

AD-A086 487

WEAPONS SYSTEMS RESEARCH LAB ADELAIDE (AUSTRALIA)
DISCRETE/CONTINUOUS SIMULATION LANGUAGE.(U)
JUN 79 T J PACKER, J R COLEBY, T J GALVIN

F/G 9/2

UNCLASSIFIED

WSRL-0104-TR

NL

loc
AD
0096-187

END
DATE
FILMED
8-80
DTIC

WSRL-0104-TR

AR-001-687



12
P.S.
LEVEL

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

WEAPONS SYSTEMS RESEARCH LABORATORY

DEFENCE RESEARCH CENTRE SALISBURY
SOUTH AUSTRALIA

TECHNICAL REPORT

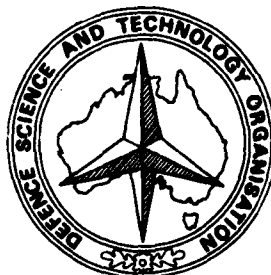
WSRL-0104-TR

DISCRETE/CONTINUOUS SIMULATION LANGUAGE

A USER GUIDE

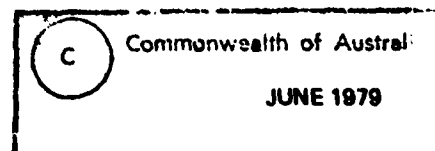
T.J. PACKER, J.R. COLEBY AND T.J. GALVIN

DTIC
ELECTE
JUL 1 1980
C



Approved for Public Release

COPY No. 38



ADA086487

DC FILE COPY

UNCLASSIFIED

12

AR-001-687

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

WEAPONS SYSTEMS RESEARCH LABORATORY

9
TECHNICAL REPORT,

14
WSRL-0104-TR

6
DISCRETE/CONTINUOUS SIMULATION LANGUAGE,

A USER GUIDE.

10
T.J./Packer, J.R./Coleby and T.J./Galvin

10, 94

DTIC
ELECTE
JUL 1 1980

SUMMARY

↙ This report takes the form of a user guide for the Discrete/Continuous Simulation Language (DCSL). This language has been developed to allow the convenient setting-up of simulations based directly upon models formulated in a graphical modelling language referred to here as GML.

The DCSL System Program is written in FORTRAN and hence DCSL is easily implemented on most modern machines.

APPROVED

FOR PUBLIC RELEASE

THE UNITED STATES NATIONAL
TECHNICAL INFORMATION SERVICE
IS AUTHORISED TO
REPRODUCE AND SELL THIS REPORT

Approved for Public Release

POSTAL ADDRESS: Chief Superintendent, Weapons Systems Research Laboratory,
Box 2151, G.P.O., Adelaide, South Australia, 5001.

UNCLASSIFIED

410919

2-1

01

DOCUMENT CONTROL DATA SHEET

Security classification of this page

UNCLASSIFIED

1	DOCUMENT NUMBERS	2	SECURITY CLASSIFICATION
AR Number: AR-001-687		a. Complete Document: Unclassified	
Report Number: WSRL-0104-TR		b. Title in Isolation: Unclassified	
Other Numbers:		c. Summary in Isolation: Unclassified	
3	TITLE		
DISCRETE/CONTINUOUS SIMULATION LANGUAGE A USER GUIDE			
4	PERSONAL AUTHOR(S):	5	DOCUMENT DATE:
T.J. Packer, J.R. Coleby and T.J. Galvin		June 1979	
		6	6.1 TOTAL NUMBER OF PAGES 85
		6.2 NUMBER OF REFERENCES:	
7	7.1 CORPORATE AUTHOR(S):	8	REFERENCE NUMBERS
Weapons Systems Research Laboratory		a. Task: DST 20/045	
7.2 DOCUMENT SERIES AND NUMBER		b. Sponsoring Agency:	
Weapons Systems Research Laboratory 0104-TR			
9	COST CODE:		
529342			
10	IMPRINT (Publishing organisation)	11	COMPUTER PROGRAM(S) (Title(s) and language(s))
Defence Research Centre Salisbury			
12	RELEASE LIMITATIONS (of the document):		
Approved for Public Release			
12.0	OVERSEAS	NO	P.R. 1 A B C D E

Security classification of this page:

UNCLASSIFIED

13 ANNOUNCEMENT LIMITATIONS (of the information on these pages):

No limitation

14 DESCRIPTORS:

a. EJC Thesaurus
Terms

Mathematical models
 Computerized simulation
 Computer programming
 Programming languages
 Computer programs
 Computer systems programs

b. Non-Thesaurus
Terms

Fortran
 DCSL
 GML

15 COSATI CODES:

1201

16 LIBRARY LOCATION CODES (for libraries listed in the distribution):

17 SUMMARY OR ABSTRACT:

(if this is security classified, the announcement of this report will be similarly classified)

This report takes the form of a user guide for the Discrete/Continuous Simulation Language (DCSL). This language has been developed to allow the convenient setting-up of simulations based directly upon models formulated in a graphical modelling language referred to here as GML.

The DCSL System Program is written in FORTRAN and hence DCSL is easily implemented on most modern machines.

Accession For
 NTIS GML&I
 DDC IAS
 Unannounced
 Justification

By _____
 Distribution _____
 Availability Codes _____
 Dist _____ and/or
 special

A

TABLE OF CONTENTS

Page No.

PART 1 - PRINCIPAL FEATURES

1. INTRODUCTION	1 - 2
2. THE VALUE OF A GRAPHICAL APPROACH	2
3. LANGUAGE CONVENTIONS	2 - 3
4. DISCRETE BLOCK TYPES	3 - 9
5. CONTINUOUS BLOCK TYPES	9 - 10
6. BLOCK INTERCONNECTION	10 - 11
7. AUTOMATIC SORTING FACILITIES	11 - 12
8. USE OF RESOURCES	12 - 13
9. LIMITATIONS OF THE CURRENT VERSION OF DCSL	13 - 14

PART 2 - MODEL ENCODING SYNTAX

1. GENERAL SYNTAX RULES	1 - 2
2. DISCRETE BLOCKS	2 - 22
3. CONTINUOUS BLOCKS	22
4. RESOURCE CONTROL	22 - 25
5. DATA INPUT	25

PART 3 - JOB AND OUTPUT CONTROL

1. ACCESS TO THE DCSL SYSTEM PROGRAM	1
2. JOB CONTROL	1 - 6
3. OUTPUT CONTROL	6 - 9
4. DECK STRUCTURE	9

PART 4 - SAMPLE PROBLEMS

1. INTRODUCTION	1
2. SAMPLE PROBLEM NO. 1 - A SIMPLE POISSON QUEUE GENERATOR	1 - 2
3. SAMPLE PROBLEM NO. 2 - A MODEL OF THE ACTIVITIES IN AN ENGINEERING CONSULTANCY	2
4. SAMPLE PROBLEM NO. 3 - A MODEL OF A MOTOR VEHICLE PRODUCTION PLANT	2 - 3

LIST OF APPENDICES

I	A GRAPHICAL REPRESENTATION OF THE DCSL SIMULATION SCHEME	1 - 2
II	THE USER DEFINED BLOCK (UDB) - OPERATION AND CODING DETAILS	1 - 4

INDEX

PART 1 - PRINCIPAL FEATURES

TABLE OF CONTENTS

	Page No.
1. INTRODUCTION	1 - 2
2. THE VALUE OF A GRAPHICAL APPROACH	2
3. LANGUAGE CONVENTIONS	2 - 3
4. DISCRETE BLOCK TYPES	3 - 9
4.1 The PROCESS block (PROC)	4
4.2 The EXTERNAL block (EXT)	4 - 5
4.3 The AND and OR blocks	5
4.4. The RANDOM DECISION block (RD)	5 - 6
4.5 The X-COUNTER (XC) and Y-COUNTER (YC) blocks	6
4.6 The FLIP/FLOP block (FF)	6
4.7 The INPUT SWITCH (ISW) and OUTPUT SWITCH (OSW) blocks	7
4.8 The TEST RESOURCE block (TR)	7
4.9 The IGNORE block (IGN)	7 - 8
4.10 The QUEUE CLEAR block (QC)	8
4.11 The USER DEFINED block (UDB)	8 - 9
4.12 The COPY block	9
5. CONTINUOUS BLOCK TYPES	9 - 10
6. BLOCK INTERCONNECTION	10 - 11
6.1 Connecting Lines	10
6.2 'Take-off' points	10
6.3 Interconnections implied by resource transfer	10
6.4 Defining the user's requirements	10 - 11
6.5 Discrete 'summing junctions'	11
7. AUTOMATIC SORTING FACILITIES	11 - 12

	Page No.
8. USE OF RESOURCES	12 - 13
9. LIMITATIONS OF THE CURRENT VERSION OF DCSL	13 - 14

LIST OF FIGURES

1.1 GML arrowhead convention	3
------------------------------	---

1. INTRODUCTION

Simulation of complex systems usually requires a computing approach. The rationale of system modelling and simulation is an elaboration in modern terms of the classical scientific method. It also embraces the concept of model formulation in terms of graphical algebra and the concept of the use of a simulation language technique to enact or solve the model relationships which characterize the nature of the physical system.

This report takes the form of a user guide for the Discrete/Continuous Simulation Language (DCSL) which has been developed to allow the convenient setting up of simulations based directly upon models formulated in a graphical modelling language referred to here as GML.

Many languages already exist which are designed for use in the simulation of complex systems. In fact, a common complaint is that there are so many simulation languages that standardization and interchange of work between analyst groups is being made difficult. Many analysts seek to have the existing languages adjusted or rectified, where they are deficient, rather than to embark on the development of new languages.

One might well ask, then, why DCSL was produced at all? There are two reasons.

Firstly, existing languages usually deal with the simulation either of discrete systems (e.g. SIMSCRIPT) or of continuous systems (e.g. DSL90 and CSMP), whereas a language capable of use for the simulation of mixed discrete/continuous systems is often necessary.

Secondly, very few languages which exist are designed to facilitate a graphical approach to model building and simulation. Those which do (e.g. GPSS) have not been developed with the 'real system' user - usually a layman where computers, simulation, and modelling are concerned - in mind, and so do not provide for his understanding of the graphical model of his system. Hence, a great deal of useful 'feedback' from the real system user is not available to the systems analyst. From experience, this feedback is necessary when dealing with complex systems.

DCSL is a mixed discrete/continuous language which is directly based upon graphical modelling; the graphics employed (GML) are such that laymen require very little instruction before they are able to read and interpret diagrams of very complex systems.

Many analysts ask why we need simulation languages at all and why not simply use lower-level general-purpose languages like FORTRAN, PL1 or BASIC to solve the model equations.

Again there are two main reasons.

The first is that the lower-level languages result in the user producing so much detailed code that he has great difficulty keeping track of it all. The overall structure of the simulation model is not sufficiently visible amongst the masses of detail.

The second is that in programming models of large systems in lower-level languages the analyst finds that it repeatedly becomes necessary to build detailed coded routines for functions, processes and facilities which should be already available on demand. This creates numerous opportunities for programming errors to occur, whereas a higher-level language provides most of these facilities in a reliable, well-tested state.

The end result of programming at too low a level for the size of model involved is often an enormous, badly organized program which is imperfectly understood and which makes any adaptation to changes in model structure a very

time-consuming and unreliable process. Also, when project staff changes occur, the new arrival finds it impossible to read the code and understand what it does.

Part 1 of this report is designed to permit a fast, readable appreciation of the principal features of DCSL in its 'Modification (Mod) Level 1' state. Parts 2 and 3, however, deal in more depth with the complexities of the language and are intended for use as a reference manual by the language user. Part 4 illustrates the use of the language by means of detailed descriptions of some example problems and their solution using DCSL.

Although DCSL is still under development, Mod Level 1 is already in a production state and is currently being used by several research agencies. However, as the earliest released form of DCSL, it is intended primarily for the simulation of purely discrete systems. To some extent continuous model sections may be included by the careful use of the User Defined Block (UDB).

Later releases of DCSL are expected to provide for purely continuous and mixed discrete/continuous models.

2. THE VALUE OF A GRAPHICAL APPROACH

With a properly designed diagrammatic approach, the modeller has a clear comprehensive picture of the model all through its development.

The model is at all stages clearly understood by users of the real system and by fellow analysts without a need for knowledge of formal mathematical, statistical or logical statement forms. This promotes valuable model criticism and 'feedback'. Such graphics allow condensation of model presentation in a concise yet mechanistic way, preserving the rigour of algebraic statement forms and ensuring that the structural links between model elements are visible. Thus, graphical elements may be regarded as building blocks which permit complex structures to be built up iteratively by trial and error if the situation demands. At each point in such a process, very little change in user code is needed to try out the model in DCSL simulations. Thus major structural amendments to simulation programs are a simple matter.

This approach ensures that DCSL structure statements can be set out in a systematic way so as to give a one-to-one correspondence with the model diagram, hence reducing the likelihood of errors.

Publication of completed work is simplified by the availability of virtually self-explanatory model block diagrams and corresponding DCSL structure statement listings.

3. LANGUAGE CONVENTIONS

The discrete system facilities of DCSL are based upon the concept of process-oriented modelling. This concept consists of separating the discrete aspects of any real system activity from its continuous aspects. That is, the logic dictating the start and finish of the activity and the occupation of the resources required during the activity, are treated as discrete. Any real variables involved in the modelling of the activity or its outcome are regarded as matters for handling by the continuous elements of the language. Further, the intention is that should both the discrete and the continuous aspects require to be simulated simultaneously, then the appropriate discrete and continuous language blocks should be interconnected so as to provide synchronisation. In this way, event control variables (orders), resource

state changes, activity durations and real variable outputs would all be provided for, if necessary, by the fully developed version of DCSL.

However, at the Mod Level 1 stage, DCSL is not yet fully developed and is primarily of value only in the simulation of discrete systems.

Nevertheless, it is important that the language conventions appropriate to the fully developed version should be adopted and used consistently from the outset. In this regard, it should be noted that GML dictates the use of solid black arrowheads to denote block diagram flow lines for continuous (real number) variables, whereas hollow white arrowheads denote the flow of discrete (1 bit) variables normally referred to as orders (see figure 1.1). Any variable which has a specific real value at all times during simulation is defined to be continuous, even though it may be switched. On the other hand, a discrete variable should be thought of as a one-bit order which is passed to an in-tray, represented by a white arrowhead, where it joins a first-in-first-out queue which may be empty or of any length. Queue members wait at the input of a discrete block until the block is ready to service them, one at a time.



Figure 1.1 GML arrowhead convention

Continuous flow lines may divide in the downstream direction at 'take-off points' in a manner synonymous with the 'take-off points' of conventional control block diagram algebra.

Similarly, discrete flow-lines may divide in the downstream direction at 'take-off points'. In the discrete case the convention used is that any order which reaches a discrete 'take-off point' is copied and one copy is passed along each of the branches originating at that point.

All other graphical language conventions are used in accordance with GML and where necessary are described at the appropriate point in the text.

4. DISCRETE BLOCK TYPES

There is no limit to how many times a given block type may be included in a model. Each block is distinguished from the others by a unique block name, which in Mod Level 1 consists of up to 4 alphanumeric characters, the first being alphabetic. Limitations on numbers of blocks arise when considering the specific computer configuration in any given implementation, since this defines the memory space available for the DCSL system program.

Before commencing the descriptions of the discrete block types available, it is important for the user to learn something of the method by which block interconnection is achieved when using DCSL.

In the structure statement for each block, the user only has to name the required output streams of the input blocks; DCSL automatically links the output streams of the block to the appropriate inputs at run time.

For further details on block interconnection refer to Section 6, and the examples in Parts 2 and 4.

4.1 The PROCESS block (PROC)

This is a discrete block and is used to simulate the starting up, duration and resource occupation of activities occurring in the system being modelled.

Three logical conditions must be satisfied before a process can start:

- (a) at least one order must be waiting in the input queue,
- (b) the process must not be running already, and
- (c) all of the resources required by the process must be available, whether exclusively or on a sharing basis.

When all three conditions exist and the process is to start, DCSL determines the duration of the process which is to occur and schedules its time of completion in an event calendar.

Process duration times are selected stochastically. Users may nominate any one of the four types of distribution function provided. The associated statistical parameters must be provided by the user as described in Part 2, Section 2.1.

4.2 The EXTERNAL block (EXT)

DCSL is primarily designed for the simulation of multi-process physical systems which are made up of a number of service units and/or plant functions each possibly requiring the use of non-dedicated resources. These service units may be similar and operating in parallel (e.g. bank tellers), thus providing a choice of service unit to be used to satisfy any demand (request for service). Also, the service units may, possibly, be categorized into groups, each group providing a different level of service. That is, in each case, the higher level service unit comprises several of the service units from the next level down. Therefore, any study of the dynamics of an organisation which exhibits both procedural and hierarchical properties, and also incorporates complex resource competition between and within the service units, is an appropriate task for the use of DCSL.

In this class of problem it is necessary to incorporate into the simulation model some representation of the sequence of demands for service which are placed upon the real system in any time interval to be simulated.

In fact, demand introduction is often a major part of the simulation analyst's task and its complexity is often underestimated. Figure I.1 of Appendix I gives a graphic impression of the magnitude of this aspect of the modelling task.

The EXTERNAL block provides a powerful and flexible means by which sequences of demands may be originated, introduced, allocated, routed or cancelled. This block also allows the modeller to redefine during run time the parameters of the service unit 'sub-models' to which demands are routed for satisfaction. That is, the EXTERNAL block can redefine the parameters of the set of one or more blocks used to model each service unit; (often the service units in a given system provide similar services and hence a model of the system will consist of, in the main, a number of

copies of the same sub-model).

In general, in addition to a DCSL structure statement (EXT statement in this case), the specification of the function of a particular EXTERNAL block requires the user to supply an associated XDAT data statement.

In its simplest form of usage the EXTERNAL block issues one copy of an order to each block in a predetermined distribution list at each time when a demand arises, as specified in the user-supplied data statement (XDAT) for the block.

This expedient serves to introduce demands for service into the users model and thus to initiate the running of the whole or, perhaps, selected autonomous parts(sub-models) of the model.

In the more general case, however, there arises a need for an external demand (order) to be issued to a service-unit which is capable of providing a variety of services. The type of service provided is controlled by the parameters of the service-unit sub-model.

This calls for the resetting of sub-model parameters by the EXTERNAL block at the time of demand issue, and allows the user to specify the details of the service called for. It also necessitates an examination by the EXTERNAL block of all those service-units suitable for the task to see which are free to provide the service. This ensures that demands are sent only to appropriate units.

The user may also specify a 'lifetime' for each demand such that it must be issued only during that interval (if at all).

Hence, the EXTERNAL block can be used to issue an unlimited number of demands each with a specific 'arise time' and 'lifetime', and each with its own dataset for conditional parameter updating if required.

Details of the usage of the EXT structure statement, the XDAT data statement and examples are provided in Part 2, Section 2.2.

4.3 The AND and OR blocks

These discrete blocks may have two or more inputs, depending on the user's requirements, but can have only one output stream each.

At any time during a simulation when the AND block finds one or more orders at each of its inputs simultaneously, it operates. This results in the instantaneous removal of one order from each input and the issue of one order via the output.

The OR block operates in a manner similar to that of the AND block except that it can be triggered by the arrival of an order at any input. Each arriving order is removed immediately and causes an order to be issued at the output.

It can be seen that these operations are directly related to the familiar AND and OR of normal logic. Hence, these blocks are used as logical conjunctions in the structuring of discrete block diagrams written in GML.

Examples are provided in Part 2, Section 2.3.

4.4. The RANDOM DECISION block (RD)

This block has a single discrete input stream and a single logical continuous output signal.

At any time when an input order arrives in the input queue an instantaneous random decision is made, and the output signal is reset to

accord with the result of the decision. Simultaneously, the order is removed from the input queue. The user must specify the value of a threshold parameter, p , in the range $0.0 \leq p \leq 1.0$. This may be specified as a real number in the RD structure statement or given a parameter name which appears in a DATA statement defining its real value (see Part 2, Section 5).

The block makes each random decision by generating a random number in the range 0.0 to 1.0 from a uniformly distributed population. If the random number is less than or equal to p , the output is set to 'off' ('false' or 0); if greater than p , the output is set to 'on' ('true' or 1),

i.e. (a) Sample $\leq p$, output OFF

(b) Sample $> p$, output ON.

Refer to Part 2, Section 2.4 for details.

4.5 The X-COUNTER (XC) and Y-COUNTER (YC) blocks

These two blocks operate similarly in counting their single input orders, comparing the count with a user-specified integer parameter, n , and making an output decision. They are often used for sequence control in iterative procedures.

The X-COUNTER has a single discrete input stream and a single discrete output stream. The input orders are collected until n orders are received at which time these n orders are removed and one output order is issued.

The Y-COUNTER has a single discrete input stream and two discrete output streams; output(1) being designated the 'NO' stream and output(2) the 'YES' stream. The input orders are directed to the 'NO' stream until n orders have been received whereupon the n th order is issued via the 'YES' stream and the event is re-initialised to allow the cycle to repeat.

Either type may be reset at any time during a run by means of the QUEUE CLEAR (QC) block.

See Part 2, Sections 2.5 and 2.6 for examples.

4.6 The FLIP/FLOP block (FF)

This block has two discrete input streams and a single logical continuous binary output signal. Orders arriving at the 'OFF' ('false') input set the output to 'OFF' ('FALSE' or 0); orders arriving at the 'ON' ('true') input set the output to 'ON' ('true' or 1). Prior to the receipt of any orders in either queue the block output is initially 'OFF'.

This block is used as a logic control element and is in effect capable of storing the value of a binary state variable for reference throughout the simulation. Its output signal is normally used as a logical switch control.

Refer to Part 2, Section 2.7 for examples.

4.7 The INPUT SWITCH (ISW) and OUTPUT SWITCH (OSW) blocks

These two blocks are controlled in a similar manner but one is the inverse of the other in that the INPUT SWITCH selects one of two inputs for transmission downstream whereas the OUTPUT SWITCH directs a single input to one of its two output streams.

Thus the former has two discrete input streams and only one discrete output stream, whereas, the latter has only one discrete input stream and two discrete output streams.

In both cases the 'OFF' (i.e. 'FALSE' or 0) state causes selection of the number 1 stream while the 'ON' (i.e. 'TRUE' or 1) causes selection of the number 2 stream.

Clearly, a queue of orders can accumulate at an input of the INPUT SWITCH block if that input has not been selected. If this input is selected at some time later in a simulation, the whole queue is instantaneously passed to the output. However, if need be, the input queue can be cleared at any time by means of the QUEUE CLEAR (QC) block.

Examples are provided in Part 2, Sections 2.8 and 2.9.

4.8 The TEST RESOURCE block (TR)

This is a discrete block with one discrete input stream and two discrete output streams. On each receipt of an input order, the status of the resource nominated by the user is checked. If this resource is free the input order is transmitted via the number 1 ('FREE') output stream and if it is not the order is transmitted via the number 2 ('BUSY') output stream.

The resource nominated may be a 'team', a team of teams, a 'pool', or a team of pools (see Section 8).

This block is normally used to route orders to a preferred section (often an autonomous sub-model) of the model when the appropriate resource is available, and to another section (or perhaps sub-model) when the resource is not available.

It should be noted that here the terms 'free' and 'available' exclude resources that are in use by sharers or that have been 'passed' (see Section 8) but not yet released.

See Part 2, Section 2.10 for further details.

4.9 The IGNORE block (IGN)

This is a discrete block with one discrete input stream and no outputs. It is able to debit nominated input queues by a nominated number of orders, even to the point of the debit exceeding the length of the existing queue so as to generate a 'negative queue'. This in effect provides for the case where the next *n* orders are to be ignored by the block whose queue is affected. One IGNORE block may so debit any number of user nominated queues and it will do so each time it receives an input order.

The user should appreciate that the IGNORE block is intended for use only in connection with PROCESS, AND, INPUT SWITCH and UDB blocks, these being capable of accumulating input order queues. It can, however, also be used to ignore any number of incoming orders for any other block type, but care must be taken in such applications to see that proper account is

taken of the order of block processing. Block sequence at run time is subject to re-arrangement by the built-in pre-run sorting facility. In general, where more than one sorted sequence is possible for nests of instantaneous blocks (i.e. non-PROCESS blocks), the sorter chooses the same order as that of the deck loaded by the user.

Refer to Part 2, Section 2.11.

4.10 The QUEUE CLEAR block (QC)

This is a discrete block with one discrete input stream and no outputs. Each time it receives an input order it clears all of the input queues in a user-supplied list. These queues are nominated by mention of the name of the block input to which they belong.

Also, when the name of a PROCESS block appears in the list, not only does this cause the input queue of that PROCESS block to be cleared, but also the PROCESS, if running, will be terminated immediately upon arrival of an order at the QUEUE CLEAR block and any attached resources (of any type) will be released.

The QUEUE CLEAR block is the only means available to the user by which a PROCESS block may be prematurely terminated.

The QUEUE CLEAR block may be used with the X-COUNTER and Y-COUNTER blocks in which case it re-initialises them each time it receives an input order.

Further details are provided in Part 2, Section 2.12.

4.11 The USER DEFINED block (UDB)

This block may be used to simulate any type of discrete and/or continuous element of a system being studied as it may possess up to 10 inputs and 10 outputs, any number of which may be discrete order streams or continuous signals.

The user may supply any number of different types of UDB, each coded in FORTRAN IV (or any other compatible language). Then any number of these blocks (with individual block names) may be incorporated in the model, just as is possible with system defined block types such as the PROC, EXT, AND, OR blocks etc.

The function of any particular type of UDB is entirely up to the DCSL user.

The method of creating a UDB is explained by example in Appendix II. Essentially, the user must provide a subroutine UDB as shown in the example. This subroutine may call further subroutines, one for each UDB type if the user so desires, or may branch to various sections of its own code to handle the various types.

The block type identifier (BTID) word in the structure statement is transmitted to subroutine UDB by the DCSL system program when it calls UDB to service each event during a simulation, so that the appropriate block type code may be executed.

The UDB subroutine is called at every simulation event time and for every UDB (block name) at each such time, whether orders have been posted to a given UDB or not and regardless of whether any real input signal changes have occurred. This gives the user complete control over what his UDB is to do under any circumstances whatsoever, involving any change in system state variables.

It is therefore important for the UDB routine to be designed so as to recognise immediate return situations and return as early as possible upon being called.

DCSL syntax is explained in Part 2, Section 2.13 as usual.

4.12 The COPY block

This is a simple discrete block with a single input order stream and a similar output. Each time an order or group of orders is received at the input the block immediately issues n copies of each order received, where n is a parameter supplied by the user as usual, either by typing its value into the structure statement or by using a parameter name in the structure statement and defining the parameter in a DATA statement.

Refer to Part 2, Section 2.14.

5. CONTINUOUS BLOCK TYPES

In this Modification Level of DCSL (Mod Level 1) the full complement of continuous facilities is not available.

The primary organization of the DCSL system program to allow for continuous simulation is incorporated and some structural block types are operational, but at present these facilities are not adequate for comprehensive continuous simulation. This of course implies that full-scale mixed discrete/continuous simulation is also not yet available.

Mod Level 1 DCSL incorporates only the following continuous blocks:

- multiple input summer,
- Cartesian to polar axis resolution,
- output switch,
- input switch,
- user-defined block,
- module control block, and
- compute control block.

In the next Modification Level of DCSL (Mod Level 2) it is planned to incorporate an extensive set of continuous blocks such as integrators, transfer functions, comparators, limiters, gains, pulse generators, function generators, sequenced time switches and signal generators, together with a variety of options for output printing and plotting control.

Integrators will incorporate active mode controls and a variety of numerical integration algorithms will be provided. A multiplicity of continuous models will be allowed to be loaded as separate 'modules' for any one run or experiment.

Within each 'module' integration step-length will be constant, although it may be varied from module to module, from one module operation to another during a run and between runs.

Some facilities for parameter-estimation cum optimization by iterative operation of a module are also envisaged.

However, at present no continuous simulation should be undertaken in DCSL except of course simply by means of the USER-DEFINED block (UDB).

6. BLOCK INTERCONNECTION

6.1 Connecting Lines

As explained in Section 3, in graphical models structure blocks are connected by lines. These lines are of two essentially different types.

One type denotes the flow of discrete orders from block to block and is terminated at a hollow, or white, arrowhead. The other denotes the flow of a continuous signal and terminates at a solid, or black, arrowhead.

Generally, discrete blocks have only discrete inputs, but some may have continuous controls or outputs; such as the discrete switches or the discrete flip/flop, for example.

Similarly, continuous blocks generally have only continuous signal inputs, but some may have discrete controls or discrete outputs.

6.2 'Take-off' points

A line connecting two blocks for the flow of discrete orders or continuous signals may pass through a 'take-off' point (as in block diagram algebra). The 'take-off' point is shown in the GML diagram as a black dot and only one line may flow towards it while any number may flow out from it.

The significance of the 'take-off' point is that each line flowing from it carries an exact copy of the flow arriving at it, whether it is a discrete order stream or a continuous signal. Thus, streams may be reproduced by take-off points when their flows need to be examined at various destinations.

Sometimes the term 'distribution list' is used herein to refer to the list of blocks receiving copies of the output stream of a block being discussed.

6.3 Interconnections implied by resource transfer

Sometimes implied feedback loops can be created by the modeller without a circuit of flow lines, when one process block requires a resource which is in use by another block downstream. Only on release of the resource can the upstream block re-commence running.

The possible ways in which the many properties of resources can be used in modelling are very numerous and will become self-evident after Section 8 has been studied.

6.4 Defining the user's requirements

The normal method for connecting blocks when coding a given GML diagram is simply to quote the name of the block from which an input is required when composing the structure statement for the receiver block.

However, some blocks produce several output streams and so when this type of block is to supply the input to the receiver block the specific output stream required must be indicated by using subscripts, for example

PROC FRED/JOE(3),C,1.5,1.5

In this case some supplier block named JOE has three or more output streams and receiver block FRED needs output number 3.

When a block is to be given several input streams (such as for the AND and OR blocks), the supplier block names may occur in any order and each may be subscripted or not as appropriate.

When a block is to receive a fixed number of inputs, the number being fixed by the DCSL rules (such as the INPUT SWITCH or the FLIP/FLOP blocks) then the input names must be entered in the order dictated by the logic of the model itself.

Control input names are usually required to be entered in a prescribed sequence position when coding the structure statement of the controlled block.

All such rules are made clear by example in Part 2.

6.5 Discrete 'summing junctions'

When interconnection lines are to be used to show the flow of discrete order streams in feedback paths, forming closed loops, the junction where an external input and/or the feedback stream(s) meet to flow down a forward path is usually provided in the form of an AND or OR block.

Thus the AND and OR blocks may be regarded in the same way as the summing junction in an analogue feedback circuit. They can, of course, also be used in many other ways.

7. AUTOMATIC SORTING FACILITIES

Users of continuous languages like CSMP or DSL/90 will be familiar with the concepts of block sorting and implicit loops.

Generally speaking a given structure may contain closed loops, some of which may not at first be evident to the modeller if his diagram is a complicated one.

Most modern high-level languages for continuous simulation provide an automatic sorting facility which arranges the blocks in a specific order for execution at run time. DCSL is no exception, the DCSL system program contains an automatic sorter both for the discrete blocks and for the continuous blocks.

Sometimes inexperienced users wonder why this is done as it may seem little trouble to arrange the blocks oneself into an order which seems either satisfactory or necessary.

Furthermore, the sorter program sometimes raises diagnostic objections to a coded structure on the grounds that it contains an 'implicit loop' or that a 'sort failure' has occurred.

It should be appreciated, however, that the need to detect such 'implicit loops' is imperative. A coded structure which contains such loops constitutes a logically incomplete definition of what the user wants the computer to do.

Thus, the sorter is not just provided to allow the user to be lazy about

the order in which he stacks his structure cards. It is there to detect ambiguities in the instructions for computation arising from the users statements.

An implicit loop has no particular starting point where computation may commence and the arbitrary choice of such a point will often lead to a different computed result each time the point is selected differently.

Generally any block with a defined initial state (or condition) such as an integrator or a process block (initially inactive by definition) may be safely assumed as a starting point. In fact, any block with a 'memory' or time lag is generally a legitimate point at which the sorter may break the loop and begin defining the initial computation sequence.

When no such block exists at any point in the loop in question, then sorting is not possible and an implicit loop exists. The user must supply further instructions, somehow, on where to begin the computation of that loop.

Such a situation usually denotes that somewhere in his model the user has created an approximation to the real system or is trying to solve its characteristic equations in an iterative manner with no initial conditions defined.

On detecting such a loop the sorter prints a 'SORT FAILURE' message and the user is expected to re-examine his GML diagram to rectify the fault.

When an implicit loop is intentionally used as a device for the iterative solution of a set of relations by successive approximation, this is possible only if a small delay (process block in the discrete structure or unit delay in the continuous structure) is inserted at the point where the iterative computation is to be commenced during simulation. The sorter will then detect the delay element and use it as a start-point.

Thus the onus is upon the user to check models for implicit loops but the sorter acts as a safety device in case he slips up.

In cases where several blocks might receive an order to operate at exactly the same time (either by the release of a resource they are all waiting for or by distribution of a copied order stream, one copy to each block, for example) then the order in which these potentially simultaneous block-starts will occur is determined simply as the order in which the user has stacked the relevant block definition structure statements.

This fact can be useful when a particular sequence is desirable for instantaneous blocks involving switching logic to control the transfer of orders between blocks.

8. USE OF RESOURCES

DCSL provides a powerful set of resource handling features summarised as below:

- (a) Resources may be called for by processes at various levels of call priority.
- (b) Resources may be shared between any number of processes nominated (as willing) by the user in a SHARE statement.
- (c) Any resource(s) may be passed exclusively down a procedural string (of processes) from one completing process to another further downstream and not yet running. By using a PASS statement the user may ensure

that such a string will not be stopped during its operation by the loss of resources to other resource users even when these have higher priority of call.

- (d) Resources need not be unary but may be defined as pools having any number of identical members by means of a POOL statement.
- (e) Resources may be declared to be 'teams' and each team is then allowed to be defined as any sized logical grouping of other resources in a candidate list.
Candidate resources in the list may themselves be declared to be further teams and these may be defined as any sized groupings of yet further resources in further candidate lists, for example

TEAM SET/2.OF.KIT,A,B

TEAM KIT/3.OF.I1,I2,J1,J2

The limit to the hierarchical depth is that a team which is itself a candidate for membership of a team, must not have candidates of its own which are also teams. This gives a three-tiered hierarchy.

(Note: this can in some respects be extended to four tiers by the use of the EXT structure statement which can also group resources in a similar way to define its resource set requirements for demand issue - see Part 2, Section 2.2)

- (f) Some of these facilities can be compounded together. For example, teams and pools may be called for on priority, shared or passed. Also, teams of pools are allowed. However, in the Mod Level 1 version of DCSL pools of teams are not allowed.

Examples of the use of each of these resource management facilities are given in Part 2, Section 4.

During a simulation, when the DCSL resource allocation algorithm is attempting to acquire resources for PROCESS blocks it examines the various means of resource acquisition in a definite order of preference. If the PROCESS block has not already received the resource because of a PASS statement, then the algorithm first looks to see if the resource is free (or if any member of its pool is free). If not, the algorithm checks to see whether the resource can be shared with its current user process. Failing this, the resource may have just been allocated to another process having a lower priority of call, in which case it is acquired by a priority take-over.

If any resource required by a process cannot be acquired by any of these means the process cannot be started until the situation changes.

It is often necessary to compile statistics on the use of resources by processes during a simulation. A facility for this purpose is provided by the RUSE statement which is described in Part 3, Section 3.

9. LIMITATIONS OF THE CURRENT VERSION OF DCSL

The major limitations of the current (Mod Level 1) version of DCSL are listed below:

- (a) as for any simulation language, the problem size available will always be limited by the configuration of the computer in any given implementation;
- (b) orders (sometimes known as transactions) have no attributes other than position in the model flow diagram;
- (c) System Defined Blocks (SDB's) cannot effect computed parameter changes for other blocks;
- (d) there is no provision for macros in the case of repeated sub-models;
- and (e) as mentioned in Section 5 above, the facilities for continuous simulation are not yet fully developed.

There are some other limitations associated with specific elements of the language. These are identified in the relevant sections of the guide.

PART 2 - MODEL ENCODING SYNTAX

TABLE OF CONTENTS

	Page No.
1. GENERAL SYNTAX RULES	1 - 2
1.1 Deck structure	1
1.2 General statement structure	1
1.3 Continuation lines	1
1.4 Content word separators	1
1.5 Comment statements and blank lines	1 - 2
2. DISCRETE BLOCKS	2 - 22
2.1 The PROCESS block (PROC)	2 - 6
2.2 The EXTERNAL block (EXT)	6 - 10
2.3 The AND and OR blocks	10 - 11
2.4 The RANDOM DECISION block (RD)	12
2.5 The X-COUNTER block (XC)	12 - 13
2.6 The Y-COUNTER block (YC)	13 - 14
2.7 The FLIP/FLOP block (FF)	14 - 15
2.8 The INPUT SWITCH block (ISW)	15 - 16
2.9 The OUTPUT SWITCH block (OSW)	16 - 17
2.10 The TEST RESOURCE block (TR)	17 - 18
2.11 The IGNORE block (IGN)	18 - 19
2.12 The QUEUE CLEAR block (QC)	19 - 20
2.13 The USER DEFINED block (UDB)	20 - 21
2.14 The COPY block	21 - 22
3. CONTINUOUS BLOCKS	22
4. RESOURCE CONTROL	22 - 25
4.1 Priority of call	22 - 23

	Page No.
4.2 Resource pools	23
4.3 Resource sharing	23 - 24
4.4 Resource passing	24
4.5 Resource teams	24 - 25
5. DATA INPUT	25

LIST OF FIGURES

2.1 The PROCESS block	4
2.2 Resource symbols	4
2.3 The modified Erlang distribution function	6
2.4 The truncated Erlang distribution function	6
2.5 An EXTERNAL block	8
2.6 The AND block	11
2.7 The OR block	11
2.8 The RANDOM DECISION block	12
2.9 The X-COUNTER block	13
2.10 The Y-COUNTER block	14
2.11 The FLIP/FLOP block	14
2.12 The INPUT SWITCH block	15
2.13 The OUTPUT SWITCH block	16
2.14 The TEST RESOURCE block	18
2.15 The IGNORE block	19
2.16 The QUEUE CLEAR block	20
2.17 A USER DEFINED BLOCK (UDB)	21
2.18 The COPY block	22

1. GENERAL SYNTAX RULES

1.1 Deck structure

The deck structure of a DCSL job is illustrated in figure 3.3.

The Job Control (DSYS, END, ENDM, EXPT, RUN, STOP, ENDJ) and Output Control (QUE, RUSE, TASK, DEBUG) elements of the deck are dealt with in Part 3.

Part 2 will describe the syntax and usage of the Model Structure (PROC, EXT, OR, AND, UDB, etc) elements, the Resource Control (POOL, SHRE, TEAM, PASS) elements and the Data Specification (XDAT, DATA) elements of the deck.

As illustrated in figure 3.3, within certain sections of the deck, the cards may appear in any order. Also, other section types may be repeated to allow modification of requirements from one run or experiment to another.

1.2 General statement structure

DCSL statements are divided into two fields. The first field is the 'type' field and is restricted to columns 1 to 4. This field must contain one of the DCSL statement types, unless it is part of the second or subsequent line of a statement (see Section 1.3 below). The second field is the 'content' field and is restricted to columns 7 to 72 (except in the case of the DSYS Job Control Statement, see Part 3, Section 2).

Within the content field block names, parameter names and resource names and numeric values will occur. The names may consist of up to 4 alpha-numeric characters but the first character must be alphabetic.

Also, within the content field the block names, parameter names, etc. are required to appear in the correct sequence position (not necessarily any particular card column position) as dictated by the syntax rules given below for each block type (see Section 2).

Definitions of valid statement types and valid contents may be found in the relevant sections below.

1.3 Continuation lines

DCSL allows any number of continuation lines for any one statement. Provided that the first line of the statement consists of a valid statement type and, that it is followed by a valid content field then each subsequent continuation line is required to have only a valid content field, i.e. the type field together with columns 5 and 6 are left blank.

1.4 Content word separators

The DCSL system program regards a number of separators as equivalent to each other - these include /,=*,; and \$.

1.5 Comment statements and blank lines

The user may insert comment statements anywhere in a deck to make the listing of the deck easier to understand.

A comment statement is signified by an asterisk in column 1 of each of its cards.

Blank cards may be inserted anywhere in the deck, for example, to space out sections in the listing.

Comments and blank cards are printed during deck listing, but are otherwise ignored.

2. DISCRETE BLOCKS

This Section provides the user with detailed descriptions of discrete block statement syntax together with diagrams showing the associated GML representation. Also, for each block type, a brief description of the operation is given.

It should be noted that in model structure diagrams two types of block interconnection lines are used; one for the passage of discrete orders and one for the passage of continuous signals. The former are denoted by their termination at a hollow arrowhead, the latter by their termination at a solid arrowhead (see various figures in Part 4).

2.1 The PROCESS block (PROC)

The principal features of this block are described in Part 1, Section 4.1.

Process duration times are selected stochastically. The user may nominate any one of the four types of distribution function and their associated statistical parameters which together define the population of duration times from which DCSL will randomly select the duration for each process run.

These types are:

- (1) constant,
- (2) rectangular,
- (3) normal, and
- (4) Erlang.

While the process is running its resources are occupied and are not available to other processes unless the user has nominated this particular PROCESS block as one which is able to share one or more of the resources as specified in a SHARE statement (see Section 4.3 and Part 1, Section 8).

When the user specifies a particular PROCESS block, he must assign a priority of call for each of the nominated resources. Then in a case where two or more PROCESS blocks are waiting for a particular resource to become available, the block with the highest priority value will acquire the resource first. If equal priorities have been assigned the PROCESS block occurring earliest in the user code will acquire the resource first.

Care must be taken in the assignment of such priorities to ensure realistic model behaviour. It is possible to create an impasse where

each of a set of blocks has highest priority on a different one of a set of resources needed by all blocks. Such a situation will cause a 'log-jam' in that no further activity will take place among these blocks. Such situations are possible in reality though unlikely and rarely occur by design.

A diagnostic message is printed in this situation.

There is no specific limit on the number or types of resources which may be called for by a PROCESS block.

When the time scheduled in the internal calendar for the end of the process is reached, the resources used by the PROCESS block are freed. (In the case of resources being shared, at this time they are checked automatically in case ownership needs to be transferred to a sharing process which is still running).

The process is then designated as not running and an order is issued and copied to all blocks on the output distribution list, that is, to those blocks which nominate this PROCESS block as an input.

Some notes on the use of PROCESS blocks are worthwhile at this point.

The question arises when designing the model, of how many PROCESS blocks to use in the description of a real activity or procedure. Should all stages of the activity be modelled separately, allotting one block to each, or could the whole activity be lumped into one block?

The answer to this depends on what the modeller wants from the model when it is in use. The PROCESS block should be considered as the smallest piece of real procedural detail to be represented and hence the block should be considered to be indivisible. That is, the modeller should plan not to interrupt the running of a PROCESS block once it begins. This is the working principle, at least. In fact, a PROCESS block run can be interrupted by the use of the QUEUE CLEAR block (see Section 2.12 and Part 1, Section 4.10). But this is intended for use only in rare situations where a major interrupt of some kind is required in the operation of the simulated system.

To answer the above question then, two main considerations are involved.

Firstly, in representing a procedure it may be required to account for the precise time of resource occupation. If some resources are only used for part of the procedure, then that part should be modelled by a separate PROCESS block from the rest, in order to preserve accuracy in the resource-use statistics collected by DCSL.

Secondly, if there is a need for an input or output of a discrete order to or from any intermediate stage of the real procedure, then the model must be divided into separate PROCESS blocks at that point.

For example, suppose a procedure consists of two operations in series, but the second is to begin only when both the first operation is complete and some other external order is received. Then the model would use one PROCESS block for the first operation, followed by an AND block (see Section 2.3) which receives the output of the first block and the external order, followed by a second PROCESS block.

Similarly, if some other section of the model needs to be notified of the completion of one of the stages in the procedure, then the modeller must divide the procedure into separate parts at that point to provide a PROCESS block output order copy as required.

It should also be borne in mind that when large numbers of PROCESS blocks are strung together in series, there is a strong tendency for the

duration of the string procedure to become more normally distributed in accordance with the Central Limit Theorem. Modellers should be wary of this in case so many blocks are used as to give unrealistic overall timing populations.

The graphical symbol used to represent a process is

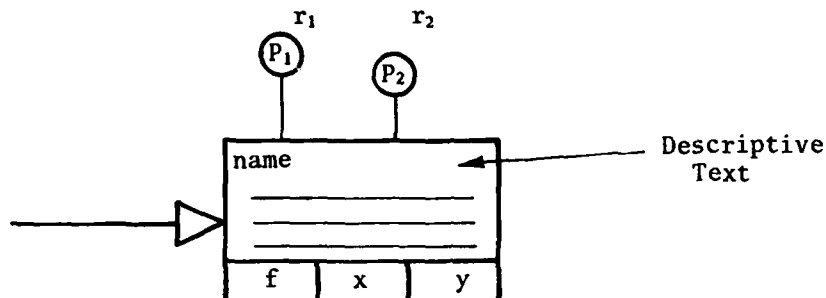


Figure 2.1 The PROCESS block

It should be noted here that, in figure 2.1, the round balloons connected to the top of the PROCESS block by vertical lines represent the unary (or single unit) resources required by the process, and that the integers contained in the balloons represent the priority of call the PROCESS block has on each of the resources. If the resources required are either pooled, sharable, to be passed, or teams (see Section 4), then modified versions of the basic symbol are used to represent them. These symbols are shown in figure 2.2.

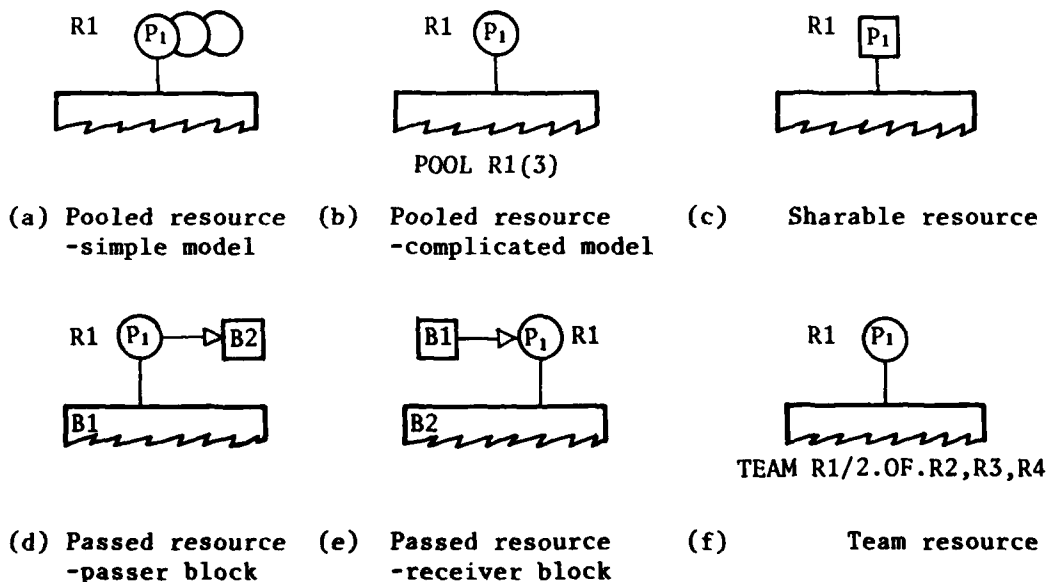


Figure 2.2 Resource symbols

In any diagram for which they are required, the legends POOL and TEAM

of figures 2.2(b) and 2.2(f) should be obvious and hence, in most cases, will appear away from the model structure (see diagrams in Part 4).

Further, the symbols of figure 2.2 may appear in any of the combinations permitted (as described in Section 4); for example, the passed resource in figure 2.2(d) may also be a shared resource, in this case the round balloon would be replaced by a square one.

The general form of the PROCESS block structure statement is

```
PROC name/input,f,x,y(,r1,p1,r2,p2,.....)
```

Here, name - is the PROCESS block name, e.g. BB4, SERV or LOAD;

input - is the input block name, e.g. BB3, WAIT or BB2(2) (the second output of the block BB2);

f,x,y - is the specification of the population from which the duration time is to be selected. Here, f indicates the type of probability density function and x and y are either real values or real parameter names which specify its parameters.

There are four types of distribution function available:

- (1) C,a,a - Constant - duration time equals a.
- (2) R,a,b - The process duration is a random variate with a rectangular distribution. The variate values are within the range (a,b).
- (3) N,m,s - The process duration is a random variate with a normal distribution having a mean of m and a standard deviation of s. The variate values are within the range (m-6s, m+6s).
- (4) En,m,c - the process duration is a random variate with a modified Erlang distribution having n degrees of freedom, a mean of m and a minimum delay time of c (see 'Note' below).

r₁,p₁,r₂,p₂,... - is an optional list of 'resource name, resource priority' pairs for the resources required by the process.

Priority values p₁,p₂ etc. are positive integer values or parameter names. The higher the value the higher the priority of call upon the resources r₁,r₂ etc. (see Section 4.1).

For example,

```
PROC B5/B4,E2,0.05,0.03,TLLR,1
```

(see Figure 4.1), or

```
PROC B5/B4,E2,P1,P2,TLLR,PR1
```

where P1, P2, and PR1 have values assigned to them in a DATA statement (see Section 5).

Note: the modified Erlang distribution function used in DCSL is not produced by truncation of a normal Erlang distribution function, but rather by shifting the origin by c in the negative x direction as below:

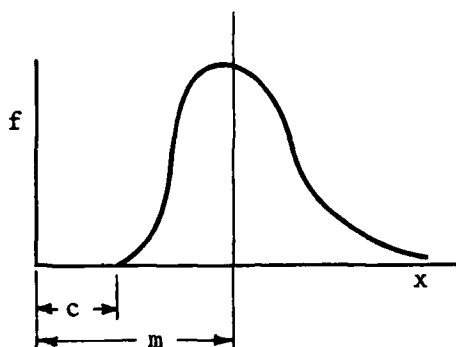


Figure 2.3 The modified Erlang distribution function

Some other simulation languages use a truncated Erlang function as below:

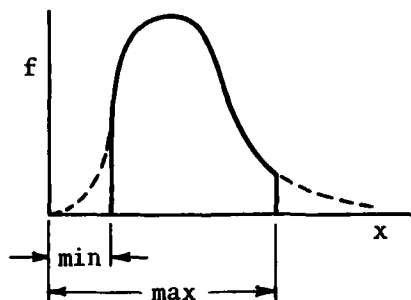


Figure 2.4 The truncated Erlang distribution function

The Erlang function used by DCSL provides a better description of real-world time processes. For example c often represents the delay in the human central nervous system and clearly the human activity time to be added to this should itself be Erlang distributed without truncation.

2.2 The EXTERNAL block (EXT)

This block is the means by which demands for service are originated, introduced, allocated, routed or cancelled; thus, in simple terms, this block initiates a model simulation run by introducing demands for service in accordance with the user's wishes with regard to, among other things,

the times of issue of the demands.

A brief description of the principle features of this block is given in Part 1, Section 4.2.

The EXT block may be used in a very simple form or in a very complex form, depending on which of the available features the user wishes to call into play.

In complex cases many powerful features are available and so the generalized form of the EXT and XDAT statements in symbolic notation are difficult to present clearly. The examples given here will show that the syntax is nevertheless simple to understand.

The generalized EXT structure statement form is

$$\text{EXT name}(/n.\text{OF}.r_1, r_2, \dots, r_k) \quad ,$$

the brackets indicating that the included proposition is optional, and where

name - is the EXT block name;

n - is the number of resources required to be 'available' for the block to issue any order;

.OF. - is literal; and

$r_1, r_2, \dots, r_k$ - is a list of names of resources, each of which is suitable to be counted as one of the n which must be 'available', and $k \geq n$.

A resource is considered to be 'available' if it is either free (not in use) or in use by a process which is designated as willing to share it.

Thus if the resource option is taken up by the user, that EXT will never issue any order except at times when at least n of the listed resources are 'available'.

To specify the time controls which the user may require to place upon the issue of orders the XDAT data statement is used. This statement is what gives most of the power and versatility to the EXTERNAL block as will be seen below.

Its generalized form is:

$$\text{XDAT } \{ \text{name}, \{ t(, \text{DELT}=\Delta t)(, p_0 \text{ list})(, r_1/(p_1 \text{ list}), (r_2/(p_2 \text{ list}), \dots (r_k/(p_k \text{ list})) \dots) \} , \dots \} , \dots$$

where the large braces signify that any number of like sets of contents may occur one after another, and the parentheses signify that their contents may be entirely omitted at the option of the user.

The other symbol definitions are,

name - is the name of the EXTERNAL block associated with any one set of contents of the outer large braces,

t - is the real number value of the arise time of a group of

simultaneous orders associated with the contents of one pair of inner large braces;

DELT= - is literal;

Δt - is the real number value of the lifetime of the group of simultaneous orders associated with the contents of one pair of inner large braces;

p_0 list - is a list of 'parameter name = real or integer value' equalities to be used to redefine model parameters as and when the associated group of simultaneous orders is issued,

r_i - is the name of the i th resource appearing in the associated EXTERNAL block's EXT resource proposition; and

p_i list - is a list of parameter equalities similar to that of the p_0 list but applied to the model as and when the associated group of orders is issued if and only if the i th resource was counted as available when the EXT proposition was satisfied to enable the EXTERNAL block to issue orders.

All this seems complex but is really very simple as will become clear from the examples below:

Example 1:

It must be appreciated that the EXTERNAL block in general may have $k+1$ output order streams. For example, consider the following EXT statement and its associated GML representation:

EXT name/3.OF.A,B,C,D (k=4)

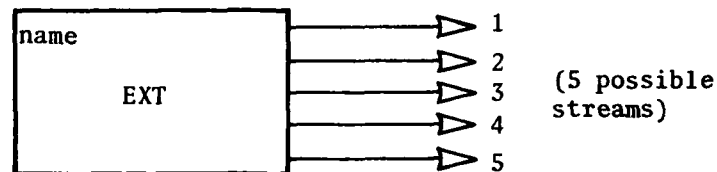


Figure 2.5 An EXTERNAL block

The 5th stream will always issue one order as soon as 3 of the 4 resources are available within the demand lifetime given in the XDAT statement (or none if this does not occur).

The 1st stream will also issue one order at this time but only if the resource A was one of the 3 available resources counted.

Similarly, the 2nd stream will issue one order at the same time but only if the resource B was one of the 3, and so on.

Also the p_0 list will be applied to the model if the 5th (generally, the last) output is able to issue an order in the time slot (t to $t+\Delta t$)

associated with that particular p_0 list.

And the p_1 list will be applied if output stream 1 issues an order at this time; p_2 list if output stream 2 issues an order at this time and so on.

Thus only $n+1$ of the $k+1$ streams can issue at the same time ($k \geq n$).

Example 2:

Consider, the simplest case possible

```
EXT    name
XDAT   name,t
```

This block issues one order at time t via its one and only output stream, regardless of resource states and does not change any model parameter.

Example 3:

The following are all legal examples of the use of the EXTERNAL block, building up in complexity. Note that the XDAT statement does not follow immediately after the EXT statement in the job deck but is placed as explained in Part 3, Section 4.

- (a) EXT EX12
XDAT EX12,0.1,11.3,29.2,101.0
- (b) EXT FILE/1.OF.A27
XDAT FILE,0.,DELT=.4,.2,.9,3.1,DELT=2.1
- (c) EXT LOAD
XDAT LOAD,99.,X=8.,JOE=50,101.,X=0.,Y=.6
- (d) EXT JOHN/1.OF.KIT
EXT MIKE
EXT FRED
XDAT JOHN,27.,DELT=9.,MIKE,63.2,FRED,88.
- (e) EXT ORDR/3.OF.A,B1,SET,GANG
XDAT ORDR,1.,DELT=10.,X=5.,A/Y=2./B1/N=8/SET/N=1/
19.,A/Y=2.5/B1/SET/GANG/N=8

- Notes:
- (1) Any of the named resources may be a pool or a team, or a team of pools, or a team of teams.
 - (2) DELT is meaningless (and ineffective) if no resources have been named.
 - (3) The DCSL language system program regards a number of separators as equivalent to each other - these include/,=*;: and \$.

- (4) Block names must begin after column 6.
- (5) Statements may be continued as long as columns 1 to 6 are blank on all continuation cards. Trailing blanks on a card are ignored.
- (6) Values of times and real parameters must be real numbers. Values of integer parameters must be integers.

Example 3(a) shows the fact that a simple EXTERNAL block (i.e. with no resource requirement) may issue an order unconditionally at each time in a list of times given in the XDAT statement.

Example 3(b) shows the use of the lifetime, DELT, for demand (order) introduction. The first demand is to arise at time 0.0 and exist until time 0.4. An order will be issued, at some instant between 0.0 and 0.4, only if the resource requirement can be met, i.e. resource A27 is to be available. The real numbers .2, .9 and 3.1 represent arise times for three more demands created by EXTERNAL block 'FILE' but at 0.2 and 0.9 the demands have zero lifetime. This means they will each cause the issue of an order only at those precise times (and only if A27 is available then).

The demand arising at 3.1 time units again is one with a lifetime and will survive until 5.2 time units. If A27 were never to be found available, none of these demands would cause the issue of an order.

Example 3(c) illustrates the fact that, even when no resources have been named as requirements, a p_0 list can be used to update (reset) model parameter values. Thus at 99.0 time units, EXTERNAL block 'LOAD' is to issue an order and at the same time the model parameters X and JOE are to be reset to 8.0 and 50 respectively. Also, at 101.0 time units another order is to be issued and the parameters X and Y are to be reset to 0.0 and 0.6 respectively.

Example 3(d) shows that a single XDAT statement may contain any number of data groups one for each of several different EXTERNAL blocks. Although each data group is a single data set in this example, any number of data sets (each at a different arise time) could be included in any data group.

Example 3(e) presents a more complex example of the EXTERNAL block usage with redundancy of resource candidates, more than one data set in a data group, lifetime usage and p_1 list, p_2 list, p_3 list and p_4 list usage. Note here in the second set, the use of null lists for p_2 list, p_3 list with p_4 list actively setting parameter N to 8 if resource 'GANG' is counted as one of the available 3 at time equals 19.0. This is done to inform DCSL to skip resources B1 and SET when looking to identify the last list as the p_4 list. XDAT statements are often many cards long, symbolizing the way real problems involve complex demand introduction in a model (see Appendix I).

2.3 The AND and OR blocks

The AND block must have two or more discrete inputs (no upper limit). At any time during a simulation, if every input queue is non-empty then the AND block issues one order via its output and removes one from each of its input queues.

The graphical symbol used for an AND block is

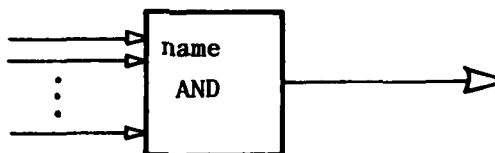


Figure 2.6 The AND block

and the general form of the AND block structure statement is

```
AND  name/input1,input2(,input3,....)  .
```

Here, name - is the AND block name;

input1,input2 - are the block names of the first two inputs; and

,input3,..... - is an optional list of block names of any extra inputs.

For example,

```
AND  GOLF/FRED,JOE,BERT,ALF  .
```

The OR block must have two or more discrete inputs (no upper limit). On receiving an order at any of its inputs, the OR block immediately removes the input order and issues one order via its output.

The graphical symbol used for an OR block is

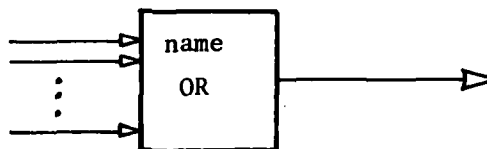


Figure 2.7 The OR block

and the general form of the OR block structure statement is

```
OR  name/input1,input2(,input3,....)  ,
```

the symbols having the same meaning as for the AND block.

For example,

```
OR  CARD/FRED,JOE  .
```

2.4 The RANDOM DECISION block (RD)

The RANDOM DECISION block has a single discrete input stream and a single logical continuous output signal.

The block makes each random decision by generating a random number in the range 0.0 to 1.0 from a uniformly distributed population. If this is less than or equal to a user nominated value, p , the output signal is set to 'OFF', if greater than p , the output signal is set to 'ON', i.e.

(a) $\text{sample} \leq p$, output OFF,

and (b) $\text{sample} > p$, output ON.

The graphical symbol used for a RANDOM DECISION block is

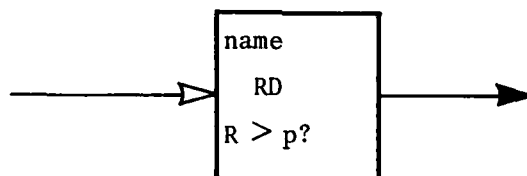


Figure 2.8 The RANDOM DECISION block

and the general form of the RANDOM DECISION block structure statement is

```
RD    name/input,p    .
```

Here, name - is the RANDOM DECISION block name;

input - is the input block name;

and p - is the real number or real parameter name which specifies the random decision test value.

For example,

```
RD    BB2/BB1,0.2    ,
```

```
or    RD    TEST/IN,P1    ,
```

where P1 has a value assigned to it by means of a DATA statement (see Section 5).

2.5 The X-COUNTER block (XC)

The X-COUNTER block issues a single output transaction whenever the number of transactions in the input queue is greater than or equal to a user nominated integer value, n . At this time, the X-COUNTER block removes n transactions from its input queue and thus resets itself ready for the next cycle.

The block is often used in the control of iterative procedures.

The graphical symbol used for an X-COUNTER block is

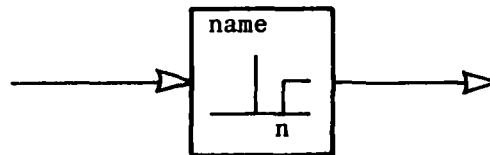


Figure 2.9 The X-COUNTER block

and the general form of the X-COUNTER block structure statement is

```
XC    name/input,n    .
```

Here, name - is the X-COUNTER block name;

input - is the input block name;

and n - is an integer value or integer parameter name which specifies the input order count index.

For example,

```
XC    END/IN,5    ,
```

```
or    XC    BB4/BB3,N1    ,
```

where N1 has a value assigned to it by means of a DATA statement (see Section 5).

2.6 The Y-COUNTER block (YC)

The Y-COUNTER block issues one output order through one of two discrete output streams for every input order received. All output orders are issued via OUTPUT(1) (the 'NO' output) until the number of input orders received is equal to a user nominated parameter value, n, at which time the nth input causes the Y-COUNTER to issue an output order via OUTPUT(2) (the 'YES' output) and the Y-COUNTER is reset ready for the next cycle. Thus, every nth order is effectively passed through to OUTPUT(2) while all other input orders are passed through to OUTPUT(1).

As for the X-COUNTER block, this block is also often used for the control of iterative procedures.

The graphical symbol used for a Y-COUNTER block is

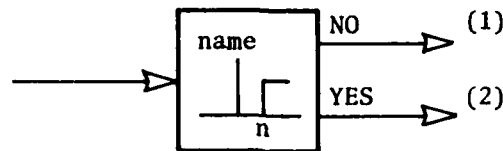


Figure 2.10 The Y-COUNTER block

and the general form of the Y-COUNTER block structure statement is

YC name/input,n ,

where the symbols have the same meaning as for the X-COUNTER. For example,

YC TERM/ENT,10 ,

or YC LOOP/INFL(3),N2 .

Note here that INFL(3) is the third output of the input block.

2.7 The FLIP/FLOP block (FF)

The FLIP/FLOP block has two discrete inputs, an 'OFF' or 'FALSE' input and an 'ON' or 'TRUE' input, and one continuous binary (i.e. logical) output signal. Orders arriving at the 'OFF' or 'FALSE' input set the output to 'OFF', i.e. the output signal has a real value of 0.0. Orders arriving at the 'ON' or 'TRUE' input set the output to 'ON', i.e. the output signal has a real value of 1.0. Prior to the receipt of any orders in either queue the block output is initially 'OFF'.

The normal use for the FLIP/FLOP block is as a logical switch control (see Sections 2.8 and 2.9).

The graphical symbol used for a FLIP/FLOP block is

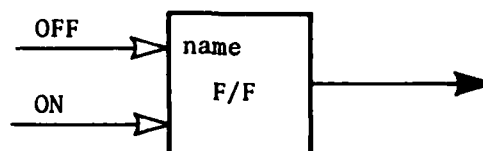


Figure 2.11 The FLIP/FLOP block

and the general form of the FLIP/FLOP block structure statement is

```
FF    name/input1,input2    .
```

Here, name - is the FLIP/FLOP block name;

input1 - is the 'OFF' or 'FALSE' input block name;

and input2 - is the 'ON' or 'TRUE' input block name.

For example,

```
FF    SW1/BL1,BL2    .
```

2.8 The INPUT SWITCH block (ISW)

The INPUT SWITCH block passes input orders from one of two inputs, the 'OFF' or first input and the 'ON' or second input, depending on the switch condition, to its output. The orders arriving at the input currently selected are issued immediately via the output. Any orders arriving at the other input are queued until the switch condition changes at which time they are instantly issued via the output.

There are two means by which the INPUT SWITCH block switch control may be achieved. The first method is by using the continuous output signals of blocks such as the FLIP/FLOP block, the RANDOM DECISION block or the USER DEFINED block. The second method is by using a real parameter value (in which case the solid arrow in figure 2.12 is replaced by a label: the parameter name).

If the continuous signal or the real parameter has a value of 0.0 then the switch is 'OFF', i.e. the first input is selected. If the continuous signal or the parameter has a value not equal to 0.0 (normally 1.0) the switch is 'ON', i.e. the second input is selected.

The graphical symbol used for an INPUT SWITCH block is

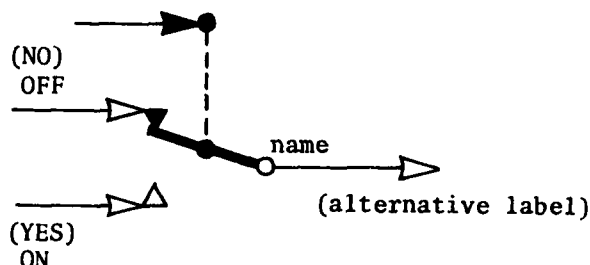


Figure 2.12 The INPUT SWITCH block

and the general form of the INPUT SWITCH block structure statement is

```
ISW  name/control,input1,input2      .
```

Here, name - is the INPUT SWITCH block name;

control - is (a) the name of the required continuous output of a switch control block, or

(b) a real parameter name, or

(c) a real number,

and if the control = 0.0 the switch is 'OFF',
if the control = 1.0 the switch is 'ON';

input1 - is the 'OFF' input block name;

and input2 - is the 'ON' input block name.

For example,

```
ISW  BL4/BL3,BL1,BL2      ,
```

```
or   ISW  BL4/P1,BL1,BL8  .
```

In the former example BL3 represents the name of a supplier block for which the first (or only) output is real (e.g. a FLIP/FLOP block).

In the latter case P1 represents a parameter name which can be defined in a DATA statement, and, if required, can also be reset several times during a run by means of an EXTERNAL block (see Section 2.2).

2.9 The OUTPUT SWITCH block (OSW)

The OUTPUT SWITCH block is controlled in the same way as the INPUT SWITCH block, but its operation is the inverse of that of the INPUT SWITCH.

Thus, the OUTPUT SWITCH block passes any orders arriving at its input to one of two outputs, the 'OFF' or OUTPUT(1) and the 'ON' or OUTPUT(2) depending on the switch condition.

The graphical symbol used for an OUTPUT SWITCH block is

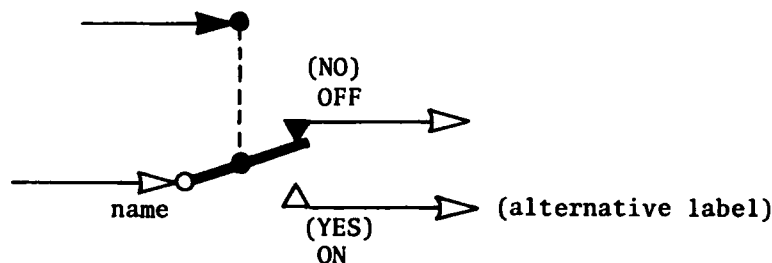


Figure 2.13 The OUTPUT SWITCH block

and the general form of the OUTPUT SWITCH block structure statement is

OSW name/control,input .

Here, name - is the OUTPUT SWITCH block name;

control - is (a) the name of the required continuous output of a switch control block, or

(b) a real parameter name, or

(c) a real number,

and if the control = 0.0 the switch is 'OFF',
if the control = 1.0 the switch is 'ON';

and input - is the input block name.

For example,

OSW SPLT/CON,INP .

2.10 The TEST RESOURCE block (TR)

The TEST RESOURCE block has one discrete input stream and two discrete output streams. On each receipt of an input order, the TEST RESOURCE block checks the status of the resource nominated by the user in the block structure statement. If the resource is free the TEST RESOURCE block transmits the input order via the 'FREE' or first output stream and if the resource is not free the input order is transmitted via the 'BUSY' or second output stream.

The resource nominated may be of any of the following types:

- a unary resource,
- a pooled resource,
- a pool member,
- a team,
- a team containing pools, and
- a team of teams.

However, the resource named must not be a pool of teams.

The TEST RESOURCE block is normally used to direct orders to a preferred section (often an autonomous sub-model) of the model when the appropriate resource is available, and to a second choice of section (or perhaps sub-model) or other disposal point when the resource is not available.

The graphical symbol used for a TEST RESOURCE block is

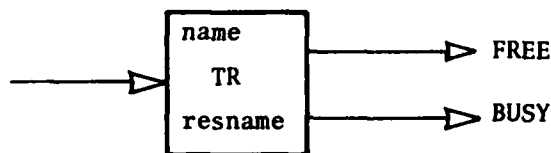


Figure 2.14 The TEST RESOURCE block

and the general form of the TEST RESOURCE block structure statement is

```
TR    name/input,resname    .
```

Here, name - is the TEST RESOURCE block name;

input - is the input block name; and

resname - is the name of the resource to be tested.

For example,

```
TR    FRED/JOE,FORK(3)
```

here, the third member of the resource pool FORK is to be tested to see if it is free or not,

```
TR    FRED/JOE,FORK
```

in this case, if FORK is still a pooled resource the TEST RESOURCE block will test the status of each member in turn, if any member is found to be free then the TEST RESOURCE block will issue an output via the 'FREE' output stream, if no member is free then an output will be issued via the 'BUSY' output stream.

2.11 The IGNORE block (IGN)

The IGNORE block has one discrete input stream and no outputs. This block is able to debit any number of nominated input queues by a nominated number of orders. This debiting operation extends to the case where the debit exceeds the existing queue length so generating a 'negative queue'.

For blocks with more than one input, their names may be subscripted to indicate which queue is to be debited.

The IGNORE block should be used only in connection with blocks which are capable of accumulating input order queues, namely the PROCESS,AND,ISW and UDB blocks. However, if necessary it can be used to ignore any number of incoming orders for any other block type, but the user must exercise great care in this case. A discussion of the precautions required when attempting to use the IGNORE block in connection with blocks other than those listed above is presented in Part 1, Section 4.9.

The graphical symbol used for an IGNORE block is

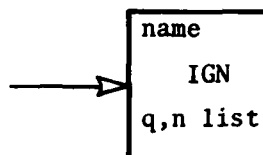


Figure 2.15 The IGNORE block

and the general form of the IGNORE block structure statement is

```
IGN name/input,q1,n1(,q2,n2,...) .
```

Here, name - is the IGNORE block name;
input - is the input block name;
q1 - is the name of the first input queue that is required to be debited;
n1 - is an integer number or integer parameter name which specifies the number of orders which are to be debited from input queue q1; and
q2,n2,... - is an optional list of any further input queues that are required to be debited and the corresponding debit sizes.

For example,

```
IGN BL8/BL7,BL2(3),4,BL2,P2 .
```

Here, the third input queue of block BL2 is required to be debited 4 orders and the first input queue of BL2 (the subscript is optional in the case of the first queue) is required to be debited P2 orders, where P2 has an integer value assigned to it in a DATA statement (see Section 5).

2.12 The QUEUE CLEAR block (QC)

The QUEUE CLEAR block has one discrete input stream and no outputs. On the receipt of each input order the QUEUE CLEAR block clears all of the input queues listed in its block structure statement.

For blocks with more than one input the block name may be subscripted so as to specify which particular queue is to be cleared. If all queues at the one block are to be cleared then each queue must be included in the QUEUE CLEAR list.

If the name of a PROCESS block appears in the list, not only does this cause the input queue of the PROCESS block to be cleared, but also the PROCESS, if running, will be terminated immediately and any attached resources will be released.

Some discussion on the use of the QUEUE CLEAR block in connection with PROCESS blocks is contained in Part 1, Section 4.10.

The QUEUE CLEAR block may also be used in connection with the X-COUNTER AND Y-COUNTER blocks in which case it clears their input queues and re-initializes their input order count to zero.

The graphical symbol for a QUEUE CLEAR block is

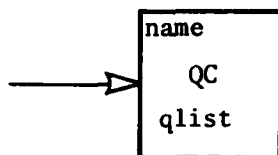


Figure 2.16 The QUEUE CLEAR block

and the general form of the QUEUE CLEAR block structure statement is

```
QC    name/input,q1(,q2,...)
```

Here, name - is the QUEUE CLEAR block name;

input - is the input block name;

q1 - is the name of the first queue to be cleared;

and q2,.. - is an optional list of any further queues that are required to be cleared.

For example,

```
QC    JIM/JOE,FRED,FRED(2),BILL,BILL(3)
```

here, block FRED has two inputs which the user wishes to have cleared, the first (subscript optional) and the second, block BILL also has two to be cleared, the first and the third.

2.13 The USER DEFINED block (UDB)

Part 1, Section 4.11 gives an explanation of the principals involved in the use of the UDB, while Appendix II contains a detailed description of the method of creating a UDB subroutine. The main purpose of this section is to provide the user with both the syntax of the block structure statement and a graphical symbol for a UDB.

It is important to understand the two classes of block in DCSL - the SYSTEM DEFINED block (SDB) and the USER DEFINED block (UDB).

They are similar insofar as each can have a number of different block types and each type can have a number of different occurrences.

The types are distinguished one from another by a block-type identifier (BTID) word in both cases, although in the case of the SDB its BTID is not readily accessible to the user. The different occurrences of a given type (SDB or UDB) are always distinguished from each other by their unique block names.

However, an important difference in the case of the UDB is that its BTID is user supplied and must appear in the UDB block structure statement, for which the general form is

```
UDB    name/btid/ndo,nco,nci*din1,din2...,cin1,cin2.....
```

where,

name - is the UDB block name;

btid - is the UDB block-type identifier word which must be an integer number greater than zero;

ndo - is the number of discrete output streams;

nco - is the number of continuous output signals;

nci - is the number of continuous input signals;

din1,din2.. - is a list of discrete input block names;

and cin1,cin2.. - is a list of nci continuous input block names (or real parameters).

Note: (a) the sum of ndo and nco must not exceed 10,

(b) the number of discrete input block names listed must not exceed 10-nci,

(c) nci must not exceed 10.

The graphical symbol used for a User Defined Block is

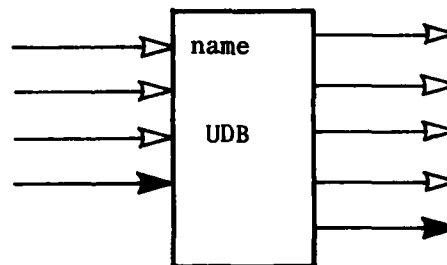


Figure 2.17 A USER DEFINED BLOCK (UDB)

2.14 The COPY block

The COPY block has one discrete input and one discrete output. On the receipt of an input order this block immediately issues n copies of the order, where n is an integer parameter value supplied in the block structure statement or a parameter name assigned a value in a DATA statement (see Section 5).

The graphical symbol used for a COPY block is

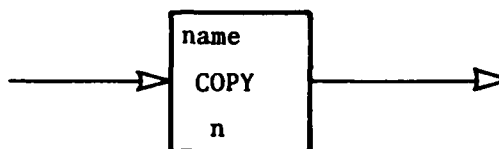


Figure 2.18 The COPY block

and the general form of the COPY block structure statement is:

`COPY name/n,input`

where, `name` - is the COPY block name;

`n` - is either an integer value or an integer parameter name specifying the number of copies to be issued;

`input` - is the input block name.

For example,

`COPY BB4/200,EB1` ,

or `COPY BB4/P2,EB1` ,

where P2 has a value assigned to it in a DATA statement (see Section 5).

3. CONTINUOUS BLOCKS

As explained in Part 1, Section 5, the current level of DCSL (Mod Level 1) does not readily permit continuous simulation except by means of the USER DEFINED block (UDB).

Plans for the future introduction of the continuous facilities are also discussed briefly in Part 1, Section 5.

4. RESOURCE CONTROL

DCSL has a powerful set of resource control features. This section provides the user with the syntax rules together with a brief description for each of these features.

For the graphical symbols used to represent these resource features, the reader is referred to the description of the PROCESS block given in Section 2.1.

4.1 Priority of call

PROCESS blocks may call for resources at various levels of call priority. The call priority is specified by the user in the structure statement for the PROCESS block, the call priority for each resource

required by the PROCESS block is coded as an integer value or parameter name following the resource name (see Section 2.1). The higher the value of the integer the higher the call priority.

For example,

```
PROC B3/B2,N,P1,P2,R1,1,R2,2      ,
```

```
or  PROC B5/B4,E2,P3,P4,R1,PR1,R3,1      ,
```

where the parameter PR1 is assigned a value in a DATA statement.

In this example blocks B3 and B5 might at some time require resource R1 at exactly the same time. If the user has specified a priority of more than 1 on R1 for B5 and a priority of only 1 for B3 then B5 will obtain the resource R1 in any such situation while B3 will have to wait at least until B5 completes its run.

4.2 Resource pools

Resources need not be unary but may be defined as pools having any number of identical members. Resources are declared to be pools by means of a POOL statement which has the following form:

```
POOL {res(s)}
```

where {res(s)} - is a list of resource names each with its pool size enclosed in parentheses. Valid DCSL separators (see Section 1.4) must be used between the list elements.

For example,

```
POOL RB1(3),RB2(2)
```

here, RB1 is a pool of size 3 (i.e. has 3 identical members) and RB2 is a pool of size 2.

The POOL statement must appear before any other statement which makes reference to any resource name (see Figure 3.3).

4.3 Resource sharing

Resources may be shared between any number of processes nominated (as willing) by the user in a SHRE statement. The statement is of the following form:

```
SHRE res,{pbn}
```

where res - is the name of the resource to be shared; and

{pbn} - is a list of the PROCESS block names of those processes which are permitted to share the resource. Valid DCSL separators (see Section 1.4) must be used between the list elements.

For example,

```
SHRE RB1,BB3,BB10,BB15,BB20 .
```

It is important to note that the SHRE statement can name only one resource. Thus, if the user wishes to nominate several resources to be sharable, he must supply one SHRE statement for each resource.

SHRE statements must appear at the correct place in the deck (see figure 3.3) which is after all of the model structure statements in a given discrete model.

4.4 Resource passing

Any resource(s) may be passed exclusively from one completing process to another further downstream and not yet running. This facility enables the user to ensure that the procedural string comprising the passer block, the receiver block and all the blocks between, will not be interrupted during its execution for want of the passed resource. However, the blocks in between will not have the passed resource directly available to them. If the modeller wishes to have the resource attached to any (or all) of the blocks in the string (thus enabling resource usage statistics for those blocks to be compiled if required), he must arrange for the resource to be passed to and from each of those blocks.

The user specifies that resources are to be passed by means of the PASS statement, this statement has the following form:

```
PASS {res,pbn1,pbn2}
```

where {res,pbn₁,pbn₂} - is a list of ordered triples which are composed of the name of the resource to be passed, the name of the passing PROCESS block and the name of the receiving PROCESS block. Valid DCSL separators (see Section 1.4) must be used between the list elements.

For example,

```
PASS R1,B1,B9,R2,B6,B10 .
```

The PASS statement must also appear at the correct place in the deck (see figure 3.3).

4.5 Resource teams

The use of the 'team' facility for resources has been discussed briefly in Part 1, Section 8.

As for the PASS and SHRE statements, the TEAM statement must appear only at the specified position in the deck (see figure 3.3) and a TEAM statement may not create a new resource name, whether it be the team name or a candidate resource. That is, each resource name included in the TEAM statement must have been created by an earlier statement in the deck, such as a PROC statement or an EXT statement.

Teams of resources may be called for by a process on priority, they may be shared and they may be passed. But they may not be pooled. However pooled resources are allowed to be candidates of a team as are other teams. There is a limit on the allowable hierarchical depth such that any team which is itself a candidate for membership of another team, must not have candidates of its own which are also teams. This gives a three-tiered hierarchy (but see the note given in Part 1, Section 8).

The form of the TEAM statement is

TEAM tn/n.OF.rn₁,rn₂,....

where tn - is the name of the team defined here;

n - is any positive integer value which specifies the number of resources required to form the team;

.OF. - is literal;

and rn₁,rn₂,... - is a list of at least n other resource names created already in PROC or EXT statements and which may each be a pool or a team (subject to the above limitation).

Note that only one team may be so defined in any one TEAM statement.

Further, there is a limit to the number of teams that the user may define. This limit is implementation dependent; the IBM 370/168 version of DCSL allows for 200 team definitions, while some other implementations only permit 100. A diagnostic is printed if the user attempts to define too many teams.

5. DATA INPUT

The DATA statement is the means by which parameter names in block structure statements are assigned initial values.

In the users deck, DATA statements must appear in the correct region (see figure 3.3 and the examples in Part 4).

The DATA statement has the following form:

DATA p₁=v₁,p₂=v₂,.....

where p₁,p₂,... - are parameter names which have been used in the block structure statements;

v₁,v₂,... - are the values assigned to p₁,p₂,... etc., and these must be integer or real according as their model usage dictates, e.g. parameter values for counter blocks must be integers while the duration time parameter values for PROCESS blocks must be real.

For example,

DATA X=0.1,JOE=9.99,FRED=1,IJ=10

Note that the method of preparing input data for the EXTERNAL block by means of the XDAT statement is fully described in Section 2.2.

WSRL-0104-TR

PART 3 - JOB AND OUTPUT CONTROL

TABLE OF CONTENTS

	Page No.
1. ACCESS TO THE DCSL SYSTEM PROGRAM	1
2. JOB CONTROL	1 - 6
2.1 The DSYS statement	1 - 4
2.2 The END statement	4
2.3 The CSYS statement	4
2.4 The ENDM statement	4
2.5 The EXPT statement	4 - 5
2.6 The RUN statement	5
2.7 The STOP statement	6
2.8 The ENDJ statement	6
3. OUTPUT CONTROL	6 - 9
3.1 The RUSE statement	6 - 7
3.2 The TASK statement	7 - 8
3.3 The QUE statement	8
3.4 The DEBUG statement	8 - 9
4. DECK STRUCTURE	9

LIST OF FIGURES

3.1 UNIVAC 1110 job control code	2
3.2 IBM 370/168 job control code	3
3.3 DCSL job deck structure schematic	10

1. ACCESS TO THE DCSL SYSTEM PROGRAM

Currently DCSL is not available for access in a terminal/interactive mode but is always run in batch mode.

The user has the usual options with his methods of

- (a) preparation of input
- (b) initiation of the batch job
- (c) selection of output device.

In other words, the user-code may be prepared in advance by card punching or by typing it in at a terminal and loading it onto disc or tape for access at job initiation time, or it may be typed in or loaded from cards at job initiation time - the job itself being initiated from a terminal or card reader respectively. Similarly the user may opt to have job output directed to a printer or to a disc or tape for later examination via terminal or for later printing.

The details of methods for accomplishing any combination of these, vary from one machine configuration to another so that it is impossible to provide a universal set of job control code. DCSL has been successfully implemented on a number of machine configurations including a UNIVAC 1110 and an IBM 370/168. In order to illustrate the principles for any configuration, figures 3.1 and 3.2 show the job control code required to initiate a DCSL batch job directly from cards with all output directed to the system default printer, for the cases of the UNIVAC 1110 and the IBM 370/168 computers respectively.

Users familiar with a specific job control language will have no difficulty in modifying this code to provide for the other job options and/or local configuration differences.

2. JOB CONTROL

There are several types of DCSL job control statements (see Part 2, Section 1) and each is explained here.

2.1 The DSYS statement

This statement defines the start of a discrete model structure definition section and must appear as the first DCSL statement in a job deck.

The DSYS statement is the only DCSL statement which does not require its content field to begin after column 6; in this case the content field must begin in columns 6 or 7.

The general form of the DSYS statement is

DSYS (n) (NOLIST)

where n is an optional 1 or 2 digit integer ending in column 7, and NOLIST is a literal option controlling the listing of the structure code. (Note: the parentheses here are purely to indicate that their contents are optional).

<u>CODE</u>	<u>REMARKS</u>
1 ↓ @RUN id,account no./code,firmcode	Initiate run.
@PASSWD pw	Account password.
@ASG,A lib/pw1/pw2	Assign library with DCSL, supply- ing READ and WRITE passwords.
@ASG,T 11.,F @ASG,T 12.,F @ASG,T 13.,F @ASG,T 14.,F	Assign temporary files to unit numbers 11, 12, 13 and 14.
@XQT lib.element	Request execution of absolute module of language.
DSYS : or @ADD lib.element ENDJ	Supply DCSL code on cards or call up file element containing DCSL code.
@FIN @@	Terminate job.

Figure 3.1. UNIVAC 1110 job control code

```

1
↓
//jobname      JOB      'account no./cost code',username,REGION=990K,CLASS=C
//LKED          EXEC     PGM=IEWL,PARM='MAP,LIST,LET,SIZE=301000'
//SYSLMOD       DD       DSN=&GOSET(MAIN),DISP=(,PASS),UNIT=SYSDA,
//              SPACE=(CYL,(1,1,1)),DCB=BUFNO=1
//SYSLIB        DD       DSN=libname,DISP=SHR
//              DD       DSN=SYS1.FORTLIB,DISP=SHR
//USERLIB       DD       DSN=libname,DISP=SHR
//SYSUT1        DD       DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(120,120),,,ROUND),
//              DCB=BUFNO=1
//SYSPRINT      DD       SYSOUT=A
//SYSLIN        DD       DDNAME=SYSIN
//SYSIN         DD       *
                INCLUDE USERLIB(DCSL1)
//GO            EXEC     PGM=*.LKED.SYSLMOD,COND=(4,LT)
//FT05F001      DD       DDNAME=SYSIN
//FT06F001      DD       SYSOUT=A
//FT07F001      DD       DUMMY
//FT01F001      DD       UNIT=SYSDA
//FT03F001      DD       UNIT=SYSDA
//FT04F001      DD       UNIT=SYSDA
//FT10F001      DD       DUMMY
//SYSIN         DD       *
DCSL CODE { DSYS
           :
           ENDJ
           } or //SYSIN DD DISP=SHR,DSN=datasetname
//              //
                REMARKS

```

Assign temporary or
dummy files to unit
numbers 1,3,4,7 and 10.

- (1) THE JCL above is usually supplied as a catalogued procedure by the installation.
- (2) libname is the name of the partitioned dataset containing the individual load modules of the DCSL system program subroutines.
- (3) datasetname is the name of the dataset containing the users DCSL code in card image form.

Figure 3.2. IBM 370/168 job control code

The usage of n and NOLIST is as follows:

- (a) n = blank - no table dumps are given;
- (b) n = 1 - a number of DCSL system tables are dumped after conversion of the user code;
- (c) n = 2 - the same tables are dumped after sorting;
- (d) n = 4 - the tables are dumped after translation;
- (e) n = 8 - the tables are dumped after loading;
- (f) n = the sum of any two or more of the above integer values - each dump from the appropriate integer, (b) to (e) above is given;
- and (g) if NOLIST is included, starting in column 9, then the discrete model structure code is not to be listed, whereas if NOLIST is omitted, it is to be listed.

The above description of the dumps available is necessarily brief. The user should nevertheless note that these dumps can be very useful to him. The table dumping facility was originally incorporated as a language development tool and not for the purposes of the user.

However, in the early stages of loading and checking any large model, this device can often most readily show up errors in the user code. For example, incorrectly defined resource names or block names may appear in the parameter table (DPT). The user is therefore advised to scan the table dumps in the initial testing stage to verify that block names appear in the discrete block table (DBT), resource names in the resource symbol table (RSYMB) and discrete parameter names in the discrete parameter table (DPT).

2.2 The END statement

This statement is used to define the end of a discrete model section. In future versions of DCSL the END statement will also be used to define the end of continuous model sections.

There is no content field associated with this statement.

2.3 The CSYS statement

This statement may be omitted in Mod Level 1 DCSL jobs, but in Mod Level 2 it will provide a similar function to that of the DSYS statement in respect to each continuous model supplied.

2.4 The ENDM statement

This statement defines the end of a complete discrete/continuous model structure definition region in the job deck.

As for the DSYS statement the ENDM statement may include a NOLIST (literal) beginning in column 9, which if present will suppress the listing of any further user statements down to the ENDJ statement.

2.5 The EXPT statement

This statement has the general form

EXPT (RNO=x)(,NRUN=y)

where x - is a suitable initiation seed for the random number generator in use, e.g. on the IBM 370, x should be an odd integer of 9 or less digits; if RNO is not specified by the user, the default value 25252525(HEX) is used.

and y - is the number of runs required if a Monte Carlo experiment is called for to examine the statistics of the population of model outputs over a series of trials; if y = 1 no such statistics are given.

Note that if the NRUN equality is omitted a single run will result.

Both equalities are optional, and there is no prescribed order in which they must be coded if both are used.

2.6 The RUN statement

This statement determines the end of each block (or set) of data and control statements associated with a particular experiment (either a single run experiment or a Monte Carlo experiment). The same model may be run several times in the one job, each time with a different data and control statement set, by providing a RUN statement after each set.

Following each RUN statement the data and control statements may be modified, or not, as follows:

- (a) the DATA statement - a new set of DATA statements (see Part 2, Section 5) may be inserted, however, only redefined parameters need be so inserted as the data facility operates in an overwrite mode;
- (b) the XDAT statement - no modifications are permitted, but new statements may be added for EXTERNAL blocks for which there is no previous XDAT statement;
- (c) the EXPT statement - both parameters of this statement (see Section 2.5) may be modified;
- (d) RUSE, QUE, and TASK - new names may be added to any existing lists (see Sections 3.1 to 3.3) but names cannot be removed; a request for Monte Carlo output only, can be made, but a previous request cannot be negated; the integer parameter in the RUSE statement can be used as usual with a new list; and
- (e) the DEBUG statement - new names may be added to an existing ALOC list (see Section 3.4), but names cannot be removed; the user may use either, or both, of the parameters POST and CALR (also see Section 3.4) to "turn on" their associated outputs, but he cannot turn off their output.

There is no content field associated with the RUN statement.

2.7 The STOP statement

This statement defines the end of the usage of the previously defined model(s) together with any previous data. After a STOP statement a new model may be defined and further experiments run, all in the same job.

The STOP statement has no content field.

2.8 The ENDJ statement

This statement defines the end of the whole DCSL job. It is the last DCSL statement in the job deck.

The ENDJ statement has no content field.

3. OUTPUT CONTROL

In this section, as previously, parentheses indicate that the contents are optional. And unless stated otherwise, the order in which the elements of the statements must be coded is as given in the general statement forms.

3.1 The RUSE statement

The RUSE statement allows the user to specify a list of resources for which usage statistics and individual block utilization history is required.

The run-by-run statistics printed for each resource are the resource name, the pool member number (always 1 for a unary resource), the number of times the resource was used, the total time in use, the mean time in use, and the standard deviation of the time in use. For example, see figure 4.6 in Part 4.

The run-by-run individual block utilization history output comprises the resource name, the pool member number, the user block name, the usage start time, the usage finish time, a label which indicates either that the block commenced its run sharing the resource, or that the block was a receiver of a passed resource (if a block is at the same time both a sharer and a receiver the label will only show the receiver status), and - if the block is a receiver - the time at which the resource was passed. Again, see figure 4.6.

The resource usage statistics for each resource provided at the end of a Monte Carlo experiment is composed of the resource name, the pool member number, the mean number of uses per run, the mean of the total use time per run, and the standard deviation of the use time per run (see figure 4.6).

The statement has the following form:

```
RUSE (MCLO,)(n,){res} ,
```

where MCLO - is optional. If it is included then all statistics requested by the statements RUSE, TASK and QUE are only provided at the end of a Monte Carlo experiment (or more than 1 run) in a condensed form based on the number of runs. If MCLO is not specified then the user obtains run-by-run statistics together with the Monte Carlo statistics.

It is stressed that if MCLO appears in any or all of the statements RUSE, TASK or QUE the Monte Carlo output is the only output obtained.

n - is an optional integer and requests that only the following n resources in the RUSE statement are to have the detailed block utilization history compiled. If n is not present, then all the resources listed in the RUSE statement will have detailed usage histories compiled. It should be noted that if MCLO is present then n has no effect.

{res} - is a list of resource names, with valid DCSL separators (see Part 2, Section 1.4) between them.

The resources must have been defined in the model structure definition section. DCSL allows the user to specify up to 71 resources in the above list. However, a new entry in the resource file is created for the storage of statistics on each sharer block's usage of a resource that is being shared. Thus, if sharing of resources occurs within a model then the number of resources specified in the list must accordingly be less than 71. Care must also be taken when calculating the number of resources to list if any pooled resources are named, since DCSL treats each pool member as a separate resource in the statistics file.

See Part 4 for examples.

3.2 The TASK statement

The TASK statement allows the user to specify pairs of blocks each of which defines the beginning and end of a procedural string for which duration statistics are required to be kept. The procedural strings named must not contain any branches except where these are purely alternate paths contained within the limits of the named string.

The run-by-run task duration statistics provided are of a similar nature to those provided for resource utilization and, among other things, include the task name, the commencement time, the ending time, a label which indicates the termination state (completed, aborted or not completed), and the number of completions for each task. Also, for example, see figure 4.6.

The Monte Carlo statistics for task duration statistics are also similar to those provided for resource utilization. Again, see figure 4.6.

The form of the statement is as follows:

TASK (MCLO,){bn₁,bn₂} ,

where MCLO - is optional and has the same operation as explained in Section 3.1. It must be noted that if MCLO is present in this statement it causes Monte Carlo statistics output only for both the RUSE and QUE statements as well.

{bn₁,bn₂} - is a list of block name pairs where each name has been defined in a model structure statement. The first

block in each pair may either be an EXTERNAL block or a PROCESS block. The second block in each pair must be a PROCESS block. Valid DCSL separators (see Part 2, Section 1.4) must be used between the list elements.

DCSL allows the user to specify up to 71 pairs of block names in this statement.

See Part 4 for examples.

3.3 The QUE statement

The QUE statement allows the user to specify a list of PROCESS blocks for which queue statistics are required.

The run-by-run queue statistics provided comprise the block (queue) name, the time-averaged mean queue length, and the maximum queue length (see figure 4.3).

The Monte Carlo queue statistics provided consist of the block (queue) name, the mean per run of the mean queue length, and the mean per run of the maximum queue length.

In the next mod level, the queue statistics output will be considerably improved and will provide, in particular, graphs of queue length histories.

The statement has the following form

QUE (MCLO,){pbn} ,

where MCLO - is optional and has the same operation as specified in Section 3.1 under the RUSE statement.

{pbn} - is a list of PROCESS block names that have been defined in model structure statements. Valid DCSL separators (see Part 2, Section 1.4) must be used between the list elements.

DCSL allows the user to specify up to 16 PROCESS block names in the QUE statement.

See Part 4 for examples.

3.4 The DEBUG statement

The DEBUG statement enables the user to specify three types of output which present details of various aspects of the execution of a DCSL job. The statement has the following form:

DEBUG (POST,)(CALR,)(ALOC,{pbn}) ,

where POST - causes DCSL to print the step-by-step posting of orders between all blocks in the model,

CALR - causes DCSL to print the contents of the block schedule calendar in a step-by-step fashion as orders are posted to and executed by all blocks in the model,

and ALOC,{pbn} - causes DCSL to print details of the resource allocation procedure for the PROCESS blocks listed after the ALOC command. There is no limit on the number of blocks that may be listed. Valid DCSL separators (see Part 2, Section 1.4) must be used between the list elements.

There is no prescribed order in which the parameters permitted in this statement must be coded.

4. DECK STRUCTURE

Figure 3.3 illustrates the detailed structure of the DCSL job deck from the DSYS to the ENDJ statements. This deck immediately follows the computer system's job control statements (see figures 3.1 and 3.2 for examples). Exactly the same DCSL job structure would be used if the job were to be loaded at a terminal.

Each type of statement has been described as to syntax and usage either in Part 2 or in the earlier sections of Part 3.

It should be noted that the Model Structure Definition region, is marked by a vertical bar in the figure. This denotes the fact that the order of the statements in this region is optional. Similarly the region following ENDM down to but not including RUN may be stacked with any internal order.

Note also that if any continuous sub-models were present each would follow a CSYS card and each would be terminated by another END card. These continuous 'modules' would be inserted between END and ENDM in figure 3.3. However, the use of continuous sub-models is not recommended with DCSL Mod Level 1.

Figure 3.3 also attempts to explain diagrammatically that the deck may be comprised of sequences of like runs with a given model and also of like sets of runs each set with a different model. Each section enclosed in large braces may recur as often as necessary, each one different in content but of the same format. Thus one job may contain any number of experiments with each of any number of models. However, each model structure must be included in full since, following the STOP card all previous user code is ignored and lost.

Within the data section for any one run, if a parameter is defined more than once, the last definition will be the one used.

Comment cards and blank cards may be inserted anywhere, as desired by the user, but the comment cards must contain an asterisk in column No.1.

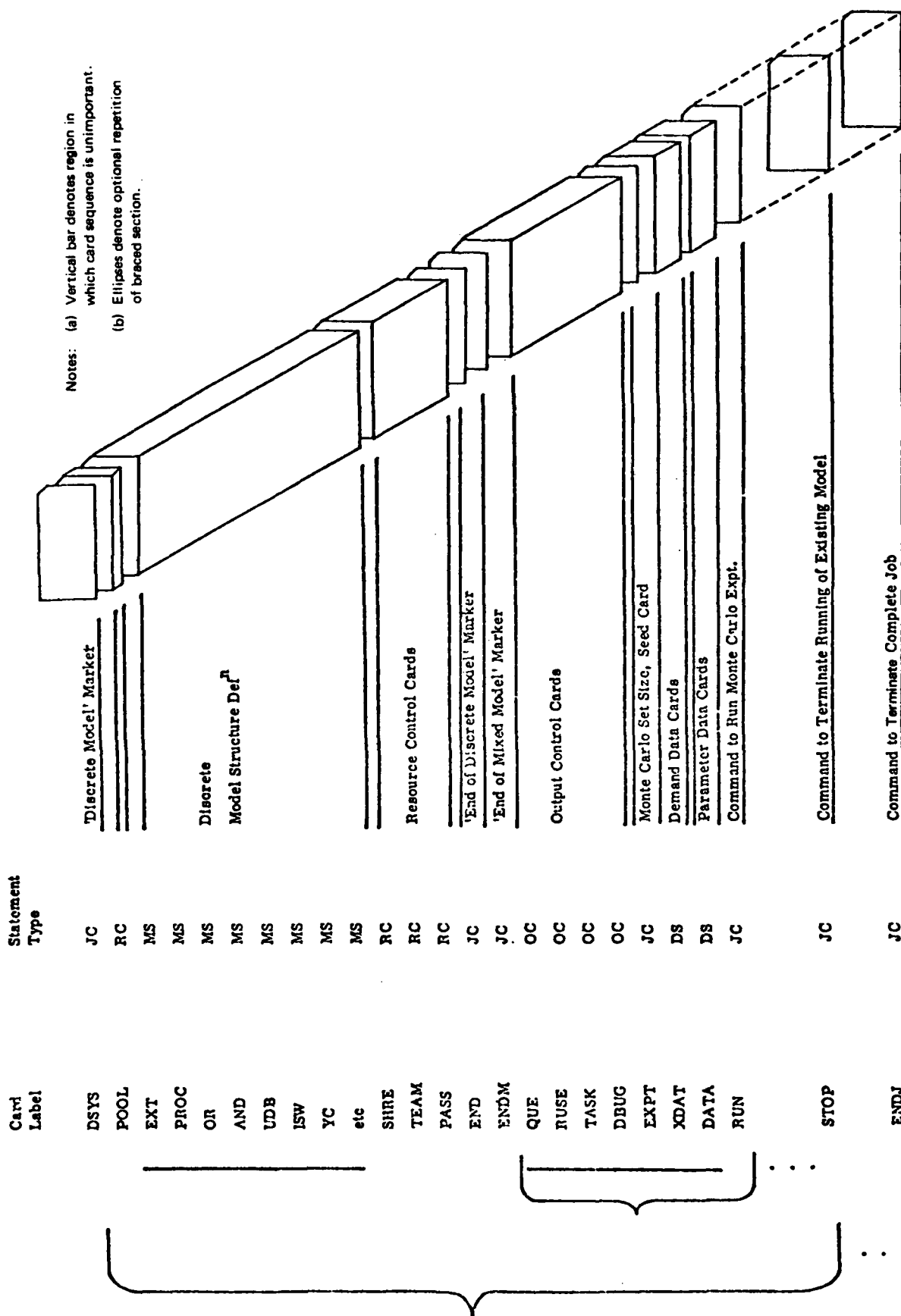


Figure 3.3. DCSL job deck structure schematic

PART 4 - SAMPLE PROBLEMS

TABLE OF CONTENTS

	Page No.
1. INTRODUCTION	1
2. SAMPLE PROBLEM NO.1 - A SIMPLE POISSON QUEUE GENERATOR	1 - 2
3. SAMPLE PROBLEM NO. 2 - A MODEL OF THE ACTIVITIES IN AN ENGINEERING CONSULTANCY	2
4. SAMPLE PROBLEM NO.3 - A MODEL OF A MOTOR VEHICLE PRODUCTION PLANT	2 - 3

LIST OF FIGURES

4.1 Poisson queue generator - Sample Problem No.1	4
4.2 Job deck for Sample Problem No.1	4
4.3 Job output for Sample Problem No.1	5
4.4 Model of activities in an engineering consultancy - Sample Problem No.2	6
4.5 Job deck for Sample Problem No.2	6
4.6 Job output for Sample Problem No.2	7 - 11
4.7 Graphical model diagram of motor vehicle production plant - Sample Problem No.3	12
4.8a Sample Problem No. 3 - Experiment a - DCSL model and data	13 - 14
4.9a Job output for Sample Problem No. 3 - Experiment a	15
4.8b Sample Problem No. 3 - Experiment b - DCSL data	16
4.9b Job output for Sample Problem No.3 - Experiment b	17
4.10 Graphical comparison of average queue lengths	18

1. INTRODUCTION

This Part of the DCSL User Guide presents three sample problems by way of illustration of the usage of both the Graphical Modelling Language (GML) technique and its sequel in the use of the DCSL simulation language. For each problem, the GML graphical model diagram is presented and this is followed by the DCSL coding. Finally, for each model, the simulation program output listings of various results of interest to the reader are also presented. The text explains the usage and meaning of all these exhibits.

It should be noted that the problems have been chosen not so much for their literal realism but as vehicles for illustration of the usage of the various features made available in DCSL. They also serve to illustrate the proper graphical syntax for the GML method.

The problems are not unrealistic however and they also show the general approach to the solution of procedural problems involving complex resource allocation.

It should be understood when studying these examples that publication of large GML diagrams and extensive output listings is difficult and so the sizes of these samples have deliberately been kept within reasonable limits.

2. SAMPLE PROBLEM NO.1 - A SIMPLE POISSON QUEUE GENERATOR

The GML block diagram of this simple model is shown in figure 4.1.

It consists of an elementary EXT block which supplies one order unconditionally, at time zero, to a COPY block (called B1). This instantly generates 1000 copies of the order and transmits them to block B2, a PROC block which is used to create a stream of issued orders separated by a small delay (1/1000 hour).

A take-off point sends a copy of this stream of orders and routes it to an RD block (B3). The original output stream from B2 passing to an OSW block (B4).

The RD makes its random decision by selecting at random from a rectangularly distributed population, a random number between 0.0 and 1.0. If this is less than or equal to 0.05 the RD output becomes OFF; otherwise it becomes ON, regardless of what state it had previously.

The OUTPUT SWITCH (B4) is controlled by the RD block so that when the latter is ON/OFF, the switch is likewise ON/OFF.

When an order passes to the OSW, the copy of it also arrives at the RD which is stacked earlier than the OSW in the user's structure code deck as shown in figure 4.2. Thus, the switch will be in the correct state by the time the order reaches it. If the state is ON, the order is passed to a disposal point and ignored. If it is OFF, the order passes to the input queue of the PROC block (B5) which represents a bank counter where a teller is serving arriving customers.

The input queue for B5 is supplied with Poisson distributed arrivals by virtue of the model logic. However, the actual queueing times will not be in accordance with the Poisson distribution, because orders (customers) are removed from the queue (served) from time to time as B5 carries out its service process.

Customers leaving the counter are represented by the PROC block output orders which in this model are sent to a disposal point and ignored.

Note that the resource TLLR, representing the teller who is required to

serve each customer, is called for at minimum priority, i.e. 1, in this model.

The structure and data code for this model, together with the output and run control data are shown in figure 4.2. Note the direct correspondence between each structure statement and its corresponding GML block.

When this simple model is run, the output listing as called for in figure 4.2 is as shown in figure 4.3.

Note the statistics given on the B5 input queue and on the use of TLLR.

3. SAMPLE PROBLEM NO. 2 - A MODEL OF THE ACTIVITIES IN AN ENGINEERING CONSULTANCY

This is the type of simple model that might be put together very quickly and run at a terminal to assess resource utilization options in a planned work programme.

The example is not taken literally from real life because of the need to abbreviate for publication purposes, but in this form it serves to illustrate the use of a number of GML/DCSL features:

- (1) the attachment of varieties of resources to PROCESS blocks;
- (2) the different graphic symbols associated with various resource functions, e.g. multiple balloons for a pooled resource, a square balloon for a sharable resource, etc. (see Part 2, Section 2.1);
- (3) passing of a unary resource, a shared team, and a pooled resource;
- (4) a team with one pooled resource candidate;
- (5) the use of the TASK statement to provide output statistics; and
- (6) the use of the EXPT control statement to provide two runs and associated Monte Carlo results (see Part 3, Section 2.5).

Figure 4.4 shows the GML diagram, figure 4.5 shows the DCSL job deck including model structure, data and job and output control statements. Figure 4.6 in five sheets displays the resource and task utilization data and statistics for the two runs requested, followed by the Monte Carlo output for the whole experiment.

Note that since only a simple form of the EXT statement has been used, rather than a form involving a finite demand lifetime and no EXT resource requirement, there can be no demand death or demand postponement in this type of experiment, as shown in the results.

Examples of complex EXT and XDAT structure code are given in Part 2, Section 2.2 - see example (e) for instance.

4. SAMPLE PROBLEM NO.3 - A MODEL OF A MOTOR VEHICLE PRODUCTION PLANT

This sample problem is again a simplified version of the type of model used in actual plant performance analysis, but is of a more complex structure than the previous two and gives a better idea of the type of model more commonly to

be encountered in practice.

A wider variety of block types is exercised here, including the AND, OR, Y-COUNTER, OSW and RD blocks.

The GML graphics conventions have all been encountered in preceding examples - for example, note the use of square balloons for shared usage of resources to distinguish this from exclusive use.

Note also that for greater legibility the multiple balloon symbol used in problem No.2, for a pooled resource has been contracted to a simple balloon in this diagram, although strictly speaking the multiple balloon should be used for each pooled resource.

The model is illustrated in figure 4.7 and is largely self-explanatory, which is one of the motivations for the development of the GML methodology. For convenience, the resource name codes are defined on the figure.

This model embodies multiple parallel and interacting procedural strings, together with resource competition interactions. That is to say the progress of the work along one string may be delayed while awaiting either the notification of completion of a stage process on another string, or the release of a resource from such a process.

Figure 4.8a (2 sheets) shows the DCSL job deck for the first of two uses of the model. Note the one-to-one mapping of the diagram onto the model structure code. This feature minimizes the possibilities for error in the setting up of the simulation experiment. Note also the use of the MCLO option in the RUSE statement to limit resource usage, task and queue outputs to the Monte Carlo form only, i.e. removing the normally printed run-by-run outputs.

Figure 4.9a (1 sheet) gives the resulting run ending times and the Monte Carlo experiment output statistics for resource usage and queue statistics.

The data and results of the second experiment using this model are presented in figures 4.8b (1 sheet) and 4.9b (1 sheet) which are similar in format with those above.

The difference between experiment a and experiment b is simply that certain model parameters have been reduced in b to simulate the use of about four times as many workers (or worker groups) deployed on PROCESSES B3 (making seat covers), B27 (making seat frames), B28 (fabricating body shells) and B38 (making engine parts). This was done because of the build-up of bottlenecks at the inputs to these processing stages in experiment a, which was reducing the productivity of the overall plant system.

The beneficial effects of the changes in resource availability can be seen by comparing figures 4.9a and 4.9b, where the respective mean queue lengths have been substantially reduced and the resource utilization figures improved accordingly. The queue length reductions are shown graphically in figure 4.10.

Note that maximum queue lengths remain unaltered, because in this simple model, an order for 50 units is placed at one time at the beginning of the simulation so that there are always 49 queue members initially at certain blocks.

It is clear that considerably more complex systems and situations can readily be analysed with little added effort by the analyst.

Space does not permit a full illustration of the usage of all possible language facilities.

Note: If $R > 0.05$ the switch, B4,
will be set ON
If $R \leq 0.05$ the switch
will be set OFF

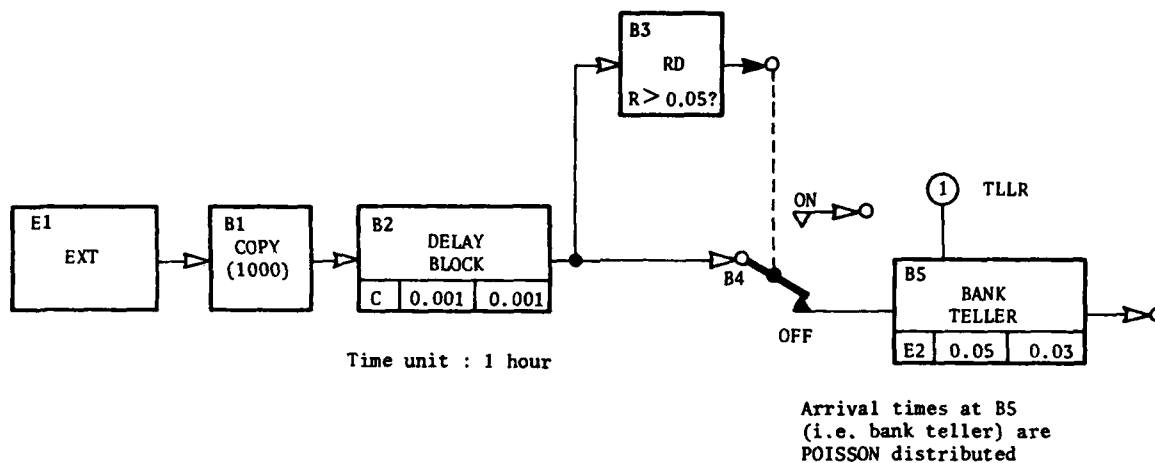


Figure 4.1. Poisson queue generator - Sample Problem No. 1

```

DSYS
*      POISSON QUEUE GENERATION
EXT    E1
COPY   B1/1000,F1
PROC   B2/B1,C,.001,.001
RD      B3/B2,P1
OSW     B4/B3,B2
*      BANK TELLER
PROC   B5/B4,E2,0.05,0.03,TLLR,1
END
ENDM
XDAT   E1,0.0
DATA   P1=0.05
RUSE    TLLR
QUF     B5
RUN
STOP
ENDJ

```

Figure 4.2. Job deck for Sample Problem No.1

*** RUN ENDED AT TIME = 2.562817E+00

TABLE 1 RESOURCE STATISTICS FOR THIS RUN

RESOURCE NAME	MEMBER NO.	NO. OF TIMES USED	TOTAL TIME IN USE	MEAN TIME IN USE	S.D. OF TIME IN USE
TLLR	1	53	2.557817E+00	4.826069E-02	1.140934E-02

TABLE 2 RESOURCE UTILIZATION DATA FOR THIS RUN

RESOURCE NAME	MEMBER NO.	USER BLOCK	START TIME	FINISH TIME	PASS OR SHARE	PASS TIME
TLLR	1	B5	4.999995E-03	4.606755E-02	-	
TLLR	1	B5	4.606755E-02	8.027059E-02	-	
TLLR	1	B5	8.027059E-02	1.488081E-01	-	
TLLR	1	B5	1.488081E-01	1.852842E-01	-	
TLLR	1	B5	1.852842E-01	2.473364E-01	-	
TLLR	1	B5	2.473364E-01	3.101877E-01	-	
TLLR	1	B5	3.101877E-01	3.536292E-01	-	
TLLR	1	B5	3.536292E-01	4.080555E-01	-	
TLLR	1	B5	4.080555E-01	4.435149E-01	-	
TLLR	1	B5	4.435149E-01	4.783296E-01	-	
TLLR	1	B5	4.783296E-01	5.426795E-01	-	
TLLR	1	B5	5.426795E-01	5.896603E-01	-	
TLLR	1	B5	5.896603E-01	6.283102E-01	-	
TLLR	1	B5	6.283102E-01	6.768958E-01	-	
TLLR	1	B5	6.768958E-01	7.137612E-01	-	
TLLR	1	B5	7.137612E-01	7.595682E-01	-	
TLLR	1	B5	7.595682E-01	8.187025E-01	-	
TLLR	1	B5	8.187025E-01	8.500519E-01	-	
TLLR	1	B5	8.500519E-01	9.182788E-01	-	
TLLR	1	B5	9.182788E-01	9.634528E-01	-	
TLLR	1	B5	9.634528E-01	1.020240E+00	-	
TLLR	1	B5	1.020240E+00	1.054048E+00	-	
TLLR	1	B5	1.054048E+00	1.100209E+00	-	
TLLR	1	B5	1.100209E+00	1.133966E+00	-	
TLLR	1	B5	1.133966E+00	1.166483E+00	-	
TLLR	1	B5	1.166483E+00	1.201179E+00	-	
TLLR	1	B5	1.201179E+00	1.252927E+00	-	
TLLR	1	B5	1.252927E+00	1.295788E+00	-	
TLLR	1	B5	1.295788E+00	1.353077E+00	-	
TLLR	1	B5	1.353077E+00	1.414927E+00	-	
TLLR	1	B5	1.414927E+00	1.466627E+00	-	
TLLR	1	B5	1.466627E+00	1.498857E+00	-	
TLLR	1	B5	1.498857E+00	1.539315E+00	-	
TLLR	1	B5	1.539315E+00	1.597717E+00	-	
TLLR	1	B5	1.597717E+00	1.640935E+00	-	
TLLR	1	B5	1.640935E+00	1.679049E+00	-	
TLLR	1	B5	1.679049E+00	1.714035E+00	-	
TLLR	1	B5	1.714035E+00	1.768420E+00	-	
TLLR	1	B5	1.768420E+00	1.832437E+00	-	
TLLR	1	B5	1.832437E+00	1.883038E+00	-	
TLLR	1	B5	1.883038E+00	1.921692E+00	-	
TLLR	1	B5	1.921692E+00	1.970072E+00	-	
TLLR	1	B5	1.970072E+00	2.048293E+00	-	
TLLR	1	B5	2.048293E+00	2.099163E+00	-	
TLLR	1	B5	2.099163E+00	2.142197E+00	-	
TLLR	1	B5	2.142197E+00	2.194633E+00	-	
TLLR	1	B5	2.194633E+00	2.250549E+00	-	
TLLR	1	B5	2.250549E+00	2.301210E+00	-	
TLLR	1	B5	2.301210E+00	2.350865E+00	-	
TLLR	1	B5	2.350865E+00	2.402905E+00	-	
TLLR	1	B5	2.402905E+00	2.472003E+00	-	
TLLR	1	B5	2.472003E+00	2.513989E+00	-	
TLLR	1	B5	2.513989E+00	2.562817E+00	-	

IF RESOURCE IS PASSED WHILE IT IS BEING SHARED ,IT IS DESIGNATED AS PASSED

QUEUE STATISTICS FOR THIS RUN

BLOCKNAME	MEAN QUEUE LENGTH	MAX QUEUE LENGTH
B5	1.629993E+01	33
STOP		
ENDJ		

Figure 4.3. Job output for Sample Problem No.1

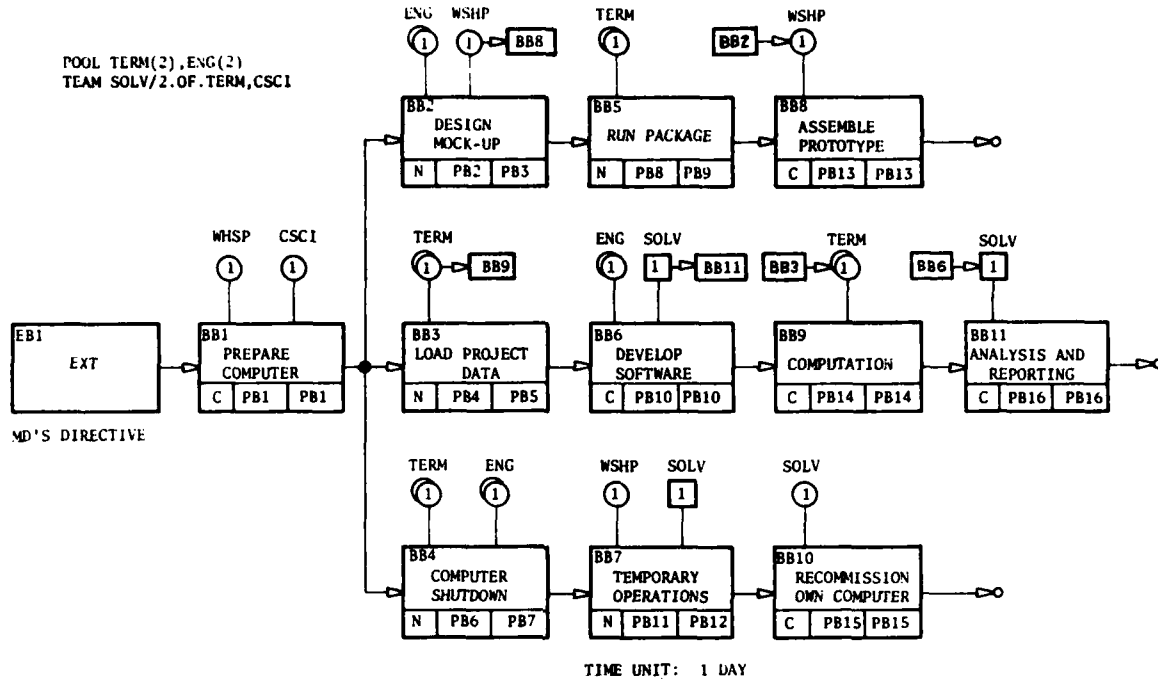


Figure 4.4. Model of activities in an engineering consultancy -
Sample Problem No. 2

```

DSYS
CONSULTING FIRM
POOL TERM(2),ENG(2)
EXT EB1
PROC BB1/EB1,C,PB1,PB1,WSHP,1,CSCI,1
PROC BB2/BB1,N,PB2,PB3,WSHP,1,ENG,1
PROC BB5/BB2,N,PB8,PB9,TERM,1
PROC BB8/BB5,C,PB13,PB13,WSHP,1
PKOC BB3/BB1,N,PB4,PB5,TERM,1
PROC BB6/BB3,C,PB10,PB10,ENG,1,SOLV,1
PROC BB9/BB6,C,PB14,PB14,TERM,1
PROC BB11/BB9,C,PB16,PB16,SOLV,1
PROC BB4/BB1,N,PB6,PB7,TERM,1,ENG,1
PROC BB7/BB4,N,PB11,PB12,WSHP,1,SOLV,1
PROC BB10/BB7,C,PB15,PB15,SOLV,1
SHRE SOLV,BB6,BB7,BB11
PASS WSHP,BB2,BB8,TERM,BB3,BB9,SOLV,BB6,BB11
TEAM SOLV/2.OF.TERM,CSCI
END
ENDM
XDAT EB1,0.0
DATA PB1=1.0,PB2=3.0,PB3=1.0,PB4=6.0,PB5=2.0,PB6=4.0
DATA PB7=1.0,PB8=6.0,PB9=2.0,PB10=2.0,PB11=3.0,PB12=1.0
DATA PB13=2.0,PB14=2.0,PB15=2.0,PB16=5.0
RUSE TERM,WSHP,ENG,SOLV,CSCI
TASK EB1,BB1,BB2
EXPT NRUN=2
RUN
STOP
ENDJ

```

Figure 4.5. Job deck for Sample Problem No.2

PASS BY SHARING PROC - RECEIVER CONVERTED TO SHARER

*** RUN ENDED AT TIME = 2.068907E+01

TABLE 1 RESOURCE STATISTICS FOR THIS RUN

RESOURCE NAME	MEMBER NO.	NO. OF TIMES USED	TOTAL TIME IN USE	MEAN TIME IN USE	S.D. OF TIME IN USE
TERM	1	2	1.468908E+01	7.344538E+00	7.558318E+00
TERM	2	4	1.768907E+01	4.422268E+00	3.685363E+00
WSHP	1	3	1.477030E+01	4.923431E+00	5.100476E+00
ENG	1	2	3.288404E+00	1.644202E+00	5.031742E-01
ENG	2	1	4.011715E+00	4.011715E+00	-
SOLV	1	3	1.100000E+01	3.666665E+00	4.725814E+00
CSCI	1	4	1.200000E+01	2.999999E+00	4.082479E+00

Figure 4.6. Job output for Sample Problem No.2

TABLE 2 RESOURCE UTILIZATION DATA FOR THIS RUN

RESOURCE NAME	MEMBER NO.	USER BLOCK	START TIME	FINISH TIME	PASS OR SHARE	PASS TIME
WSHP	1	BB1	0.0	1.000000E+00	-	
CSCI	1	RA1	0.0	1.000000E+00	-	
WSHP	1	BR2	1.000000E+00	2.288404E+00	-	
ENG	1	RR2	1.000000E+00	2.288404E+00	-	
TERM	2	RB4	1.000000E+00	5.011715E+00	-	
ENG	2	BR4	1.000000E+00	5.011715E+00	-	
TERM	1	BB3	1.000000E+00	8.673336E+00	-	
TERM	2	BB5	5.011715E+00	9.689075E+00	-	
ENG	1	BR6	9.689075E+00	1.168908E+01	-	
SOLV	1	BB6	9.689075E+00	1.168908E+01	-	
TERM	2	BB6	9.689075E+00	1.168908E+01	-	
CSCI	1	BB6	9.689075E+00	1.168908E+01	-	
WSHP	1	BB8	9.689075E+00	1.168908E+01	PASS	2.288404E+00
TERM	1	BB9	1.168908E+01	1.368908E+01	PASS	8.673336E+00
WSHP	1	BB7	1.168908E+01	1.477030E+01	-	
SOLV	1	BB7	1.168908E+01	1.477030E+01	SHRE	
TERM	2	BB7	1.168908E+01	1.477030E+01	SHRE	
CSCI	1	BB7	1.168908E+01	1.477030E+01	SHRE	
SOLV	1	BB11	1.368908E+01	1.868907E+01	PASS	1.168908E+01
TERM	2	BB11	1.368908E+01	1.868907E+01	PASS	1.168908E+01
CSCI	1	BB11	1.368908E+01	1.868907E+01	PASS	1.168908E+01
SOLV	1	BB10	1.868907E+01	2.068907E+01	-	
TERM	1	BB10	1.868907E+01	2.068907E+01	-	
CSCI	1	BB10	1.868907E+01	2.068907E+01	-	

(IF RESOURCE IS PASSED WHILE IT IS BEING SHARED, IT IS DESIGNATED AS PASSED)

Figure 4.6(Contd.).

TABLE 3 TASK UTILIZATION FOR THIS RUN

TASK NAME	TIME COMMENCED	TIME ENDED	DURATION TIME	TERMINATION STATE
EB1 /BB1	0.0	1.000000E+00	1.000000E+00	COMPLETED
BB1 /BB2	1.000000E+00	2.288404E+00	1.288404E+00	COMPLETED

TABLE 4 TASK STATS FOR THIS RUN

TASK NAME	NUMBER COMPLETED	NUMBER NOT COMPLETED	TOTAL TIME COMPLETIONS	AVERAGE TIME COMPLETIONS	S.D. TIME COMPLETIONS
EB1 /BB1	1	0	1.000000E+00	1.000000E+00	-
BB1 /BB2	1	0	1.288404E+00	1.288404E+00	-

PASS BY SHARING PROC - RECEIVER CONVERTED TO SHARER

*** RUN ENDED AT TIME = 2.227950E+01

Figure 4.6(Contd.).

TABLE 1 RESOURCE STATISTICS FOR THIS RUN

RESOURCE NAME	MEMBER NO.	NO. OF TIMES USED	TOTAL TIME IN USE	MEAN TIME IN USE	S.D. OF TIME IN USE
TERM	1	2	1.627950E+01	8.139748E+00	8.682921E+00
TERM	2	4	1.927950E+01	4.819874E+00	3.911621E+00
WSHP	1	3	1.611459E+01	5.371531E+00	6.052430E+00
ENG	1	2	4.175111E+00	2.087555E+00	1.238388E-01
ENG	2	1	3.559706E+00	3.559706E+00	-
SOLV	1	3	1.100000E+01	3.666665E+00	4.725814E+00
CSCI	1	4	1.200000E+01	2.999999E+00	4.082479E+00

TABLE 2 RESOURCE UTILIZATION DATA FOR THIS RUN

RESOURCE NAME	MEMBER NO.	USER BLOCK	START TIME	FINISH TIME	PASS OR SHARE	PASS TIME
WSHP	1	BB1	0.0	1.000000E+00	-	-
CSCI	1	BB1	0.0	1.000000E+00	-	-
WSHP	1	BB2	1.000000E+00	3.175111E+00	-	-
ENG	1	BB2	1.000000E+00	3.175111E+00	-	-
TERM	2	BB4	1.000000E+00	4.559706E+00	-	-
ENG	2	BB4	1.000000E+00	4.559706E+00	-	-
TERM	1	BB3	1.000000E+00	9.448679E+00	-	-
TERM	2	BB5	4.559706E+00	1.127950E+01	-	-
ENG	1	BB6	1.127950E+01	1.327950E+01	-	-
SOLV	1	BB6	1.127950E+01	1.327950E+01	-	-
TERM	2	BB6	1.127950E+01	1.327950E+01	-	-
CSCI	1	BB6	1.127950E+01	1.327950E+01	-	-
WSHP	1	BB8	1.127950E+01	1.327950E+01	-	-
TERM	1	BB9	1.327950E+01	1.527950E+01	-	-
WSHP	1	BB7	1.327950E+01	1.611459E+01	-	-
SOLV	1	BB7	1.327950E+01	1.611459E+01	-	-
TERM	2	BB7	1.327950E+01	1.611459E+01	-	-
CSCI	1	BB7	1.327950E+01	1.611459E+01	-	-
SOLV	1	BB11	1.527950E+01	2.027950E+01	-	-
TERM	2	BB11	1.527950E+01	2.027950E+01	-	-
CSCI	1	BB11	1.527950E+01	2.027950E+01	-	-
SOLV	1	BB10	2.027950E+01	2.227950E+01	-	-
TERM	1	BB10	2.027950E+01	2.227950E+01	-	-
CSCI	1	BB10	2.027950E+01	2.227950E+01	-	-
					PASS	3.175111E+00
					PASS	9.448679E+00
					SHRF	-
					SHRF	-
					SHRF	-
					PASS	1.327950E+01
					PASS	1.327950E+01
					PASS	1.327950E+01

IF RESOURCE IS PASSED WHILE IT IS BEING SHARED ,IT IS DESIGNATED AS PASSED

Figure 4.6(Contd.).

TABLE 3 TASK UTILIZATION FOR THIS RUN

TASK NAME	TIME COMMENCED	TIME ENDED	DURATION TIME	TERMINATION STATE
EB1 /BB1	0.0	1.000000E+00	1.000000E+00	COMPLETED
BB1 /BB2	1.000000E+00	3.175111E+00	2.175111E+00	COMPLETED

TABLE 4 TASK STATS FOR THIS RUN

TASK NAME	NUMBER COMPLETED	NUMBER NOT COMPLETED	TOTAL TIME COMPLETIONS	AVERAGE TIME COMPLETIONS	S.D. TIME COMPLETIONS
EB1 /BB1	1	0	1.000000E+00	1.000000E+00	-
BB1 /BB2	1	0	2.175111E+00	2.175111E+00	-

***** MONTE CARLO RESULTS *****

NUMBER OF RUNS	MEAN RUN TIME	STANDARD DEVIATION OF RUN TIMES
2	2.148428E+01	1.124675E+00

MONTE CARLO EXTERNAL EVENT DEATH STATISTICS

MEAN NO. OF EXTERNAL EVENT DEATHS PER RUN WAS 0.0

MONTE CARLO EXTERNAL EVENT POSTPONEMENT STATISTICS

BLOCKNAME	DATA SET	NUMBER OF DELAYS	TOTAL DELAY	MEAN DELAY PER DELAY	S.D. OF DELAY
-----------	----------	---------------------	-------------	-------------------------	---------------

MONTE CARLO RESOURCE UTILIZATION STATISTICS FOR PREVIOUS 2 RUNS

RESOURCE NAME	MEMBER NO.	MEAN OF NUMBER OF USES/RUN	MEAN OF TOTAL USE TIME/RUN	S.D. OF TOTAL USE TIME/RUN
TERM	1	2.000	1.548428E+01	1.124566E+00
TERM	2	4.000	1.848428E+01	1.124457E+00
WSHP	1	3.000	1.544244E+01	9.505601E-01
ENG	1	2.000	3.731757E+00	6.269988E-01
ENG	2	1.000	3.785710E+00	3.195971E-01
SOLV	1	3.000	1.099999E+01	1.235265E-02
CSCI	1	4.000	1.199999E+01	1.562500E-02

MONTE CARLO TASK STATISTICS FOR PREVIOUS 2 RUNS

TASK NAME	MEAN OF NO. COMPLETED	MEAN OF NO. NOT COMPLETED	MEAN OF TOTAL TIME/RUN	S.D. OF TOTAL TIME/RUN
EB1 /BB1	1.000	0.0	1.000000E+00	0.0
BB1 /BB2	1.000	0.0	1.731757E+00	6.269988E-01

STOP
ENDJ

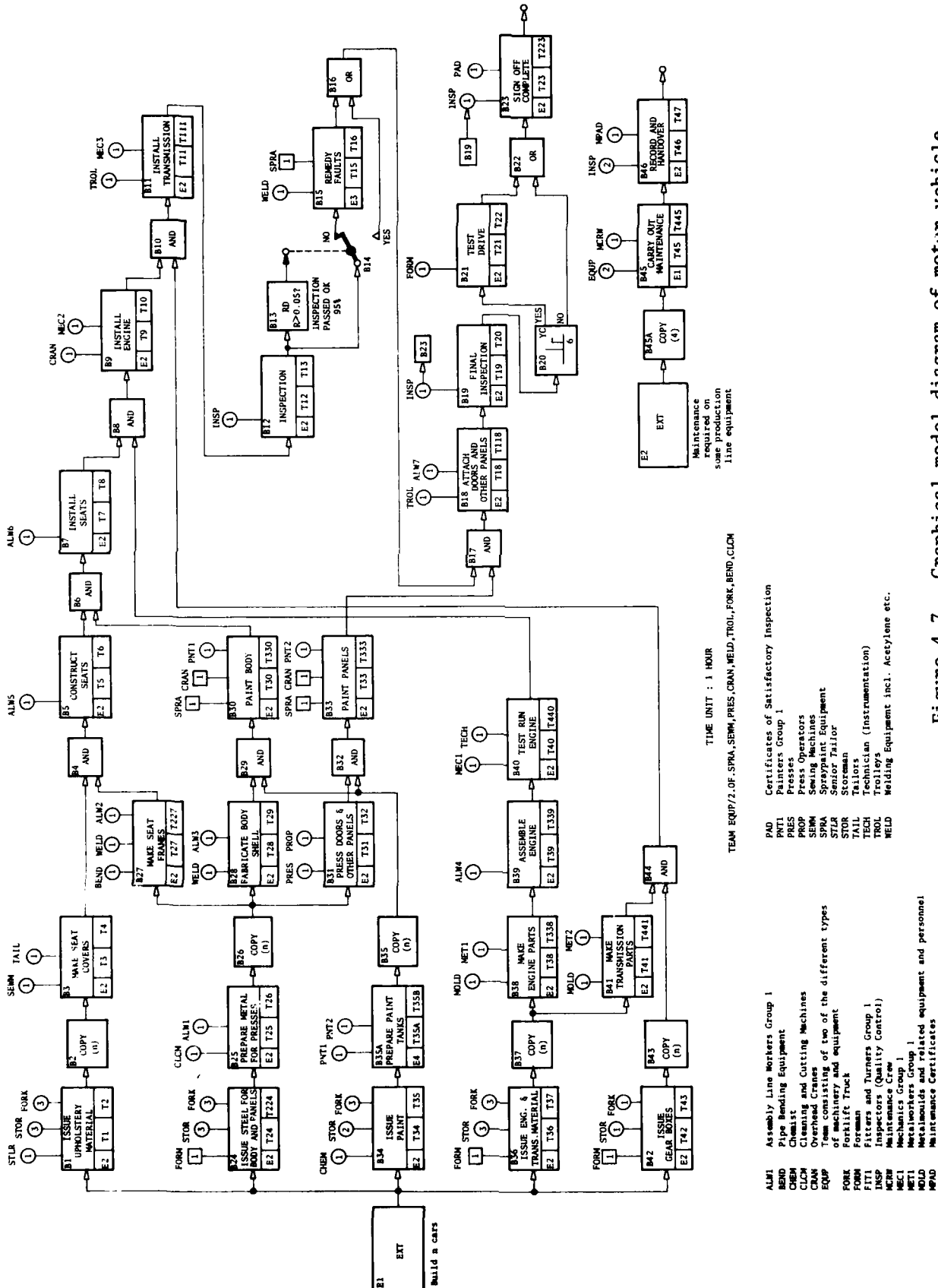


Figure 4.7. Graphical model diagram of motor vehicle production plant - Sample Problem No. 3

```
DSYS
POOL STOR(3),FORK(2),MOLD(2),CRAN(2),WELD(2),INSP(2),TROL(2)
*
* ORDER FOR BATCH OF CARS TO BE MADE IS GIVEN
EXT E1
* THE MODEL STRUCTURE FOR THE PRODUCTION LINE FOLLOWS
PROC B1/E1,E2,T1,T2,STLR,1,STOR,3,FORK,3
COPY B2/N,B1
PROC B3/B2,E2,T3,T4,SEWM,1,TAIL,1
PROC B24/E1,E2,T24,T224,FORM,1,STOR,3,FORK,3
PROC B25/B24,E2,T25,T26,CLCM,1,ALW1,1
COPY B26/N,B25
PROC B27/B26,E2,T27,T227,BEND,1,WELD,1,ALW2,1
AND B4/B3,B27
PROC B5/B4,F2,T5,T6,ALW5,1
PROC B28/B26,E2,T28,T29,WELD,1,ALW3,1
PROC B31/B26,E2,T31,T32,PRES,1,PROP,1
PROC B34/E1,E2,T34,T35,CHEM,1,STOR,2,FORK,3
PROC B35A/B34,E4,T35A,T35B,PNT1,1,PNT2,1
COPY B35/N,B35A
AND B29/B28,B35
PROC B30/B29,E2,T30,T330,SPRA,1,CRAN,1,PNT1,1
AND B6/B5,B30
PROC B7/B6,E2,T7,T8,ALW6,1
AND B32/B31,B35
PROC B33/B32,E2,T33,T333,SPRA,1,CPAN,1,PNT2,1
PROC B36/E1,E2,T36,T37,FORM,1,STOR,3,FORK,3
COPY B37/N,B36
PROC B38/B37,E2,T38,T338,MOLD,1,MET1,1
PROC B39/B38,E2,T39,T339,ALW4,1
PROC B40/B39,E2,T40,T440,MEC1,1,TECH,1
AND B8/B7,B40
PROC B9/B8,E2,T9,T10,CRAN,1,MEC2,1
PROC B41/B37,E2,T41,T441,MOLD,1,MET2,1
PROC B42/E1,E2,T42,T43,FORM,1,STOR,1,FORK,1
COPY B43/N,B42
AND B44/B41,B43
AND B10/B9,B44
PROC B11/B10,E2,T11,T111,TROL,1,MEC3,1
PROC B12/B11,E2,T12,T13,INSP,1
*
* 95 PER CENT OF CARS PASS FIRST INSPECTION
RD B13/B12,0.05
OSW B14/B13,B12
*
* 5 PER CENT OF CARS NEED FAULTS REMEDIED
PROC B15/B14,E3,T15,T16,SPRA,1,WELD,1
OR B16/B15,B14(2)
AND B17/B16,B33
PROC B18/B17,E2,T18,T118,TROL,1,ALW7,1
PROC B19/B18,E2,T19,T20,INSP,1
YC B20/B19,6
*
* TEST DRIVE EVERY 6TH CAR
PROC B21/B20(2),E2,T21,T22,FORM,1
```

Figure 4.8a. Sample Problem No.3 - Experiment a - DCSL model and data

```
OR      B22/B21,B20
*
*      SIGN OFF COMPLETED CAR
PROC    B23/B22,E2,T23,T223,INSP,1,PAD,1
*
*      GANG IS REQUIRED TO CARRY OUT ROUTINE MAINTENANCE ON EQUIPMENT
*      AT VARIOUS TIMES DURING THE PRODUCTION SEQUENCE
EXT      E2
COPY    B45A/4,E2
PROC    B45/B45A,E1,T45,T445,EQU,2,MCRW,1
PROC    B46/B45,E2,T46,T47,INSP,2,MPAD,1
*
*      RESOURCE MANAGEMENT (POOL STATEMENT IS AT FRONT OF DECK)
PASS    INSP,B19,B23
SHRE    FORM,B24,B36,B42
SHRE    SPRA,B15,B30,B33
SHRE    CRAN,B30,B33
TEAM    EQUIP/2,OF.SPRA,SEWM,PRES,CRAN,WELD,TROL,FORK,BEND,CLOM
END
ENDM
*
*      INPUT DEMAND DATA
*      PRODUCTION BEGINS AT 0730 ON DAY 1
XDAT    E1,7.5
*
*      MAINTENANCE CARRIED OUT COMMENCING AT 1600 EVERY DAY
*      WHILE PRODUCTION IS IN PROGRESS
XDAT    E2,16.0,40.0,64.0,88.0,112.0,136.0
*
*      ASSIGN VALUES TO SYMBOLIC PARAMETERS USED IN STRUCTURE CODE
DATA    T1=5.0,T2=4.0,T3=1.5,T4=1.0,T5=1.0,T6=0.5,T7=0.25,T8=0.2
DATA    T9=0.5,T10=0.3,T11=0.4,T111=0.2,T12=0.25,T13=0.15,T15=0.7
DATA    T16=0.3,T18=0.3,T118=0.2,T19=0.3,T20=0.2,T21=0.09,T22=0.05
DATA    T23=0.03,T223=0.02,T24=7.0,T224=6.0,T25=5.0,T26=2.5,T27=0.75
DATA    T227=0.6,T28=2.0,T29=1.5,T30=0.3,T330=0.2,T31=0.15,T32=0.13
DATA    T33=0.15,T333=0.11,T34=3.0,T35=2.0,T35A=5.0,T35B=2.5,T36=4.0
DATA    T37=2.5,T38=1.5,T338=1.2,T39=0.6,T339=0.5,T40=0.1,T440=0.09
DATA    T41=1.2,T441=1.0,T42=5.0,T43=3.5,T45=1.0,T445=0.8,T46=0.15
DATA    T47=0.1,N=50
RUSE    MCLO,PAD,BEND,CLOM,CRAN,FORK,PRES,SEWM,SPRA,TROL,WELD,EQU,STOR,
        PNT1,INSP,MCRW,FORM
QUE      B3,B5,B7,B9,B11,B15,B18,B27,B28,B30,B31,B33,B38,B39,B40,B41
EXPT    NRUN=5
RUN
STOP
ENDJ
```

Figure 4.8a(Contd.).

***** MONTE CARLO RESULTS *****

NUMBER OF RUNS	MEAN RUN TIME	STANDARD DEVIATION OF RUN TIMES
5	1.406568E+02	3.750000E-01

MONTE CARLO EXTERNAL EVENT DEATH STATISTICS

MEAN NO. OF EXTERNAL EVENT DEATHS PER RUN WAS 0.0

MONTE CARLO EXTERNAL EVENT POSTPONEMENT STATISTICS

BLOCKNAME	DATA SET	NUMBER OF DELAYS	TOTAL DELAY	MEAN DELAY PER DELAY	S.D. OF DELAY
-----------	----------	---------------------	-------------	-------------------------	---------------

MONTE CARLO RESOURCE UTILIZATION STATISTICS FOR PREVIOUS 5 RUNS

RESOURCE NAME	MEMBER NO.	MEAN OF NUMBER OF USES/RUN	MEAN OF TOTAL USE TIME/RUN	S.D. OF TOTAL USE TIME/RUN
PAD	1	50.000	1.477639E+00	4.705775E-02
BEND	1	50.000	3.692625E+01	7.486968E-01
CLCM	1	1.000	6.074514E+00	2.943854E+00
CRAN	1	133.400	3.729672E+01	1.259340E+00
CRAN	2	16.800	2.679637E+00	2.814195E-01
FORK	1	3.000	7.836426E+00	7.966835E-01
FORK	2	2.000	1.249622E+01	1.291688E+00
PRES	1	62.800	2.068193E+01	1.650640E+00
SEWM	1	61.400	8.603558E+01	1.543003E+00
SPRA	1	125.400	4.047856E+01	5.659616E-01
TROL	1	92.200	3.219795E+01	1.213531E+00
TROL	2	7.800	2.570827E+00	5.168701E-01
WELD	1	81.200	9.932416E+01	2.466178E+00
WELD	2	20.600	3.868959E+01	6.335354E-01
EQUP	1	24.000	2.504732E+01	3.017699E-01
STOR	1	3.000	7.836426E+00	7.966835E-01
STOR	2	2.000	1.249622E+01	1.291688E+00
STOR	3	0.0	0.0	0.0
PNT1	1	51.000	1.483505E+01	5.098517E-01
INSP	1	115.400	3.005499E+01	6.327643E-01
INSP	2	8.800	2.372091E+00	3.639527E-01
MCRW	1	24.000	2.504732E+01	3.017699E-01
FORM	1	11.000	1.321221E+01	1.344840E+00

MONTE CARLO QUEUE STATISTICS FOR PREVIOUS 5 RUNS

BLOCK NAME	MEAN PER RUN OF MEAN QUEUE LENGTH	MEAN PER RUN OF MAX QUEUE LENGTH
B3	1.286711E+01	4.900000E+01
B5	1.767206E-01	3.200000E+00
B7	7.027651E-04	1.000000E+00
B9	1.042538E-02	1.200000E+00
B11	1.992563E-03	1.000000E+00
B15	2.544429E-02	4.000000E-01
B18	1.506401E-03	1.000000E+00
B27	6.366089E+00	4.900000E+01
B28	1.719969E+01	4.900000E+01
B30	1.382424E-01	2.599999E+00
B31	1.708325E+00	4.979999E+01
B33	0.0	0.0
B38	1.303113E+01	4.900000E+01
B39	0.0	0.0
B40	0.0	0.0

STOP

ENDJ

Figure 4.9a. Job output for Sample Problem No.3 - Experiment a

```
*
* INPUT DEMAND DATA
* PRODUCTION BEGINS AT 0730 ON DAY 1
XDAT E1,7.5
*
* MAINTENANCE CARRIED OUT COMMENCING AT 1600 EVERY DAY
* WHILE PRODUCTION IS IN PROGRESS
XDAT E2,16.0,40.0,64.0,88.0,112.0,136.0
*
* ASSIGN VALUES TO SYMBOLIC PARAMETERS USED IN STRUCTURE CODE
DATA T1=5.0,T2=4.0,T3=0.4,T4=0.25,T5=1.0,T6=0.5,T7=0.25,T8=0.2
DATA T9=0.5,T10=0.3,T11=0.4,T111=0.2,T12=0.25,T13=0.15,T15=0.7
DATA T16=0.3,T18=0.3,T118=0.2,T19=0.3,T20=0.2,T21=0.09,T22=0.05
DATA T23=0.03,T223=0.02,T24=7.0,T224=6.0,T25=5.0,T26=2.5,T27=0.2
DATA T227=0.15,T28=0.5,T29=0.4,T30=0.3,T330=0.2,T31=0.15,T32=0.13
DATA T33=0.15,T333=0.11,T34=3.0,T35=2.0,T35A=5.0,T358=2.5,T36=4.0
DATA T37=2.5,T38=0.4,T338=0.3,T39=0.6,T339=0.5,T40=0.1,T440=0.09
DATA T41=1.2,T441=1.0,T42=5.0,T43=3.5,T45=1.0,T445=0.8,T46=0.15
DATA T47=0.1,N=50
RUSE MCLD,PAD,BEND,CLCM,CRAN,FORK,PRES,SEWM,SPRA,TROL,WELD,EQUP,STOR,
PNT1,INSP,MCRW,FORM
QUE B3,B5,B7,B9,B11,B15,B18,B27,B28,B30,B31,B33,B38,B39,B40,B41
EXPT NRUN=5
```

Figure 4.8b. Sample Problem No.3 - Experiment b - DCSL data

***** MONTE CARLO RESULTS *****

NUMBER OF RUNS	MEAN RUN TIME	STANDARD DEVIATION OF RUN TIMES
5	1.400276E+02	2.795085E-01

MONTE CARLO EXTERNAL EVENT DEATH STATISTICS

MEAN NO. OF EXTERNAL EVENT DEATHS PER RUN WAS 0.0

MONTE CARLO EXTERNAL EVENT POSTPONMENT STATISTICS

BLOCKNAME	DATA SET	NUMBER OF DELAYS	TOTAL DELAY	MEAN DELAY PER DELAY	S.D. OF DELAY
-----------	----------	---------------------	-------------	-------------------------	---------------

MONTE CARLO RESOURCE UTILIZATION STATISTICS FOR PREVIOUS 5 RUNS

RESOURCE NAME	MEMBER NO.	MEAN OF NUMBER OF USES/RUN	MEAN OF TOTAL USE TIME/RUN	S.D. OF TOTAL USE TIME/RUN
PAD	1	50.000	1.485629E+00	5.756337E-02
BEND	1	50.000	9.965393E+00	2.601691E-01
CLCM	1	1.000	4.644565E+00	2.150549E+00
CRAN	1	125.200	3.014665E+01	1.559372E+00
CRAN	2	24.800	1.070640E+01	6.740065E-01
FORK	1	9.000	1.279261E+01	2.114533E+00
FORK	2	2.000	1.062161E+01	1.756163E+00
PRES	1	55.400	1.281374E+01	9.677414E-01
SEWM	1	70.000	4.010719E+01	2.706329E-01
SPRA	1	125.000	4.056017E+01	1.792181E+00
TROL	1	85.000	2.888815E+01	1.663310E+00
TROL	2	15.000	5.316384E+00	1.527089E+00
WELD	1	81.000	2.584091E+01	9.435353E-01
WELD	2	21.400	1.054966E+01	5.950336E-01
EQUIP	1	24.000	2.418561E+01	8.768117E-01
STOR	1	3.000	1.279261E+01	2.114533E+00
STOR	2	2.000	1.062161E+01	1.756163E+00
STOR	3	0.0	0.0	0.0
PNT1	1	51.000	2.104517E+01	5.396283E-01
INSP	1	106.800	2.782713E+01	1.102507E+00
INSP	2	17.200	4.460751E+00	9.429064E-01
MCRW	1	24.000	2.418561E+01	8.768117E-01
FORM	1	11.000	1.411288E+01	1.679706E+00

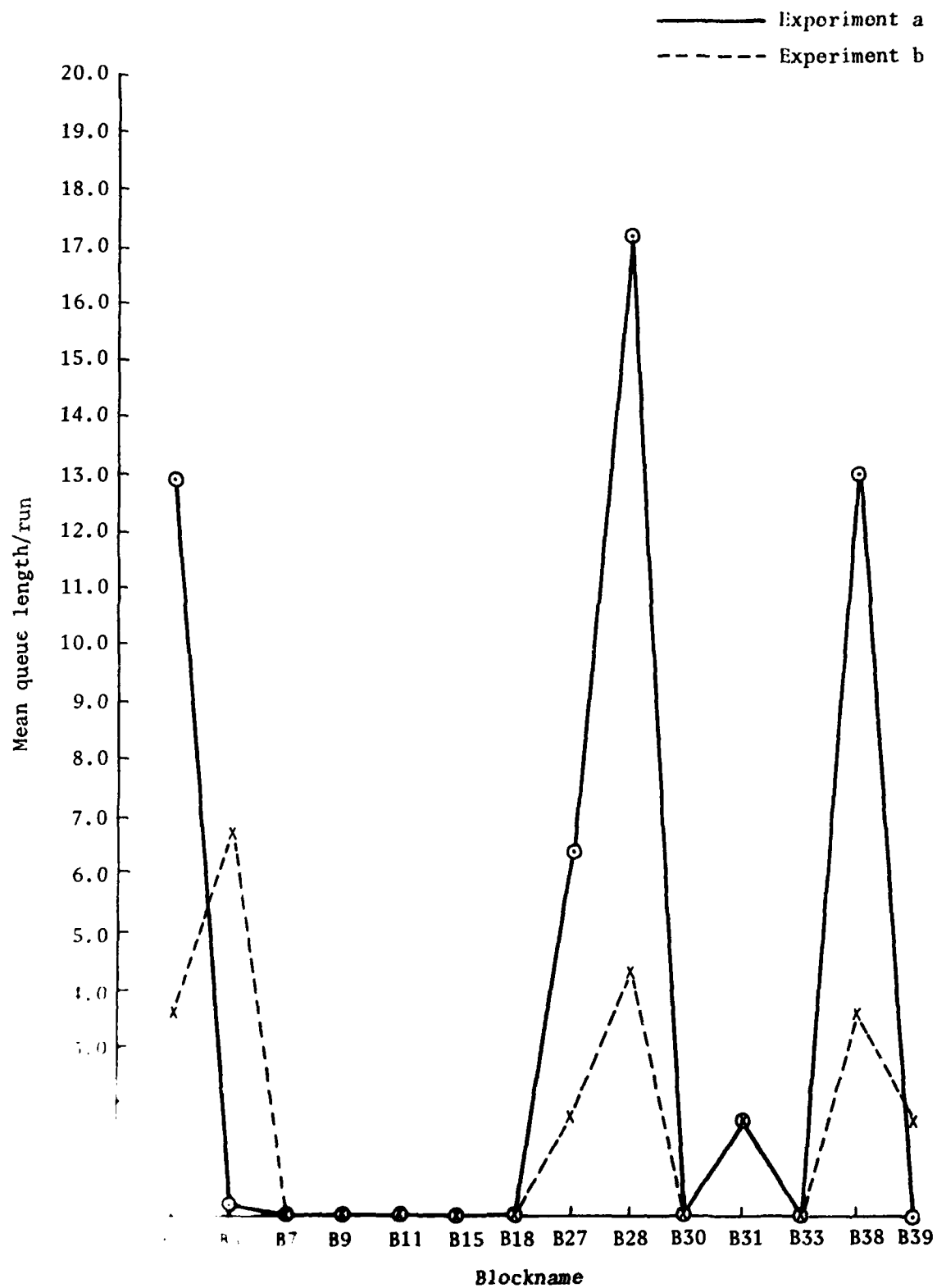
MONTE CARLO QUEUE STATISTICS FOR PREVIOUS 5 RUNS

BLOCK NAME	MEAN PER RUN OF MEAN QUEUE LENGTH	MEAN PER RUN OF MAX QUEUE LENGTH
B3	3.542984E+00	4.900000E+01
B5	6.711139E+00	3.679999E+01
B7	3.890803E-04	6.000000E-01
B9	1.320394E-02	1.799999E+00
B11	1.221505E-03	6.000000E-01
B15	3.124106E-03	4.000000E-01
B18	9.015938E-04	1.000000E+00
B27	1.750236E+00	4.900000E+01
B28	4.344430E+00	4.900000E+01
B30	2.137606E-01	7.200000E+00
B31	1.713040E+00	4.959999E+01
B33	4.342722E-01	1.100000E+01
B38	3.556667E+00	4.900000E+01
B39	1.674566E+00	1.639999E+01
B40	0.0	0.0

STOP

ENDJ

Figure 4.9b. Job output for Sample Problem No.3 - Experiment b



Graphical comparison of average queue lengths

APPENDIX I

A GRAPHICAL REPRESENTATION OF THE DCSL SIMULATION
SCHEME

It has been mentioned (see Part 2, Section 2.2) that the simulation of real problems creates the need for complex mechanisms for the introduction of demands (or requests for service) into a model.

The extent of the demand control aspects of the overall simulation scheme can be appreciated by reference to figure I.1 which is both structural and procedural in nature. Structurally it shows the inter-relationships of the generic components of the simulation scheme, while procedurally it shows the order of computation in handling each component aspect at simulation time. The order of computation proceeds from left to right across the bottom level of the figure. Notice that the EXTERNAL block provides all of the facilities of figure I.1 except for demand gating (within appropriate service units) and simulation of procedures (i.e. boxes 7, 8 and 9).

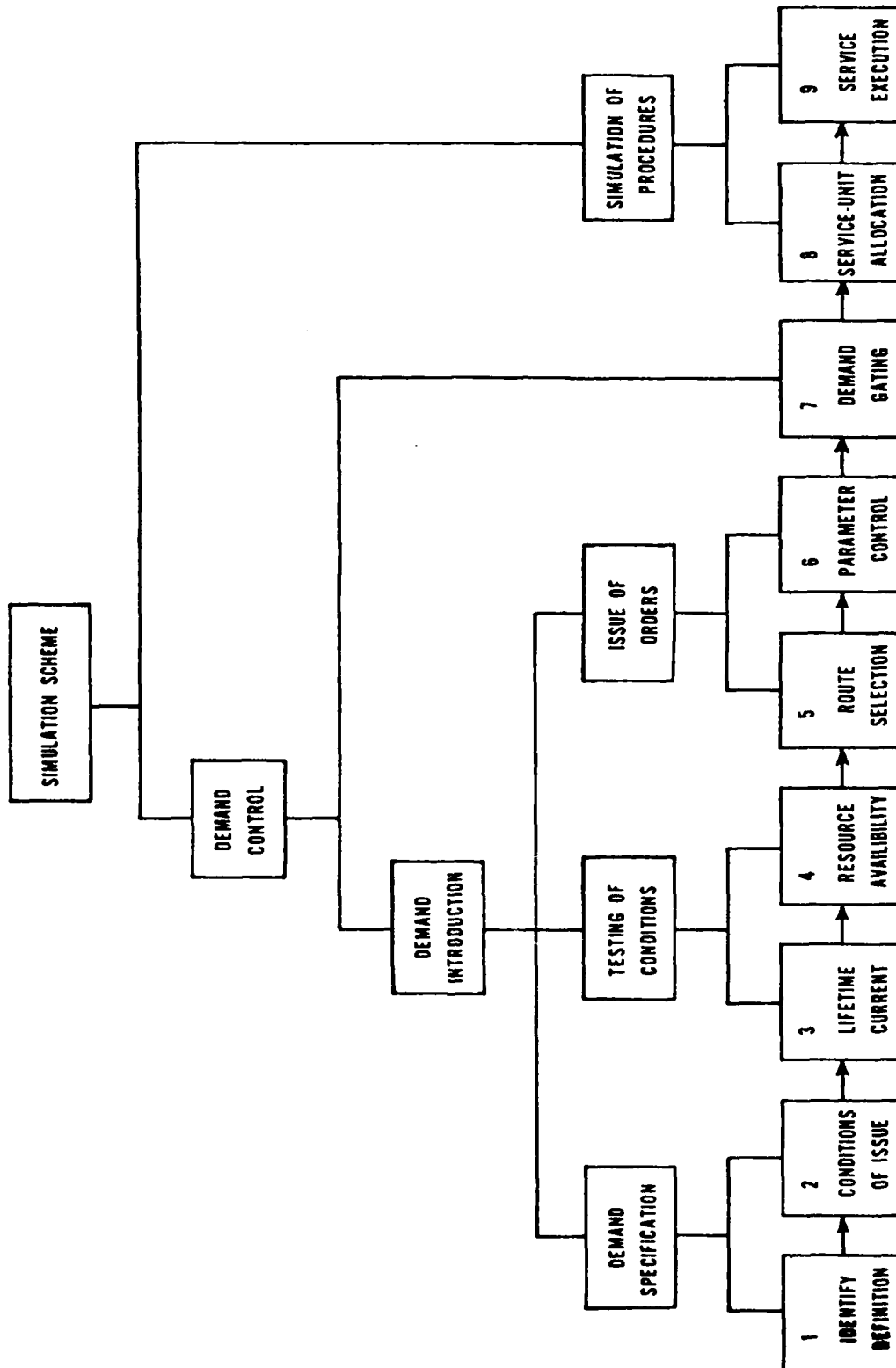


Figure I.1 Classification of the elements of the simulation scheme, showing the sequence of computation

APPENDIX II

THE USER DEFINED BLOCK (UDB) - OPERATION AND CODING
DETAILS

This appendix is intended to provide enough detailed information on the DCSL system program operation and UDB construction to enable the user to produce any UDB he requires.

II.1 DCSL system program operation

Each UDB (block name), defined by the user in the Model Structure Definition region (see Figure 3.3), is allocated its own block memory area by the DCSL system program. On each call to the UDB subroutine (see below) to service the given occurrence of the UDB, its input state variables (i.e. queue lengths and signal values) are copied from this area into an input vector IX(10), contained in the labelled COMMON block IN. Equally, upon the subsequent return from the UDB subroutine, the contents of the output vector IY(10), contained in the labelled COMMON block OUT, are copied into the block memory area for this UDB. By this means, these outputs (i.e. output orders and signals) may be transmitted downstream in the model structure.

Although IX and IY are used as temporary registers in the servicing of every UDB occurrence, the user should realize that this system operated loading and unloading of them provides him with the only safe local variable storage for use with each occurrence. That is unless the UDB subroutine is the only one of its type, or unless it is coded to recognize UDB block names and use them to branch to appropriate local memory words. The latter is tricky, requiring character handling routines and is not recommended. In general, it is best to follow the simple instructions contained in II.2 below.

II.2 Coding UDB subroutines

In this section, it is assumed that the reader is familiar with Fortran; it should be an easy matter for a user who prefers to use some other language to relate the comments made here to his preferred language.

The method of coding a UDB subroutine is described by means of an example UDB routine and a number of associated notes which highlight the important points.

Figure II.1 shows an example listing of a UDB subroutine, in this case written in Fortran IV. The associated DCSL block definition statements for this subroutine are:

UDB name1/1/4,1,0*ibn1,ibn2,ibn3,ibn4 (subroutine
SR1)

where, in particular, name1, ibn1, ibn2, ibn3, and ibn4 are the blockname of this occurrence of the UDB type 1 and the names of the four discrete input blocks, respectively; and

UDB name2/11/0,0,0*ibn1,XXX (subroutine
SR11)

here, again, name2 and ibn1 are the blockname of this occurrence of the UDB type 11 and the name of the first discrete input block, respectively. The input block XXX is a dummy input block, the use of which is fully explained in note (7) below.

A full description of the syntax for the UDB DCSL statement is given in Part 2, Section 2.13.

The following notes should provide the user with all the information he requires to create his own UDB's.

- (1) The JCL instructions at the beginning of the figure were as required for FORTRAN IV(H) compilation by an IBM 370/168.
- (2) Note that the calling program transmits three variables into subroutine UDB, namely ID, BNAME and IEND. ID is the block type identifier, BNAME is the block name of the UDB-type occurrence to be serviced and IEND is a flag which if equal to zero signals that this call was made precisely at the end of the run. This latter feature allows the user to test IEND and, if zero is found, to write an end-of-run report directly in the UDB, or in another subroutine called by UDB - in this case in SR11.
- (3) Note the use of the computed GO TO in UDB to branch to the appropriate UDB-type subroutine call. (This could of course be simply an area of code within UDB itself.)
- (4) Note that the calling program knows the name of the occurrence being serviced at 'CALL UDB time' and has just loaded the COMMON vector IX (which is shared by all UDB's) with the input data (queues and signals) which are local to the occurrence being serviced. Thus the local inputs are passed to the UDB-type routine with the values appropriate to the occurrence in question. These values are normally kept by the DCSL program in a block definition file which is part of either the discrete or continuous block table as appropriate.

```

//      EXFC      FORTXC1,FXPQPT=2,PGMLB='IPC,INCS,IRH,1,0A0'.
//      PARM,LKED='MAF,LE1,L1S1,XREF,NCAL',FXPNAME=UD9
//FURT.SYSIN DD *
      SUBROUTINE UDB (ID,BNAME,IEND)
      IF(IEND.EQ.0) GO TO 11
      GO TO (1,2,3,4,5,6,7,8,9,10,11),ID
1      CALL SR1
      RETURN
2      CALL SR2
      RETURN
3      CALL SR3
      RETURN
4      CALL SR4
      RETURN
5      CALL SR5
      RETURN
6      CALL SR6
      RETURN
7      CALL SR7
      RETURN
8      CALL SR8
      RETURN
9      CALL SR9
      RETURN
10     CALL SR10
      RETURN
11     IF(ID.NE.11) RETURN
      CALL SR11(BNAME,IEND)
      RETURN
      END

      SUBROUTINE SR1
      COMMON/IN/IX(10)
      COMMON/OUT/IY(10)
      DO 10 J=1,4
      IY(J)=0
10     CONTINUE
      IY(5)=N
      IF(IY(5).EQ.0) IY(5)=1
      IF(IX(4).EQ.0) RETURN
      N = 0
      DO 20 J=1,3
      IY(J)=IX(J)
      IF(IX(J).EQ.1) N=N+1
      IX(J) = 0
20     CONTINUE
      IX(4)=0
      M = N+1
      GO TO (30,40,60,60),M
30     WRITE (6,100)
100    FORMAT (40HNO BTY AVAILABLE, A REGT, ABORT MISSION)
      IY(4)=1
      RETURN
40     WRITE (6,200)
200    FORMAT (45HONE BTY AVAILABLE, A REGT, GO TO BTY MISSION)
60     IY(5)=N
      RETURN
      END

      SUBROUTINE SR11(BNAME,IEND)
      COMMON /IN/IX(10)
      COMMON /OUT/IY(10)
      IX(2)=IX(2)+IX(1)
      IX(1)=0
      IF(IEND.NE.0) RETURN
      IF(IX(2).EQ.0) RETURN
      WRITE(6,100) BNAME,IX(2)
100    FORMAT(' BLOCK ',A4,' COLLECTED ',I6,' ORDERS THIS RUN')
      RETURN
      END
//LKED=SYS1 MMD DD DSN=PGMLB.(&FXPNAME),DISP=OLD

```

Note: For convenience only Subroutines SR1 and SR11 are shown in this figure. In this case Subroutines SR2 to SR10 were all similar to SR1 but referred to different batteries or regiments.

Figure II.1 Listing of a UDB subroutine

This file is reserved specifically for the individual block occurrence and is identified by means of its block name.

- (5) BNAME is transmitted to UDB simply for convenience in writing the user's reports during or at the end of the run.
- (6) At the completion of the UDB service the output orders and signals are loaded into the COMMON vector IY by the user-written code. On return to the DCSL calling program, this vector is unloaded back into the space in the block definition file reserved specifically for outputs from this block occurrence and subsequently DCSL transmits these on to the distribution list of receiver blocks as necessary.
- (7) Since the same UDB user-written code is executed each time a given UDB-type is serviced, the user must exercise great care with the creation of local variables for storage of data between calls to a given occurrence. This is because an intervening call to the same UDB-type for a different occurrence will be likely to overwrite the local storage word.
If local storage words are required the safe way to obtain them is to create dummy inputs and to load the stored values into the appropriate words in IX. There are two ways to do this:
 - (a) Create a dummy EXTERNAL block with a dummy name such as XXX by simply writing an EXT XXX statement and not naming it as an input to any block other than the UDB occurrence in question (no associated XDAT statement is needed).
 - (b) If the storage word is real, nci may be artificially increased and an extra real value added to the cin list in the UDB structure statement. This value may be any real number and will initialize the storage word accordingly.
- (8) To access the value of the simulation time word during execution of a UDB the user may use a common statement reference below:

COMMON/DCSLXX/time

Here the label 'time' may be any legal real name.

I N D E X

	Part-Section
Access to DCSL	3-1
AND	1-4.3, 2-2.3
Blank lines/cards	2-1.5
Comment statements	2-1.5
Connecting lines	1-6.1
Content word separators	2-1.4
Continuation lines	2-1.3
COPY	1-4.12, 2-2.14
CSYS	3-2.3
DATA	2-5
DEBUG	3-3.4
Deck Structure	2-1.1, 3-4
Discrete 'summing junctions'	1-6.5
DSYS	3-2.1
END	3-2.2
ENDJ	3-2.8
ENDM	3-2.4
EXPT	3-2.5
EXTERNAL(EXT)	1-4.2, 2-2.2
FLIP/FLOP(FF)	1-4.6, 2-2.7
IGNORE(IGN)	1-4.9, 2-2.11
Implementation	3-1
INPUT SWITCH (SW)	1-4.7, 2-2.8
Names - block -	2-1.2
- Parameter -	2-1.2
- resource -	2-1.2
Limitations	1-9
OR	1-4.3, 2-2.3
OUTPUT SWITCH(OSW)	1-4.7, 2-2.9
PASS	2-4.4
- graphical symbols -	2-2.1
POOL	2-4.2
- graphical symbols -	2-2.1
Priority	2-4.1
PROCESS(PROC)	1-4.1, 2-2.1
QUE	3-3.3
QUEUE CLEAR(QC)	1-4.10, 2-2.12

	Part-Section
RANDOM DECISION(RD)	1-4.4,2-2.4
RUN	3-2.6
RUSE	3-3.1
Separators	2-1.4
SHARE(SHRE)	2-4.3
- graphical symbols -	2-2.1
Sorting facilities	1-7
Statement structure	2-1.2
STOP	3-2.7
'Take-off' points	1-6.2,1-3
TASK	3-3.2
TEAM	1-8,2-4.5
- graphical symbols -	2-2.1
TEST RESOURCES (TR)	1-4.8,2-2.10
USER DEFINED BLOCK(UDB)	1-4.11,2-2.13
X-COUNTER(XC)	1-4.5,2-2.5
XDAT	2-2.2
Y-COUNTER(YC)	1-4.5,2-2.6

DISTRIBUTION

Copy No.

EXTERNAL

In United Kingdom

Defence Scientific and Technical Representative, London	1 - 2
Library, Royal Aircraft Establishment	3
Royal Armament Research and Development Establishment	
Library	4
(Attention: Mr L.F. Jones, UK National Leader, TTCP WAG-6)	5
(Attention: Mr T.J. Stakemire, MA3 Branch)	6

In United States of America

Counsellor Defence Science, Washington D.C.	7
Defence Research and Development Attache, Washington D.C.	8
US Army Materiel Systems Analysis Activity, Maryland	
(Attention: Dr J. Sperazza)	9
(Attention: Mr C.T. Odom)	10
US Army Ballistic Research Laboratory	
(Attention: Mr H.L. Reed, Jr., US National Leader, TTCP WAG-6)	11 - 12
(Attention: Mr B.L. Reichard)	13
Aerospace Medical Research Laboratory (HED)	
(Attention: Dr G.P. Chubb)	14
Mitchell and Gunther Association Inc.	
(Attention: Dr E.E.L. Mitchell)	15

In Canada

Defence Research Establishment, Valcartier	16
National Defence Headquarters, Ottawa	
(Attention: Major Y. Girard, Canadian National Leader, TTCP WAG-6)	17

In Australia

Chief Defence Scientist	18
Deputy Chief Defence Scientist	19
Director, Joint Intelligence Organisation (DDSTI)	20
Controller, Projects and Analytical Studies	21
Superintendent, Analytical Studies	22
Army Scientific Adviser	23
Director of Operational Analysis - Army	
(Attention: Dr M.W. Jarvis)	24 - 25
Navy Scientific Adviser	26

	Copy No.
Air Force Scientific Adviser	27
Staff Officer (Science) HQ FF Comd	28
Staff Officer (Science) HQ 1 Division	29
Superintendent, Science and Technology Programmes	30
Defence Information Services Branch (for microfilming)	31
Defence Information Services Branch for:	
United Kingdom, Ministry of Defence, Defence Research Information Centre (DRIC)	32
United States, Department of Defense, Defense Documentation Center	33 - 44
Canada, Department of National Defence, Defence Science Information Service	45
New Zealand, Department of Defence	46
Australian National Library	47
Defence Library, Campbell Park	48
Library, Aeronautical Research Laboratories	49
Library, Materials Research Laboratories	50
Royal Australian Navy Research Laboratory (Attention: Mr H.V. Pillow)	51
Library, University of Adelaide	52
Library, Australian National University	53
Library, Deakin University	54
Library, Flinders University	55
Library, Griffith University	56
Library, James Cook University	57
Library, La Trobe University	58
Library, Macquarie University	59
Library, University of Melbourne	60
Library, Monash University	61
Library, Newcastle University	62
Library, University of New England	63
Library, University of New South Wales	64
Library, University of Queensland	65
Library, University of Sydney	66
Library, University of Tasmania	67
Library, University of Western Australia	68
Library, Caulfield Institute of Technology	69
Library, New South Wales Institute of Technology	70
Library, Queensland Institute of Technology	71

Copy No.

Library, Royal Melbourne Institute of Technology	72
Library, South Australian Institute of Technology	73
Library, Swinburne College of Technology	74
Library, Western Australian Institute of Technology	75

WITHIN DRCS

Chief Superintendent, Weapons Systems Research Laboratory	76
Superintendent, Aeroballistics Division	77
Head, Ballistics	78
Principal Officer, Ballistic Studies Group	79
Principal Officer, Flight Research Group	80
Principal Officer, Dynamics Group	81
Principal Officer, Computing Services Group	82
Principal Officer, Cybernetic Electronics Group	83
Mr P.F. Calder	84
Authors	85 - 87
Ballistic Studies Group	88 - 92
DRCS Library	93 - 94
Aeroballistics Library	95 - 97
Spares	98 - 150