Technical Report

UCAM-CL-TR-4
ISSN 1476-2986

Number 4

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

The dynamic creation of I/O paths under OS/360-MVT

A.J.M. Stoneley

April 1975

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

TECHNICAL REPORT No. 4

THE DYNAMIC CREATION OF I/O PATHS UNDER OS/360-MVT

by

A. J. M. STONELEY

University of Cambridge Computing Service

Series Editor:

M.F.Challis
University of Cambridge
Computer Laboratory
Corn Exchange Street
Cambridge CB2 3QG
England.

April 1975

Summary

In a large computer it is often desirable and convenient for an ordinary program to be able to establish for itself a logical connection to a peripheral device. This ability is normally provided through a routine within the operating system, which may be called by any user program at any time. OS/360 lacks such a routine. For the batch job, peripheral connections can only be made through the job control language and this cannot be done dynamically at run-time. In the restricted context of TSO (IBM's terminal system) a routine for establishing peripheral connections does exist, but it is extremely inefficient and difficult to use.

This paper describes how a suitable routine was written and grafted into the operating system of the Cambridge 370/165.

A. J. M. Stoneley
May 1975.

Contents

1 Introduction

Under IBM's OS/360 the logical connection between a program and an I/O device is the 'data definition', or 'DD' for short. A program refers to a device indirectly, whether it be a real device, a pseudo device', a file, or whatever, by referring to the DD, which is identified by an eight character name, the DDname. The DD refers to the device or device type required. The commonest cases are that of the file, when the DD contains a pointer to the file, which it refers to by 'file name', and that of the 'pseudo printer', when the DD refers to a software entity which routes output to a printer.

OS was designed when the use of computers was very much less sophisticated than nowadays, and certainly before the advent of interactive terminal systems. It was an integral part of the design that the DDs required by a program should be set up solely by the use of the job control language, to be compiled in its entirety before any other part of the job was run. This compiled code was then re-interpreted by a part of the supervisor known as the 'initiator'. For reasons of efficiency, the initiator took advantage of the fact that all DD creation was done in between running user programs, and created the DDs required for a program all together and in parallel, and with sundry interrelations. The consequence of all this was that it was extremely difficult to graft on a method whereby a program could decide for itself the DDs that were required. Not only was there no system routine called 'create DD', but also there was no possibility of a user program creating job control language statements for subsequent execution within the current job. Furthermore it would have been extremely difficult to graft a new DD into the web of links between the existing ones.

With the advent of terminal systems, however, such a mechanism became necessary. IBM produced a mechanism for use by their terminal system, TSO, whose specification was roughly 'create DD'. Unfortunately, this was only available to programs running from a terminal, and could not be used by batch jobs.

There are other disadvantages of the TSO mechanism. In the first place, it is extremely expensive of both CPU and IO resources. For example, to create a DD establishing a connection with a terminal, a relatively cheap case, some six IO transfers of data blocks are absolutely inevitable. For a terminal system with over forty users this is a serious matter. Secondly, the user interface to this program is very complicated and difficult to use.

Thus for three main reasons did Cambridge embark on replacement mechanism; for homogeneity between terminal and batch working, for efficiency, and for comprehensibility.

2 Some reasons for wanting the facility

i. Any form of job control language will need a means of setting up DDs, for the prime object of a job control program is to provide the logical links between programs and files.

ii. There are many programs which require particular DDs only under special circumstances. For example, a program may require temporary disc space if the available core store is inadequate for the particular task in hand. If the DD is supplied by the job control program, it must always be provided, even if it may not be needed. This represents unnecessary work. It were better were the called program to create the DD only if it needed it.

iii. It often happens that a program knows better than its user just what its requirements are. The requirements may be fixed, in which case fixed instructions could be given to the user, or they could be variable, in which case the user could be asked to calculate the requirements. Either alternative is an unnecessary and unreasonable burden to place on a human when a computer is to hand to do such chores.

iv. On occasion, one does not know in advance the names of the files one wishes to read. The classic example is the program which makes reserve copies of all of a user's files (the 'dumper'). The program has to read the file directory and only then does it know what DDs are required.

v. A program often uses its DDs serially. For example, a compiler may read once through the source file, writing intermediate text onto a temporary file. Thereafter it has finished with the source file. At some later stage it will need an output file. It does not necessarily need all its files at once. Since the existence of a DD is a reservation of resources and consequently a load on the system, some saving can be made by only having a DD when it is actually needed.

Summary of objections to the IBM facility (DAIR & SVC 99):

i. The facilities are only available to terminal jobs.

ii. The cost of using the facilities is enormous.

iii. The interfaces to the facilities are cumbersome and tortuous.

3 Facilities provided

The new system, SVC 245 provides most of the facilities needed when managing DDs.

A DD may be created to permanent disc file, temporary disc space, terminal, printer, punch, 'internal reader', and 'dummy'. Disc files, permanent or temporary, may be created and deleted as necessary. The file characteristics may be specified just as from job control language. DDs may be deleted, and they may also be renamed.

One may interrogate the system to find out the current state of the DDs available to the program.

One feature is perhaps not obvious. One may create a DD to temporary disc space, write to it, and subsequently give this disc space a permanent file name. If a file of that name already exists, then the file directory is merely changed to point to the new file, and the old space is freed. The point of this is that one can, for example, edit a file to new disc space, refile this result with the original name, and be relatively safe against system crashes. The refiling operation merely consists of changing a pointer, and this can be done in an essentially indivisible fashion, so that there is no period of limbo when one has neither the old file nor the new file. This feature was inspired from the Titan operating system.

4 How it works

In this section a reasonable familiarity with OS/360 is assumed.

Most of the work of SVC 245 is done under the control of the initiator task. When a problem program requests some action, this request is chained onto the XJCB (extension job control block) and the initiator is POSTed. The initiator then itself calls SVC 245 to do the requested work. There are several reasons for this. In the first place, this is a natural way to serialise the operations. Secondly, the problem program is vulnerable to abends, both through its own asynchronous activities and through those of other problem tasks within the job step and also from the initiator, when, for example, CPU time limit expires. One cannot afford to leave the data structures in an inconsistent state. Setting SMC=STEP is no solution to this difficulty, since OS routines which are to be called also use this and end by releasing the hold prematurely. Thirdly, core must be obtained for control blocks, which will be freed by other tasks. Subpool 255 could be used for this purpose and this would sidestep this problem. Finally, the ENQ on the use of a file must be done by the initiator, since that is the only continuing task and since it already holds the ENQs for the files that OS knows about. The consequence of running as the initiator is that one absolutely must not abend.

The data structures needed to make OPEN, CLOSE and EOV work correctly are the TIOT and the JFCB. Entries in the TIOT are of variable length and will come and go as DD statements are created and deleted. In view of this, the TIOT is remade in a new segment of core every time a DD is created or deleted. (SVC 99 shuffles the TIOT in place). To prevent the TIOT from being thus moved while, for example, an OPEN is in progress, OPEN, CLOSE, EOV and SVC 245 all ENQ on SYSTIOT exclusively and over the tasks of the job step. The initiator does not do this because of the risk of a deadly embrace: the ENQ associated with the SVC 245 activity is done by the requesting task just before POSTing the initiator. Unfortunately, it is still necessary to reset DCBTIOT, the TIOT entry offset, in each open DCB every time the TIOT is remade. SVC 245 does not use SYSJOBQE. TIOT entries referring to dynamically created DDs are marked, and for such entries O/C/EOV calls a routine within SVC 245 instead of reading a JFCB from the jobqueue. This routine constructs a JFCB in the required place in core from data structures which are already in core.

The modules of O/C/EOV which read or write JFCBs have been modified to call SVC 245 when appropriate. The DD use counts are updated when the JFCB is written back.

After abend has failed to close a DCB by the front door, it merely deletes the DEB. In order that the use count may be updated at this point, its address is planted in the DEB at OPEN time.

The original TIOT created by OS is not freed by SVC 245. If a new TIOT is made, the old one is freed only if it was also made by SVC 245. At end of step any such TIOT is freed and the original is reinstated. The TIOT is regarded as a redundant dressing, set up for the benefit of OS. The sources of the data are the private structures of SVC 245.

At end of job, all DDs and temporary files are deleted.

4.1 The good points

Those who use the system have no difficulty understanding the interfaces. Indeed the interface is entirely described in an eight page document (see appendix). This is a substantial improvement over DAIR, which is described in about thirty pages.

The system is an order less costly to use than DAIR/SVC 99. The most significant saving is in disc transfers. For example, to get into and out of EDIT, the Cambridge text editor, would involve about sixty more data block transfers between core and disc (SYSJOBQE) when using DAIR than when using SVC 245. There is also a saving in the volume of code which in practice must be resident. The code of SVC 245 occupies about 7k bytes, as against some 26k for DAIR and the commonly used parts of SVC 99.

SVC 245 works equally well for a terminal job and a batch job, except that printer and punch are currently unavailable to the terminal job, and the terminal is, of course, unavailable to the batch job.

4.2 The bad points

The major drawback is the incompatibility. The system is available only at Cambridge, and programs written for export must make alternative arrangements. The prime users of the system are, however, the system programs, such as the local job control program, for which exportability is of less concern. On the other hand, standard facilities such as the catalogue and the disc space management routines are used internally.

The incompatibilities are also evident internally, inasmuch as a substantial modification of IBM code has been necessary. The fact that no further releases of OS/360 are expected has been a substantial encouragement.

A1 Specification of the Cambridge Dynamic DD System

A1.1 Dynamic DD statements

Dynamic DD statements can be created specifying the following devices: DUMMY, DISC, TAPE (not yet available), INTRDR (privileges required if online), SYSOUT (if the job is offline), or online terminal (if online).

After a dynamic DD has been created, there is an entry for it in the Task Input/Output Table (TIOT entries for such DD's are positioned after all other entries), and when it is deleted the corresponding entry is removed. DD statements created by OS (or IBM dynamic allocation) are inaccessible to the Cambridge system (except for information and renaming calls), and vice versa. Dynamically created DD's can be referenced in DCBs that may be be opened and closed in the usual way. The RDJFCB macro (see User's Reference Manual for restrictions on use) applied to such a DD will return a JFCB in which most of the useful fields are set. For an offline job, dynamic DD's persist across step changes - they are added on to the end of the TIOT constructed by OS for the new step. However, when using the PHOENIX command program all dynamic DD's are deleted on entry and all DD's created by the user program are deleted on return to the command program (named temporary files are not affected).

A1.2 Disc Datasets

Datasets on disc (referred to as 'files' throughout this section) accessed by dynamic allocation are of three types: permanent files which have full file titles of the usual format ('fully qualified names') up to 26 characters in length; 'named' temporary files whose names are specified by an ampersand ('&') followed by up to 8 characters; and 'anonymous' temporary files which have unspecified names. Files of all three types can be dynamically created and deleted. Named temporary files remain in existence for the duration of the job, unless explicitly deleted (see section A1.3.6). Anonymous temporary files are deleted when the corresponding DD is deleted (or, again, at the end of the job).

Permanent files on the public volumes are catalogued (and accounted against disc limit, etc.); any others are not. Note that, although permanent files can be accessed via JCL or via the Cambridge system, the temporary files created by the Cambridge system cannot be accessed by JCL, and vice versa. For example, a file &T created by JCL and a file &T created by the Cambridge system are two different datasets.

There is a facility (described in section A1.3.12) for obtaining a list of dynamically allocated temporary datasets in existence for a job at any one time. A list of permanent datasets may be obtained by means of the EXAMINE program, or by use of the LOCATE macro (see User's Reference Manual for restrictions on use). One may note in passing that it is (practically) impossible to obtain a list of all OS temporary files in existence for a job.

A1.3 Calls to SVC 245

All entries to the system are made by use of SVC 245. On entry, R0 is taken as a function code, while R1 should point to a parameter area in the user's region, whose contents and format depend on the particular function requested. A return code is given in register 15.

Wherever permanent file names are supplied by the user program as parameters to SVC 245, the first character of the name may be a dot('.'). In these circumstances the 'set user characters' or the userid is prefixed to the name.

A1.3.1 Create DD (R0=0)

R1 points to a prototype DD, as mapped by the macro SYS1.MACLIB(DDSTMT). It consists of a 12-byte header and one 52-byte section for each DD in a set of concatenated DD's (just one if no concatenation required).

Offset	Bytes & Alignment	Field name	Description
		DDROOT	
0(0)	8	DDNAME	Name of the DD to be created. It must be distinct from all already existing OS or dynamic ddnames in the current job step.
8(8)	1	DDCONC	Number of concatenations (i.e. number of DDDD's after the first), 0 if no concatenation. Must not exceed DDMCONC (currently 6).
9(9)	.1	DDFLGS	Disposition flags.
		1...	DDWRITE - should be specified if the dataset will be written to (equivalent to DISP=OLD versus DISP=SHR). It is an error to create a DD for a file for output that already has a DD, in the same or any other job; or to create any DD if it already has a DD for output. If DDWRITE is specified the DD may not be concatenated.

		.1..	DDMOD - causes the file to be positioned after the last record if opened for output. It does not imply DDNEW or DDWRITE.	
10(A)	. . 2	spare		
		DDDD	From here to the end repeated for each concatenation:	
+0(0)	1	DDDEVT	Device type as follows:	
		0	DDDUM	Dummy.
		1	DDDISC	Disc.
		2	DDTAPE	Tape (not yet implemented).
		3	DDPRINT	Printer (SYSOUT=A).
		4	DDPUNCH	Card Punch.
		5	DDINTRDR	Internal reader. Note: for DDPRINT, DDPUNCH and DDINTRDR, the job must be offline, DDWRITE must be specified, and special forms devices are not available. The latter restriction rules out the plotter and the paper tape punch.
		6	DDDEV	Privileged and unimplemented.
		7	DDTERM	PHOENIX terminal: the job must be online.
		8	DDN	Create DD from the same read-mode file as is referred to by the ddname in DDDDN. Both DD's must be for input, and the prior one not concatenated in any way.
		9	DDSHRP	This will eventually be SYSOUT=C. At the moment it is a single (pseudo-) printer (of type SYSOUT=C) acquired when needed and freed at the end of the job. Several DD's may be simultaneously attached to this printer, and output appears in the order that it is actually produced.
		10	DDINTPRT	Internal printer - unimplemented.
+1(1)	. 1	DDSPACE	Flags relevant to disc datasets.	
		1...	DDCYL - allocate space by cylinders (default is tracks).	
		.1..	DDRLSE - release unused space when closing dataset (relevant if old or new).	
		..1.	DDCONTIG - allocate one contiguous extent when allocating the file (if new). (Users are urged not to use this facility unless it is essential to do so.)	

		.... ..1.	DDOLDF - if the file is old, disregard the contents of DDDCB.
		1	DDNEW - unless DDWRITE and DDNEW are both set, the file must pre-exist. Otherwise it is created if it does not exist, but no fault is given if it does exist.
+2(2)	. . 18	DDDCB	'DCB' parameters - these 18 bytes are copied to offset 88 (decimal) in the notional JFCB.
+20(14)	2	DDFSEQ	File number for magnetic tape (not yet implemented).
+20(14)	8	DDMEMB	Member name in a partitioned dataset. Should be blanks if not set or not relevant.
+28(1C)	6	DDVOLSER	Specifies volume serial for magnetic tape or private discs. For disc datasets this should normally be spaces, in which case the public storage volumes will be used, old files being located from the catalogue (if permanent), and a suitable volume being chosen for new files according to a system algorithm. Users permitted to use private discs may specify them here, but in all cases the volume must be pre-mounted.
+34(22)	. . 2	spare	
+36(24)	8	DDDDN	DDname referenced when DDDEVT=8.
+36(24)	4	DDDSN	For disc (later, tape) datasets, a pointer to the name of the file. If this field is zero, the file is anonymous, otherwise it must point to a halfword containing the length of the name and followed immediately by that name. Temporary file names begin with '&'. Trailing spaces are ignored. 'Alias' names can be specified for permanent datasets (the interlock then works on this name). It is impossible to have simultaneously DD's for files with identical names on different volumes.
+40(28)	4	DDPRI	Primary space allocation (new disc datasets).
+44(2C)	4	DDSEC	Secondary space allocation (new disc datasets only - note not like JCL).
+48(30)	4	DDDICT	Dictionary allocation in blocks (new PDS's).

A1.3.2 Delete DD (R0=1)

R1 points to an 8-byte ddname. For a disc dataset, the file itself is not deleted by this operation, unless it is anonymous. Unit record devices are relinquished (except for 'shared printer's). It is an error to delete a DD while a DCB is still open for it.

A1.3.3 Delete all non-PHOENIX DDs (R0=11)

R1 is not relevant. This deletes all dynamic DDs not created by the command program.

A1.3.4 Rename DD (R0=4)

R1 points to the old ddname (8 bytes) followed by the new ddname (8 bytes). This operation can be done at any stage; the DD may have an open DCB. OS DD's can also be renamed, though they will still expire at the end of the step in the usual way.

A1.3.5 Rename DD changing read/write status (R0=9 or 10)

R1 points to the old ddname (8 bytes) followed by the new ddname (8 bytes). The old and new ddnames may be the same. The DD must be dynamic and represent a file: if R0=9 the DDWRITE flag is turned on if possible; if R0=10 it is turned off.

A1.3.6 Delete named temporary file (R0=2)

R1 points to the name (8 bytes, without the '&'). This is an error if any DD exists for the file.

A1.3.7 Delete permanent file (R0=14)

This call can be used to delete permanent files, provided that no dynamic DD exists for the file in this job and no other job is using the file in any way. R1 points to a parameter list mapped by SYS1.MACLIB(DRFPARM), as follows:-

Offset	Length	Name	Description
0	2	DRFPFLGS	The only allocated flags are privileged.
2	6	DRFPVOL	Volume serial. If the file is catalogued a volume name of all blanks should be given.
8	4	DRFPDSN	A pointer to the permanent dataset name (in the same format as for DDDSN in section A1.3.1).

A1.3.8 Rename temporary file as permanent file (R0=3 or 8)

For an anonymous file (R0=3) R1 points to an 8-byte ddname, followed by a pointer to a permanent file name. The DD must represent an anonymous disc file, and must not be open. The DD is left with DDWRITE off but is otherwise undisturbed. For a named temporary file (R0=8) R1 points to the temporary file name (8 bytes, without the '&'), followed by a pointer to a permanent file name.

In each case, the permanent file name has the same format as for DDDSN in section A1.3.1. The file is renamed as the permanent file (and is catalogued and accounted, if on the public volumes). Any pre-existing catalogued file of the same name is deleted. As a renaming operation, this should be fast and reduce the time during which a system crash can wreck the operation.

A1.3.9 Read information on one DD (R0=5)

R1 points to an 8-byte ddname. The information is placed in R0 in the following format:-

d0	Set if DD exists.
d1	Set if DD currently open.
d2	Set if DD was created by OS. If so, bits 1 and 3-31 are inapplicable.
d8	Named temporary file.
d9	Anonymous file.
d10	Writeable file (n.b. only set if file).
d15	File was not created by PHOENIX. (n.b. d8-d15 have undefined values, not necessarily zero, if the DD is not a disc dataset.)
d16	DD has been opened at least once.
d18	DD was not created by PHOENIX.
d24-d31	Device type, in the same format as DDDEVT in section A1.3.1.

A1.3.10 Read information on all DDs (R0=6)

This reads a list of ddnames and statuses. R1 points to a receiving area in the user's region, which should start with a halfword containing the maximum number of 12-byte entries the area will accommodate. The list is filled with 8-byte ddnames (only non-OS ones) and corresponding 4-byte statuses (as in section A1.3.9), following the halfword, which itself is set to the number of entries actually filled. If the area is not large enough, it is filled to capacity and R15 set to 19 (see below).

A1.3.11 Read status of named temporary file (R0=13)

R1 should point to the name of the file (8 bytes, without the '&'); a byte of status bits is returned as d8-d15 of R0. The bits have the same meaning as the corresponding bits described in section A1.3.9.

A1.3.12 Read information on named temporary files (R0=7)

R1 points to a receiving area in the user's region, which should start with a halfword containing the maximum number of 8-byte entries that the area will accommodate. The area is filled with 8-byte temporary file names (without the '&') following the halfword, which is itself set to the number of entries actually filled. If the area is not large enough, it is filled to capacity and R15 set to 19 (see section A1.4.1).

A1.4 Errors

A1.4.1 Return code format

Succesful completion of SVC 245 is indicated by setting R15 to zero. Otherwise an error code is placed in R15 in the following format:

d0	is used to distinguish between faults detected within dynamic allocation itself (bit unset) and errors returned from IBM's direct-access space management routines (DADSM) or equivalent (bit set). Note, however, that some DADSM return codes are treated specially, and result in SVC 245 faults, or in corrective action.
d8-d15	contains the concatenation number of the DDDD being complained of in a concatenation (0 if unconcatenated).
d16-d23	contains a secondary return code from SCRATCH for 'delete permanent file' only (section A1.3.7).
d24-d31	contains the SVC 245 or DADSM error code.

A1.4.2 SVC 245 fault numbers

1	Unassigned or privileged value of R0.
2	Parameter area not in user's key.
3	DD already exists, and should not.
4	DD does not exist, but should.
5	Too many dynamic DD's (current limit is 20).
6	DD in use (i.e. OPEN).
7	Too many concatenations (current limit is 6).
8	Concatenated DD for output.
9	Unknown device type.
10	Device type not allowed.
11	Device not available (privileged options only).
12	File in use (by this or another job).
13	File does not exist.
14	Volume not mounted.
15	Volume already in use (tape only).
16	Volume(s) full (disc).
17	File should be anonymous, but is not.
18	Terminal request by offline job.
19	Information list incomplete (sections A1.3.10, A1.3.12).
20	HASP pseudo-device request by online job.
21	Bad DD referred to by DDDEVT=8.
22	Use of identical file name on a different volume.
23	Too many such devices.
24	DDWRITE not specified for unit-record output.
25	CVOL not mounted (catalogue search failure).
26	Inappropriate device for requested action.