an ordered organization (indexed) to a sequential file, downloaded, converted to ASCII, and reloaded into the ordered organization.

  The collating sequence change will not be automatic if a sequential file is assumed to be in EBCDIC collating sequence order. In such cases, it may be necessary to sort the data after conversion so that it is stored in ASCII collating sequence order, though the code will have to be reviewed as described in "EBCDIC to ASCII Coding Differences" to ensure the logic is valid for the new data order.

Database and File Size

The amount of data contained in a database or data file has not yet been an obstacle to running applications that were migrated from the mainframe. Some of the world's larger databases are supported by SQL Server, according to the results published in Winter Corporation's Top Ten lists[3]. Appropriate software and hardware configurations in the Windows environment are well able to support any amount of data effectively.

Where size is a factor is in the switchover of production from the mainframe to the new platform. The largest files and databases can take so long to transfer between platforms that it is not practical to do the switchover by stopping operations on the mainframe to perform the migration and then to restart the applications in Windows. The downtime is too long for most operations; and the shorter the tolerable downtime, the smaller the threshold is for the size of database that needs an alternative switchover approach. Alternative approaches need careful planning and typically involve migrating the bulk of the data at one point, and then migrating data updates that can be applied just prior to the switchover.

Planning and Testing

Most people who embark on a migration project for the first time underestimate the amount of work involved in working with the data.

A major part of what is underestimated is that the need for good mechanisms to automate the data conversion so that it can be easily repeated during the testing phases. Many batch jobs require specific snapshots of data that have many interdependencies. Trying to get a full snapshot at any given time in order to properly test is often a big challenge. The solution is to establish good project management at the outset and to perform clear upfront analysis of the operational characteristics of the source and target environments. The analysis indicates the testing requirements from which the required data snapshots can be specified and appropriate repeatable mechanisms set up.

Performance

TPC-C benchmarks[4] show a reasonable proportion of systems running SQL Server in the top ten performance category and a high proportion of these systems in the Top Ten by Price/Performance category. This suggests that performance is more dependent on the

---

[3] Winter Corporation Top Ten Award Winners: www.wintercorp.com/VLDB/2005_TopTen_Survey/TopTenWinners_2005.asp

[4] TPC Web site: www.tpc.org/default.asp. See Top Ten TPC-C Results by Performance and Top Ten TPC-C Results by Price/Performance.

hardware platform than the software, and it demonstrates that the best price-to-performance ratios are likely to be found on Windows platforms. IBM does not publish TPC-C tests for its mainframe so a direct comparison is not possible.

As with application performance, most people need to see the figures for their own applications before placing confidence in a new platform. It is a long-accepted notion that mainframes rule when it comes to execution power. That may no longer be the case.

4. Script Control—JCL

JCL is a complex control language with strong links to mainframe features. Although mainframe JCL is not pretty, it has some very useful features developed over many years of batch job support and these features are not standard in the Windows environment. Ensuring that the business value of mainframe batch jobs is preserved requires that some of these powerful features—file structures such as Generation Data Groups (GDGs)—be replicated.

Main Issues

- IBM JCL does not run on Windows as a native script syntax, so support for JCL is needed, including support for JCL's stronger features
- JES2/JES3 control statements (JECL) and PROCs must also be supported.
- Identifying unsupported features.
- JCL represents a carefully designed set of processing actions usually understood by a group of specialized engineers. This expertise should be transferred to the new environment

Expert JCL knowledge is not immediately applicable to working with .NET technology. Consideration must be given to how people with an intimate understanding of the batch jobs can transfer knowledge (including much business knowledge) when the applications migrate.

The JES has its own set of command statements (JECL) that can be intermixed with JCL statements. Any tool used to migrate JCL should be able to recognize JECL commands in batch jobs and handle or ignore those commands as appropriate to the context.

Likewise, JCL coders often create libraries of JCL procedures, called PROCs, from the JCL script, and any tool used to migrate JCL must support these.

A repeating theme of consideration among the elements that comprise batch jobs is whether they use JCL syntax or features supported on the mainframe but not supported in the new environment. Mainframe JCL has evolved over 30 to 40 years and has many detailed features. Infrequently used features may not be supported in the new environment because they require more effort than makes sense for the migration. When considering migration from the mainframe, unsupported features should be quickly identified to see if acceptable ways of providing support exist in the new environment.

Handling JCL Issues

The above list makes clear that any straightforward migration of JCL will require tools designed specifically to handle the form and functions of the language. Decision makers considering the migration of JCL to a new platform should recognize the complexity and challenge involved if they expect their own team of engineers to solve these issues in a short time frame.

Although JCL can be regarded as a simplistic scripting "language" in terms of the constructs it offers, JCL statements have evolved to have complex arrays of parameters. During migration, one option is to rewrite JCL script into a Windows-based scripting language. However,

authoring new scripts is time-consuming and risks the introduction of errors. Also, the new scripting language may not have all of the options found in the mainframe JCL command set.

Microsoft Partner Fujitsu has addressed these issues with NeoBatch, a product that runs JCL and provides a Job Manager with interfaces and nomenclature familiar to users of mainframe batch systems. NeoBatch also provides an option to convert JCL to Microsoft Jscript® so that application development can transition to a more modern scripting language if desired. The NeoBatch conversion process provides maintainable code with object models to implement some of the more difficult features.

For example: using NeoBatch, JCL jobs are migrated and run just as they were on the mainframe, with changes made only for the unsupported syntax and features. NeoBatch provides the option to execute the JCL directly or to do a once-for-all-time conversion to JScript. This feature allows flexibility in deciding how long to continue working with JCL and when it is the best time to transition to JScript in order to access enhancements on the Windows platform. These features of NeoBatch help engineers avoid a large script recoding effort while also offering the option of not having to stay with JCL forever.

Support JCL Features

Another process for which Windows does not have analogous capabilities is the link JCL creates between the executable program and the operating system services. Some of these features include:

- Generation data groups (GDGs)
- Partitioned data sets (PDS)
- SYSOUT organization of batch data output

The importance of these features means that if they are to be migrated to the Windows environment, the batch system support package must replicate them. Without that support, migration would involve a significant number of manual changes to the application program and require operations staff to adapt to new processes and techniques to achieve the same results.

NeoBatch supports the syntax of each of these features, providing identical behavior and mapping data set and GDG names to corresponding folder names and file names. This is achieved by providing classes that emulate the behavior of these features and by automatically converting JCL to JScript scripts that use the classes to obtain the same results as the original JCL. This approach has the following advantages:

- Batch jobs that rely on the strong features of JCL can be migrated straightforwardly.
- There is no loss of functionality due to migration.
- Because the batch jobs are run as JScript scripts, future enhancements can take advantage of JScript constructs and the new technologies accessible from JScript.

Transfer of Batch Job Expertise

Because NeoBatch supports JCL as it is and provides a Job Manager with familiar objects and terms such as file catalog, SYSOUT job output, and console window, operators with many years of experience working with batch jobs can transfer skills to the Windows environment almost immediately. Figure 8 illustrates how the NeoBatch Job Manager presents batch job setup, controls, and results using familiar terminology and with a user interface layout containing multiple viewing panes (similar to the Microsoft Visual Studio® development system user interface). Becoming familiar with the Windows user interface and using a mouse may be all that is required, and most already have those skills. It is then up to

management and the experts to determine how quickly to add new skills, such as coding in JScript.

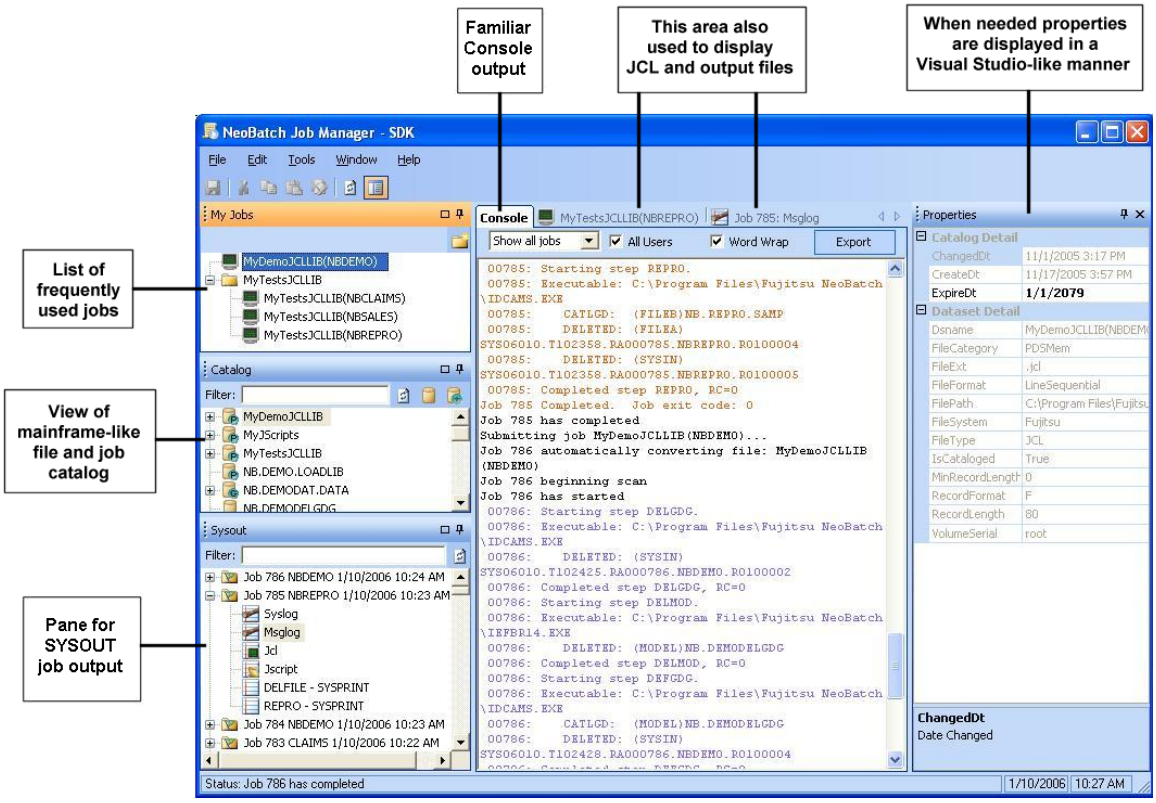


Figure 8. The NeoBatch user interface

Support for JECL

NeoBatch assumes that most of the JECL (JES commands) is used for actions on the mainframe that are irrelevant on the Windows operating system and the .NET Framework. NeoBatch recognizes JECL statements—to avoid parsing errors when the JCL is processed—but these statements are treated as if they were comments. To confirm that this is a valid strategy for your system, consider your use of JECL and whether that use is relevant in Windows and the .NET Framework. For example, JECL commands are occasionally used for routing output printing to a specific device. Such requirements must be reviewed individually for each case.

Support for PROCs

PROCs are such an integral part of batch jobs that any tool supporting JCL must also support them. NeoBatch stores PROCs using the same file names and emulates the mainframe PROCLIB search order by storing PROCs in folders, one folder per PROCLIB, and specifying search order at setup.

Unsupported Syntax and Features

The big question for any migration-supporting utility is: “What does it not support?” Using a source analysis tool, such as EZSource, designed specifically for mainframe-based systems can help identify unsupported features beforehand. Unsupported features need to be

reviewed to identify the ones that will be critical in the new environment and to devise strategies for how they will be supported.

The NeoBatch documentation lists what is and what is not supported, so it does not take long to determine if a critical feature of your batch jobs is supported. NeoBatch supports the most common features, but the number and impact of unsupported elements will vary from installation to installation.

Other Ways of Handling JCL Issues

The above approaches to handling JCL issues during migration focus on Fujitsu's approach using NeoBatch. Other ways of dealing with JCL issues:

- Employ other third-party JCL migration or emulation solutions. These solutions employ similar approaches to NeoBatch's, but they differ in user interface design, degree of integration with the .NET Framework, the syntax elements supported, and the manner in which the batch jobs can be updated in the future.
- Rewrite the JCL or perform a one-time conversion to a modern scripting language.

Revitalization Potential

Over the long-term, batch applications can be revived and extended in a number of ways. Some possibilities include:

- Enhanced reporting through SQL Server Reporting Services
- Transitioning JCL to JScript so that batch jobs can utilize features of the .NET Framework
- Reducing reliance on COBOL to widen recruiting prospects when COBOL programmers are scarce or to allow the adoption of more recently developed languages
- Taking advantage of XML
- Making some applications available as Web services
- Giving certain users the ability to execute batch jobs on demand

Companies that have taken the step of migrating batch applications to the Windows platform typically report the following benefits:

- Cost savings anywhere from 50 percent to 90 percent of annual maintenance budgets
- Performance improvements from 10 percent faster to 10 times faster
- Improved developer productivity

Check out published case studies to learn more about these benefits.[5]

[5] Migration case studies can be found on the Fujitsu and Microsoft Web sites.
Fujitsu: www.netcobol.com/migrations/index.htm#casestudies
Microsoft: Go to members.microsoft.com/CustomerEvidence/Search/FindEvidence.aspx and enter "mainframe migration" in the search field.

Conclusions—Complete Migration Solutions are Available and Provide Benefits

This white paper discussed the key areas to be considered when migrating batch applications to the Windows operating system and the .NET Framework. Although the list of areas may be daunting, when each area is broken down and the issues tackled methodically, it is evident that solutions are available for organizations seeking the short-term and long-term benefits of migration from mainframes. In summary:

- **Working with job classes, initiators, and schedulers**: The investment in hundreds or thousands of batch jobs set up and prioritized so they run efficiently and service business needs appropriately can be preserved when migrating to a system that provides initiator-like constructs so that job class to initiator mappings are still valid for tuning and balancing loads. Scheduling is another area where numerous business rules are intrinsically stored. This can be preserved by transferring scheduling rules mechanically to one of many schedulers available for the Windows environment.
- **Executable programs, such as COBOL programs or sort utilities**: These carry the majority of the business-rule processing contained in batch jobs. Compatible compilers and utilities running on Windows and the .NET Framework, together with available tools and expertise for performing the migrations, provide manageable program migration.
- **Data files and databases**: These contain the organization's lifeblood—the information about its customers and transactions. Once converted from EBCDIC to ASCII, COBOL and SQL standards ensure that the majority of functionality is preserved, with known solutions available for details such as SQL statements that do not have direct equivalents in the Windows environment.
- **JCL**: Representing many staff-years of effort in controlling program and utility execution, JCL can be migrated directly with JCL-supporting software and with the option of converting to a more modern, Windows-based control language such as JScript.

With sufficient knowledge of the challenges involved in migrating from the mainframe to a Windows environment, an organization can make a reasonable assessment of whether migration is the best solution before committing to that course. We hope this white paper will help you begin that assessment. For those seriously considering migration, we recommend engaging a solution provider to perform a proof-of-concept project in which one or two representative applications are migrated so that you can see the effort involved and the performance results on your own code.

With so much value invested in batch processing and so much potential to be unleashed in Windows environments, now is a good time to review how to take advantage of a great hidden asset: your batch applications.

Appendix A. The Details of Batch Processing

The details of each batch component, particularly those details relevant to or assumed in this paper, are explained in this appendix.

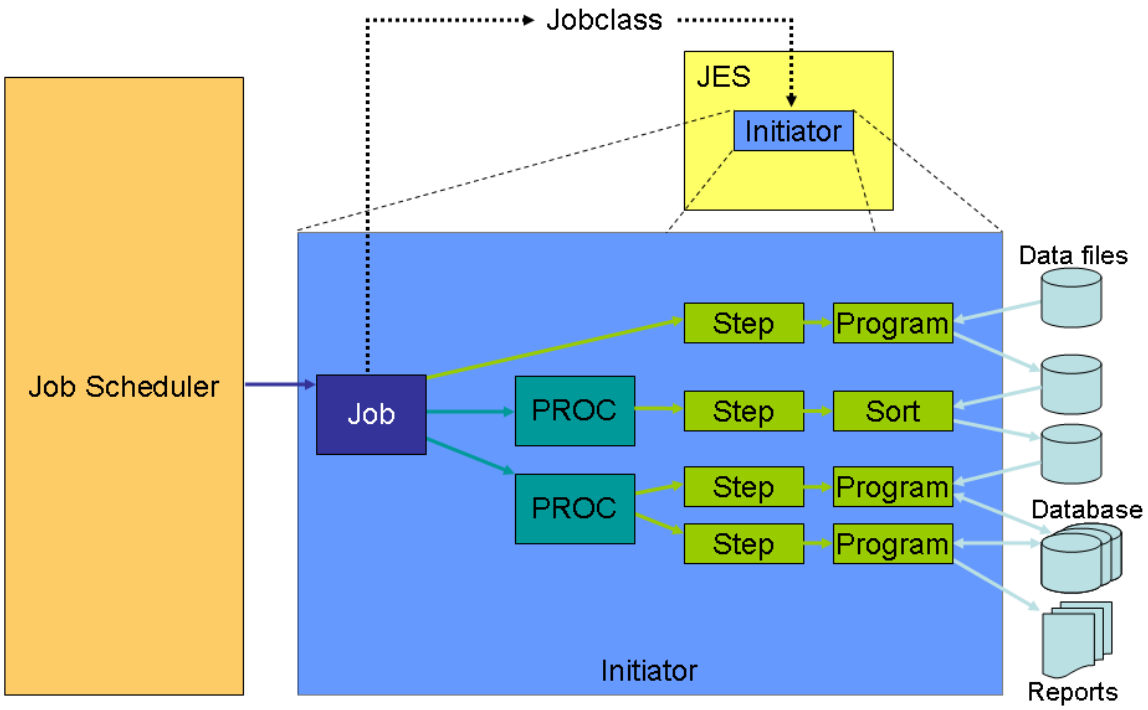


Figure 9. Schematic diagram of batch components

Batch Jobs—A Quick Overview

Figure 9, a repeat of the overview diagram in Figure 1, illustrates the key elements of batch jobs on the mainframe. Jobs are written in **JCL** (Job Control Language), which provides instructions on which **programs** will be run in the job's **steps** and which **files** or **databases** will be accessed or created by those jobs. JCL can contain conditional instructions or branches so that different actions can be taken based on the input to the job and the results and failures of the job steps. Jobs often use procedures, called **PROCs**, which are subsets of JCL stored in a system catalog.

Usually a **job scheduler** determines when the job will be started, though it is **JES**, the job entry subsystem, that starts the jobs based on the priority of the job and the availability of initiators. **Initiators** are sets of machine resources that can support one job at a time. Every job has a **job class,** and system programmers specify the classes that can execute in each initiator. Configuring the properties and number of initiators and the classes assigned to the initiators provides the ability to give different job classes different amounts of processing power or priority.

Each of these elements is described in more detail below.

Jobs—JCL

The surface layer of batch systems is the job—a set of instructions executed together and written in IBM JCL.

JCL

The "Dictionary of the Mainframe World[6]," published on the Internet by Software Diversified Services, provides the following definition of JCL.

> *Job Control Language. The language used to describe the steps of a batch job (files to be used, programs to be run, etc.). A generic term, hijacked by IBM to refer to the batch control languages for its System/360 operating systems. Then, as now, z/OS and VSE/ESA JCL is a clumsy and cumbersome system that is hard to learn, full of inconsistencies, and avoided by anyone with an iota of common sense and access to an alternative. The only excuse for it comes from one of the original OS/360 developers: They ran out of time to build a JCL interpreter so just used the Assembler macro processor and built a language (JCL) around it.*

From this description it is clear that JCL batch jobs may not be elegant, but JCL and related systems have evolved to provide sophisticated fault-tolerant environments for executing batch applications.

Steps

Jobs consist of one or more steps, with each step specifying a program to be executed and defining the files to be used as input to and output from that program. Generally the steps in a job are logically connected and make sense to run as a batch, such as end-of-day processing or calculating monthly interest payments. Exactly how much processing is performed in each job depends on the system architects—for example, whether files are backed-up as the last steps in the end-of-day processing job or as steps in a separate job. JCL provides statements to test whether a step executed successfully and to take different actions, such as aborting the job, if a step did not terminate successfully.

PROCs

Often jobs use procedures, usually referred to as "PROCs," which are prepackaged sets of JCL stored in the system catalog. Use of PROCs means that blocks of JCL that are similar or identical across many jobs can be maintained in one source file.

Starting Jobs

Originally, jobs were started by operators following instructions for when each job should be started and the order in which jobs should be run. Today, jobs are generally started by job scheduling software.

Business Rules

Although the programs run are responsible for the main business processing, the jobs and scheduling systems generally embody many business rules that are vital to preserve (for example, "execute end-of-day processing before executing the job to apply monthly interest earned").

---

[6] See www.sdsusa.com/dictionary

Other Elements

Jobs often contain more than just JCL; they can also contain the job entry control language (JECL) statements of JES described below.

Figure 10 shows some typical elements of JCL:

- The identifying JOB statement
- DD statements to define file details
- An EXEC statement that names the program to be executed
- Comment statements starting with `//*`

Even this short sample illustrates that JCL provides many options, that it is not an intuitive language, and that it takes a while to learn or master.

```
//DLYSTORE JOB 0000PRV,'DAILY STORE PROCESSING',
//     CLASS=E,MSGCLASS=K,REGION=2048K,TIME=1440,
//     DATE='060101'
//*****************************************************************
//* DELETE FILES CATALOGED BY THIS JOB
//*****************************************************************
//DELETE EXEC PGM=IEFBR14
//DD1  DD DSN=FU01.FU01.STORE.DAILY,DISP=(MOD,DELETE,DELETE)
//*****************************************************************
//* UPDATE MASTER FILES AND CREATE REPORT (STEP 3 above)
//*****************************************************************
//DLYRPT  EXEC PGM=DAILYRPT
//SYSOUT  DD SYSOUT=*
//TRANSIN DD DSN=FU01.STORE.DAILY,DISP=SHR
//*
//MSTRFILE DD DSN=FU01.CUSTOMER.MASTER,DISP=MOD
//*
//NATLCUST DD DSN=&&NATLCUST,
//      DISP=(NEW,PASS,DELETE),UNIT=SYSDA,
//      DCB=(RECFM=FB,LRECL=100),
//      SPACE=(CYL,(100,40),RLSE)
//*
//
```

Figure 10. JCL Sample

Programs/Steps—COBOL, Sort, Assembler

Every job consists of a number of steps, and each step essentially does two things:

1. Indicates the program to be executed in the step. This is identified in the EXEC statement.
2. Sets up the environment (such as specifying input and output files, job priorities, and print file classes) within which the program should run.

Each step invokes a program that might be an application program written in COBOL, assembler, PL/I, or other language, or a system utility such as a sort step. The programs and utilities are the elements that perform the main business processing that manipulates files and data and produces reports and other types of outputs (for example, mailing bills or statements to customers).

When we talk of a job step being executed we are essentially saying that an application program or a utility is going to be run.

Programs

A large part of migrating batch applications involves the programs executed by the steps. The majority of these programs are written in COBOL, so COBOL compatibility and the ease of migrating COBOL code receive a lot of attention. Use of other languages such as assembler or PL/I needs careful consideration because migrating them to Windows usually means rewriting or converting the code to another language, such as COBOL.

The JCL in Figure 11 illustrates calling a COBOL program. In this sample, all the lines following from `//STEP003` make up the step.

```
//SALESJC JOB 'R. MALEK',CLASS=A,MSGCLASS=A,RESTART=*
//*
//JOBLIB DD DSN=NEOTEST.STEPLIB,DISP=SHR
//*************************************************************
//*  BUILD DATA FILE                        *
//*                          *
//* The PARM determines the number of records to create.    *
//* This can be a value up to 1,000,000 records for testing.  *
//*************************************************************
//STEP003  EXEC PGM=BLDFILE,PARM=500,COND=(0,NE,STEP002)
//SYSPRINT DD SYSOUT=*
//SYSOUT  DD SYSOUT=*
//OUTFILE DD DSN=FSW1.SALESDAT.DATA(+1),DISP=(,CATLG,DELETE),
//      SPACE=(CYL,10,,RLSE),
//      DCB=(RECFM=F,LRECL=66,BLKSIZE=0)
//*
```

Figure 11. JCL invoking a COBOL program

Sort

Sorting data is a common function of job steps. This is because it is usually quicker to arrange data into an appropriate order for the application process than to have the process fetch each record from an indexed file or database that does not have the data in the desired order. For example, sales may be stored in the order they are received, but statements may need to be produced by customer order or printed by mailing district; and reports may need to be produced by product item or summarized by sales representative. Using an efficient sort routine to order the data first can greatly reduce the overall time to produce reports and simplify application programming. Thus, supporting the batch sort utility and statements can be an important task of batch job migration.

Figure 12 shows JCL invoking a sort utility. This example shows two steps. One step begins with `//DELETE` and ends with the line of asterisks. It invokes the system utility IEFBR14 to delete any earlier versions of the file that will be created. The other step begins at `//INCLUDE`; it invokes the sort utility.

```
//PRV72500 JOB 0000PRV,'SORT INCLUDE',
//     CLASS=E,MSGCLASS=K,REGION=2048K,TIME=1440
//*******************************************************************
//DELETE EXEC PGM=IEFBR14
//DD1  DD DSN=FUO1.SORT.OUTPUT,
//     DISP=(MOD,DELETE,DELETE)
//*******************************************************************
//INCLUDE EXEC PGM=SORT,COND=(0,NE)
//SYSOUT  DD SYSOUT=*
//SORTIN  DD DSN=FUO1.SORT.IN,
//      DISP=SHR
//*
//SORTOUT DD DSN=FUO1.SORT.OUTPUT,
//      DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
//      SPACE=(CYL,(60,20),RLSE),
//      DCB=(MODELDCB,RECFM=VB,LRECL=623,BLKSIZE=0)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(30,15))
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(30,15))
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(30,15))
//SORTWK04 DD UNIT=SYSDA,SPACE=(CYL,(30,15))
//SORTWK05 DD UNIT=SYSDA,SPACE=(CYL,(30,15))
//SORTWK06 DD UNIT=SYSDA,SPACE=(CYL,(30,15))
//SORTWK07 DD UNIT=SYSDA,SPACE=(CYL,(30,15))
//SORTWK08 DD UNIT=SYSDA,SPACE=(CYL,(30,15))
//SYSIN DD *
 INCLUDE COND=(183,04,CH,EQ,C'EDUC',AND,
      (192,06,CH,EQ,C'101001',OR,192,06,CH,EQ,C'104000'))
 SORT FIELDS=COPY
 END
//
```

Figure 12. JCL invoking a sort utility

The above example illustrates the parameters for sorting an “in-stream dataset” (see `//SYSIN DD *` to `END`. More often in production JCL the sort parameters would be stored in a PDS library member referred to as a `CARDLIB` or `PARMLIB`. In that case, the `//SYSIN DD` statement would look more like the following.

```
//SYSIN DD DSN=MY.PROD.CARDLIB(SRT123),DISP=SHR
```

This method is not just for coding convenience—you cannot use in-stream datasets in PROCs.

Data Files and Databases

The purpose of batch jobs is to manipulate and/or report on the data held in data files and databases. IBM mainframes support a large variety of data files and databases, the most common of which are:

- VSAM (Virtual Storage Access Method)
  Often VSAM is equated with the indexed sequential files commonly used by COBOL, but it actually supports several formats that include sequential, relative, and indexed formats. The term “virtual” does not refer to specific attributes of the access method but served to tie the access method to other products in the late 1970s.

- QSAM (Queued Sequential Access Method)
  The most common format used to support COBOL sequential files.
- BSAM (Basic Sequential Access Method)
  An older sequential access method.
- DB2
  IBM's relational database.
- Oracle
  Oracle's relational database.
- IMS DB
  A hierarchical database developed by IBM, also referred to as a DL/I database.
- IDMS (now sold as Advantage CA-IDMS)
  Originally a network database management system, it has evolved into a relational database that continues to support the original network database applications.

When migrating applications it is not important to know whether a data file is supported by BSAM, QSAM, or VSAM, it is only important to know its organization—sequential, relative, or indexed—and whether the records are fixed or variable length. The Windows environment supports all of these organizations, but the differences between the mainframe file systems (VSAM, QSAM, BSAM) that can be important for tuning performance on the mainframe become irrelevant in the Windows environment.

Whereas databases are always long-lived, the life span of data files can vary from a few seconds to permanent. The following example illustrates some of the uses and operations on data files.

Each store in a national chain of tire stores might log the day's transactions in a fixed-length sequential file with records in the order in which the transactions were completed. At the end of each day, all files are uploaded to the central mainframe for producing reports for the company's national customers (such as national bus or trucking companies). The batch job that produces these reports might take the following steps:

1. Sort every transaction file by the customer number to create a temporary sequential file with the data in that order.
2. Merge all sorted store transaction files into a single file that contains all transactions for that day in order by customer number.
3. Run a program that reads the full transaction file and updates a master customer database (or VSAM indexed file) with each transaction or summarizes transaction totals and at the same time writes data to a new temporary data file that is in a convenient form for producing the daily summary statements sent to the national customers.
4. Run a program that uses the temporary data file to create the formatted reports (for print or electronic delivery) that will be sent to customers.
5. Send copies of the critical files to the company's backup devices.
6. Erase the temporary files. Because these are temporary, meaning they exist only for the duration of the job, this task completes automatically when the job finishes.

When creating the programs and batch job(s) to handle such tasks, consider the following points related to the handling of data files:

- Mapping the internal names for the data files used within the programs to the actual file names on disk.
- The maximum size of the files. Specifying an upper limit can be a useful way to discover when a program is writing records in an endless loop.
- The number and size of work files required for the sort operations.
- The files that should remain when the job terminates successfully and which of these should be backed up to an offsite location.
- The files that should remain if the job, or a step, does not conclude successfully.
- Whether any files should be set up with a generation data group (GDG)—a useful feature for automatically maintaining different versions of a file from one batch run to another.

Because mainframe resources are expensive, many organizations charge each group that uses the mainframe for the resources it used—including disk space. This forces a disciplined approach to file management: unnecessary files are not kept and batch jobs do not reserve more disk space than needed. These practices are useful in any environment, but their effect on the job control language is probably more pronounced for mainframe JCL than in Windows scripting languages. The following JCL data file definition statements illustrate this.

```
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(30,15))
```

This statement declares a temporary file to be used by the sort utility. It states that the file should need 30 cylinders of space and that if it needs more the additional space should be provided as a block of 15 cylinders.

```
//NATLCUST DD DSN=&&NATLCUST,
//      DISP=(NEW,PASS,DELETE),UNIT=SYSDA,
//      DCB=(RECFM=FB,LRECL=100),
//      SPACE=(CYL,(100,40),RLSE)
```

This data set definition declares the name &&NATLCUST, the data set's "disposition" at the start of the job (NEW); what to do if the step executes successfully (PASS on to next step) and if it does not execute successfully (DELETE). It specifies values for the file's data control block (DCB) and the space (SPACE) to be allocated for the data set, including the instruction to release (RLSE) unused space when the data set is closed. The file name prefix && indicates a temporary file that will be deleted at the end of the batch job.

Generation Data Groups (GDGs)

Mainframe JCL has the ability to catalog versions of data files through generation data groups (GDGs). For example, each time a batch job runs, a new version (or generation) of the data file is created. When the job finishes, both the starting and end versions of the data file have been saved. Using GDGs makes it easier to backtrack to a particular version of a data file and allows for older versions to be processed as part of the batch job.

At set up, the number of generation data sets in the generation data group (GDG) is specified. This is the number of versions that will be stored before the system starts replacing earlier versions.

Data sets within a GDG can be referenced using:

- An absolute reference, such as "use the fifth generation data set"

- A relative reference, such as “use the previous generation data set”
- All generations, such as “process all the generations”

Absolute references to GDG data sets use names that include a G0000V00 string (colloquially pronounced “goovoo”)—in which the zeroes are replaced by numbers. For example, an absolute reference to the fifth generation data set might look like:

```
DSN=TEST.GDG.DATASET.G0005V00
```

Relative references use subscripts to indicate an offset from the current generation. For example, `DSN=TEST.GDG.DATASET(0)` can refer to the next generation when the subscript equals 1 and the previous generation when the subscript equals -1. Relative references contain a subscript that gives a relative number. A reference to all generations omits the absolute or relative reference. For example:

```
DSN=TEST.GDG.DATASET
```

Because GDGs are used extensively, a batch job migration solution must propose how to replicate this functionality.

PROC—Prepackaged JCL

A frequently used feature of JCL is the JCL procedure, or PROC. A PROC is a collection of JCL statements that might be used by several batch jobs and that are coded in a single source file to be invoked when needed by each job. PROCs are stored in libraries searchable by the JCL system.

PROCs can accept parameters when invoked so that the behavior of the code differs depending on the parameters passed. For example, a simple parameter might provide the root directory of the file names to be used. However, PROC code parameterization can also be much more complex than this. Thus, the same PROC can be used with completely different data sets as long as the input file formats are the same.

```
//PROCSUB JOB 'PROC SUB',CLASS=A,MSGCLASS=A,RESTART=*
//*
//*
//STEP001  EXEC PROCSUB,PARMIN=500
//*
//PS001.INFILE DD DSN=SYS2.PROD.PARMLIB(PROCSUB),DISP=SHR
//
```

The above example invokes a PROC just as the example below does, substituting the provided values for INFILE and PARMIN.

```
//PROCSUB PROC
//*************************************************************
//*  BUILD DATA FILE                        *
//*                              *
//* The PARM determines the number of records to create.    *
//* This can be a value up to 1,000,000 records for testing.  *
//*************************************************************
//PS001  EXEC PGM=SUBPROC,PARM=&PARMIN
//STEPLIB DD DSN=FU01.STEPLIB,DISP=SHR
```

```
//SYSPRINT DD SYSOUT=*
//SYSOUT  DD SYSOUT=*
//OUTFILE DD DSN=FU01.SALESDAT.DATA(+1),DISP=(,CATLG,DELETE),
//      SPACE=(CYL,10,,RLSE)
//*
//INFILE DD DUMMY
//
```

The line that begins with INFILE above will be replaced with the information from the calling JCL.

```
DD DSN=SYS2.PROD.PARMLIB(PROCSUB),DISP=SHR
```

Because parameterized JCL is a powerful feature relied upon by many batch jobs, support of PROC is quite critical when migrating to a new environment.

Job Class

Job classes are used to group jobs by priority or function and determine which initiator(s) can be used to execute the job. Every job has an explicit or implicit job class. The job entry subsystem (JES), described below, is the part of the system that assigns jobs to initiators. Variations in job assignment are discussed under "Initiators" later in this section.

JES can be given a priority order for job classes so that jobs of a higher priority are always executed before jobs of a lower priority.

Within a job class, the PRTY option on the job card can be used to set priorities among jobs in the same class.

JES

JES stands for job entry subsystem and comes in two versions: JES2 and JES3. Although the names are similar, the control statements (JECL) that drive JES2 and JES3 are quite different. JES2 is used more commonly. JES3 is used by a smaller number of very large organizations. In summary, JES receives jobs, schedules execution, and spools output.

IBM's manual provides the following slightly more detailed description:

> *Simply stated, JES2 is that component of MVS that provides the necessary functions to get jobs into, and output out of, the MVS system. It is designed to provide efficient spooling, scheduling, and management facilities for the MVS operating system.*
>
> *In its most simple view, MVS divides the management of jobs and resources between the JES and the base control program of MVS. In this manner, JES2 manages jobs before and after running the program; the base control program manages them during processing.*
>
> *JES2 can specify the maximum time a particular job is allowed to run, the storage a job may consume, the number of copies of output a job can print, and the type of paper (form) on which the output prints.*[7]

Because JES is so closely associated with how the mainframe works, its control statements largely support performance tuning. Consequently, it is not necessary to emulate JES in a

[7] JES2 Introduction. IBM manual number SA22-7535-03, pages 1 and 3.

new environment or to support the JECL statements—other than being aware that these statements can be mixed in with the JCL of batch jobs, as illustrated in the following JCL extract (the JECL line is highlighted).

```
//NBREPRO JOB 63241,'FUJITSU NEOBATCH',CLASS=A,MSGCLASS=X
//    NOTIFY=&SYSUID,RESTART=*
//*
/*JOBPARM SYSAFF=MVSA
//*******************************
//* IDCAMS REPRO STEP       *
//*******************************
//REPRO  EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
```

Initiators

Initiators provide a way for the mainframe operating system to manage resources allocated to processes that may be running at the same time. A fixed number of initiators are set up, with instructions for the system to allocate a specific amount of machine resources (memory, processor time, DASD) to each initiator. The system then runs a single process within each initiator. Typical numbers of initiators range between 5 and 10, depending on the size of the mainframe system and organizational requirements.

JES manages job execution within each initiator based on the job classes assigned to each initiator and the priority order of the job classes. JES reads the class and priority (PRTY option, if present) specified in the job card, queues jobs in order of priority for the appropriate initiator, and starts the first job once the initiator becomes available.

Figure 13 illustrates this process for a system with four initiators:

- Initiator 1 supporting job class A
- Initiator 2 supporting job classes B and C
- Initiator 3 supporting job classes B and F
- Initiator 4 supporting job classes W, X, Y and Z

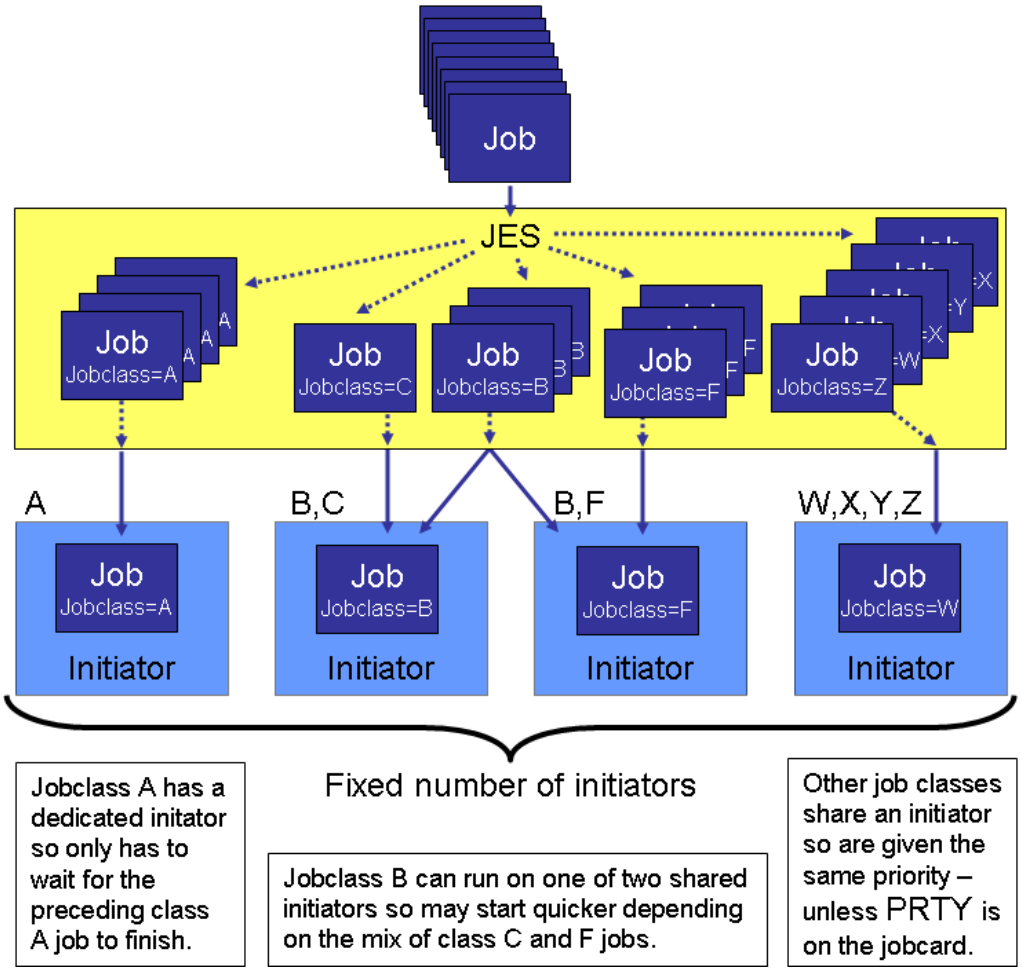


Figure 13. How JES uses initiators to manage job execution

Job Scheduler

At first glance, the job scheduler has a fairly simple role: it starts jobs on specified days or dates at the specified times. However, job schedulers also help manage a number of issues that surround this apparently simple process. For example:

- The days and times that jobs run may be determined by complex rules that account for public holidays or leap years or for working around the schedules of other batch jobs.
- The execution of some jobs may be dependent on the execution of other jobs or the arrival of data from third parties.
- Other scheduling contingencies involve postponing or continuing with a job run if a particular job failed and sending detailed notifications when jobs do fail.
- Most job schedulers now also support scheduling and management across multiple platforms.

These are just a few of the abilities common to leading job schedulers. All rules stored within the mainframe job scheduler represent a significant amount of business knowledge that must be preserved.