Honeywell

LEVEL 66
SYSTEM SOFTWARE OVERVIEW

SERIES 60 (LEVEL 66)

SYSTEM SOFTWARE OVERVIEW

SUBJECT

General Introduction to the System Software for the Honeywell Series 60 Level 66 Information System

SOFTWARE SUPPORTED

Series 60 Level 66

ORDER NUMBER

DE61, Rev. 0

July 1976

Honeywell

PREFACE

This document presents a broad overview of Honeywell Series 60 Level 66 system software: of its capabilities and significant characteristics. It is intended for readers with data processing experience but with no specific knowledge of the Honeywell Series 60 Level 66 Information System. Furthermore, its viewpoint is that of a professional application programmer using a high-level language -- most likely COBOL. Therefore, certain aspects of the system that would be of interest mainly to an assembly-language programmer, computer operator, or technical support specialist are covered only briefly or not at all.

NOTE: Throughout this overview, the term "end user" means a user of the application programmer's developments, or of preprogrammed end-user facilities provided by the system. The unqualified term "user" is employed here to mean a professional programmer.

This overview is the highest-level document in an evolving set of introductory documents concerning the Series 60 Level 66 system software. For information regarding the availability of other manuals in this set, contact your local Honeywell Information Systems marketing representative.

APPLICABILITY OF THIS DOCUMENT TO THE HONEYWELL SERIES 6000 INFORMATION SYSTEM: Most of the information in this overview is equally applicable to Honeywell Series 6000 system software, due to the high degree of software compatibility between the Series 6000 and Series 60 Level 66 systems. The software components listed below do not apply to Series 6000 Models 6023/6030/6031/6050/6051/6070:

- COBOL-74 Compiler
- Integrated Data Store/II (I-D-S/II)
- Unified File Access System (UFAS)
- UTL2 Utility Processor
- Management Data Query System (MDQS)

File No.: 1713, 1P13

DE61

FUNCTIONAL LISTING OF PUBLICATIONS
for
SERIES 60 (LEVEL 66) and SERIES 6000 SYSTEMS

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE Series 60 (Level 66)/Series 6000:	ORDER NO.
Hardware reference:		
Series 60 Level 66 System	Series 60 Level 66 Summary Description	DC64
Series 6000 System	Series 6000 Summary Description	DA48
DATANET 355 Processor	DATANET 355 Systems Manual	BS03
DATANET 6600 Processor	DATANET 6600 Systems Manual	DC88
Operating system:		
Basic Operating System	General Comprehensive Operating Supervisor (GCOS)	DD19
Job Control Language	Control Cards Reference Manual	DD31
Table Definitions	System Tables	DD14
Table Definitions	NPS Tables	DE34
I/O Via MME GEINOS	I/O Programming	DB82
System initialization:		
System Startup	System Startup	DD33
System Operation	System Operation Techniques	DD50
Communications System	GRTS/355 and GRTS/6600 Startup Procedures	DD05
Communications System	NPS Startup	DD51
Communications System	NPS Configuration Examples	DE76
DSS180 Subsystem Startup	DSS180 Startup (Series 6000 only)	DD34
Program Recovery	Program Recovery/Restart	DC98
Data management:		
File System	File Management Supervisor	DD45
Integrated Data Store (I-D-S)	I-D-S/I Programmer's Guide	DC52
Integrated Data Store (I-D-S)	I-D-S/I User's Guide	DC53
File Processing	Indexed Sequential Processor	DD38
File Input/Output	File and Record Control	DD07
File Input/Output	Unified File Access System (UFAS) (Series 60 only)	DC89
I-D-S Data Query System	I-D-S Data Query System Installation	DD47
I-D-S Data Query System	I-D-S Data Query System User's Guide	DD46
Coexistent I-D-S	Coexistent I-D-S Overview	DE60
Program maintenance:		
Object Program	Source and Object Library Editor	DD06
System Editing	System Library Editor	DD30
Test system:		
Online Test Program	Total Online Test System (TOLTS)	DD39
Test Descriptions	Total Online Test System (TOLTS) Test Pages	DD49
Error Analysis and Logging	Honeywell Error Analysis and Logging System (HEALS)	DD44
Language processors:		
Macro Assembly Language	Macro Assembler Program (GMAP)	DD08
COBOL-68 Language	COBOL Reference Manual	DD25
COBOL-68 Usage	COBOL User's Guide	DD26
Standard COBOL-68 Language	Standard COBOL-68 Reference Manual	DE17
Standard COBOL-68 Usage	Standard COBOL-68 User's Guide	DE18
JOVIAL Language	JOVIAL	DD23
FORTRAN Language	FORTRAN	DD02
Macro Assembly Language	DATANET 355/6600 Macro Assembler Program	DD01

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE Series 60 (Level 66)/Series 6000:	ORDER NO.
Generators:		
Sorting	Sort/Merge Program	DD09
Merging	Sort/Merge Program	DD09
Simulators:		
DATANET 355/6600 Simulation	DATANET 355/6600 Simulator	DD32
Service and utility routines:		
Loader	General Loader	DD10
Utility Programs	Utility	DD12
Utility Programs	UTL2 Utility Routine (Series 60 only)	DC91
Media Conversion	Bulk Media Conversion	DD11
System Accounting	Summary Edit Program	DD24
FORTRAN	FORTRAN Subroutine Libraries	DD20
FNP Loader	DATANET 355/6600 Relocatable Loader	DD35
Service Routines	Service Routines	DD42
Software Debugging	Debug and Trace Routines	DD43
Time Sharing systems:		
Operating System	TSS General Information	DD22
System Programming	TSS Terminal/Batch Interface	DD21
System Programming	TSS System Programmer's Reference Manual	DD17
BASIC Language	Time Sharing BASIC	DD16
FORTRAN Language	FORTRAN	DD02
Text Editing	Time Sharing Text Editor	DD18
dataBASIC Language	dataBASIC System Language Manual	DD95
dataBASIC Loading	dataBASIC Load/Unload System	DD96
Remote communications:		
DATANET 30/305/355/6600 FNP	Remote Terminal Supervisor (GRTS)	DD40
DATANET 355/6600 FNP	Network Processing Supervisor (NPS)	DD48
DATANET 355/6600 FNP	NPS Micro-Ops Programming	DE35
DATANET 355/6600 FNP	Application Guidelines for NPS	DE77
DATANET 700 RNP	RNP/FNP Interface	DB92
Communication facilities:		
COBOL-74 Communications	Message Control System Site Manual	DC99
Transaction processing:		
User's Procedures	Transaction Processing System User's Guide	DD41
Handbooks:		
System-operator communication	System Console Messages	DD13
Error Messages, Abort Codes	Error Messages and Abort Codes	DC97
NPS Error Messages	NPS Error Messages	DE75
Pocket guides:		
Control Card Formats	Control Cards and Abort Codes	DD04
FORTRAN	FORTRAN Pocket Guide	DD82

CONTENTS

ILLUSTRATION

SECTION I

INTRODUCTION

A Honeywell Series 60 Level 66 Information System consists of two distinctly different kinds of "components:" hardware and software. The software components are the system programs specifically developed by Honeywell to complement the Level 66 system hardware. (The term "system software" as used here excludes the application-oriented programs and packages also available from Honeywell.)

The system software is designed to facilitate efficient and convenient use of the hardware, and to expand the capabilities of the system as a whole. The general objectives of the system software are the following:

1. To insulate the user from the detailed, machine-level characteristics of the system, thereby making it easier to use.

2. To aid the operator and site manager in the overall operation of the system.

3. To maximize throughput, i.e., to allow the most efficient utilization of the total system's resources.

These objectives and their implications are discussed further in Section II.

This overview does not attempt to teach the use of the software, but rather describes and explains its major functions, its overall organization, and its particular terminology.

In addition to providing a general introduction to the subject, this overview should serve prospective users of the system as a pathway to more detailed levels of documentation for further study.

SYSTEM SOFTWARE CATEGORIES

From a user's viewpoint, the system software can be divided roughly into three categories:

- Operating system (GCOS)
- Language and utility processors
- Interactive subexecutives

Although all of the software in these three categories is functionally (or otherwise) related, an understanding of the operating-system functions and facilities is prerequisite to the actual use of the other two categories of software (with few exceptions). Therefore, the central emphasis in this overview is on operating-system software. Unlike the highly interrelated components of the operating system, the various processors and interactive subexecutives can be viewed as independent, self-contained entities, and are so treated in the reference level documentation.

In addition to system software, Honeywell also makes available several libraries of general application programs. These programs are designed to satisfy various processing requirements common to many computer installations, and often can be tailored to meet a customer's specific needs. A representative sample of the application programs available is described in the "Series 60 Level 66 Summary Description" document, Order No. DC64. (For information on the complete range of currently available application programs, contact your local Honeywell Information System sales representative.)

SECTION II

ROLE OF SYSTEM SOFTWARE

This section identifies the functions performed by each category of Level 66 system software (operating system, language and utility processors, and interactive subexecutives), as they relate to system capabilities.

A note on terminology and categories of software: The acronym GCOS stands for General Comprehensive Operating Supervisor, which is the proper name of the Level 66 operating system. Regarding the software categories used in this overview, it is sometimes hard to get general agreement on what is and what is not operating-system software in certain areas, primarily because the decision is viewpoint dependent. That is, it depends upon whether one is looking at the system from the "inside out" (technically) or from the "outside in" (functionally). In this document we shall take a user-oriented viewpoint wherever necessary and make the distinction on functional grounds.

ROLE OF THE OPERATING SYSTEM

The operating system, GCOS, manages the system's resources and provides the basic service and control functions utilized by all other software and by user programs. The essential purpose of resource management and the service/control functions, respectively, is to:

1. Maximize system utilization - Allow a given system's resources to be shared most efficiently by the largest number of users, and reduce operator interaction to a minimum.

2. Reduce overall programming burden - Provide centralized, machine-level coding sequences for initiation and control of basic system services including: I/O operations; file creation and access; dynamic memory and secondary-storage acquisition and release; program, module, and overlay-segment loading and intercommunication; exception and error condition handling; dynamic peripheral allocation and de-allocation; and abnormal program termination.

Resource Management

The resource-management activities are largely automatic, in that they are performed invisibly on behalf of all other software and user programs and, in general, without any explicit request from the program affected. For example, both user and system programs are regularly "swapped" in and out of memory during their execution cycle when conditions warrant or require such action: usually when a program is temporarily incapable of further processing (e.g., awaiting an I/O completion), or when the memory space of a low-urgency program is preempted by a high-urgency program. Such swapping is transparent to the program processes affected and, excepting certain system programs, is not controlled by the programs affected. Swapping is also referred to as "roll-in/roll-out." (NOTE: This is not a universal capability of all contemporary, large-scale operating systems.)

Other automatic resource-management functions include:

- Initial peripheral device allocation, and deallocation on program/job termination.
- Scheduling of programs/jobs on a resource-requirement plus urgency basis.
- Initial memory and temporary mass-storage space allocation, and its deallocation on program termination.
- Switching of central-processor control (or "execution time") between programs in memory.
- Connection, management, and disconnection of communication lines for time sharing, transaction processing, and remote-batch processing.
- Dynamic error-analysis and retry.
- Normal or abnormal program/job termination, with 'cleanup' as required.

Service And Control Functions

The basic system service and control functions provided by the operating system are utilized primarily by other system software (e.g., compilers), and secondarily by some user programs coded in assembly language. The higher-level-language programmer does not explicitly request these functions, although many of them are invoked indirectly on behalf of such users' programs. Therefore, as a coding interface, these basic functions performed by the operating system are invisible to the normal user.

Some of these functions, however, are explicitly but indirectly requested by the user through his standard interfaces with the operating system: the job control language (JCL) for batch operations, and the Time Sharing System commands for online operations. Although the inherent nature of user operations differs somewhat in each of these usage modes, the user can request through either interface, for example, the loading and execution of a program; he can request creation of and access to a permanent file, creation of a temporary file, saving of a temporary file on a permanent file, and deletion of a permanent file; and he can request abnormal termination (abort) of a program process because of excessive execution time or excessive output. He can request common utility functions, such as automatic media or code conversions, in either mode.

In the batch mode and to a lesser extent under time sharing, he can request special program-loading options, e.g., for testing developmental programs with diagnostic aids and selective "dump" conditions, or for running segmented, overlay programs. (In general, more system-default options are automatically exercised in the time sharing environment, for greater user convenience.)

It might be noted here that time sharing operation under Level-66 GCOS has been engineered for the maximum human convenience of the greatest number of users (programmers and non-programmers alike), rather than to mirror batch-world operations as is sometimes the case with other operating systems. For the time sharing user who needs the maximum degree of flexibility, however, the full array of batch capabilities are available through the TSS Terminal-Batch facilities. (Also, recall that the GCOS file system is common to both batch and online modes, and the GCOS system-standard file formats are compatible in both modes.)

Centralization of the basic system service and control functions in the operating system serves a twofold purpose (one of which also relates to the resource-management function). The provision of generalized, machine-level coding sequences for physical I/O initiation, for memory and mass-storage space allotment, for file-system services, etc. -- with simplified, higher-level interfaces to these sequences provided for all other levels of software -- obviously reduces the amount of detailed coding effort required throughout the system (both in software and user programming).

Centralized initiation of basic system functions on behalf of many programs allows centralized supervision of the resulting 'state of affairs.' Control of I/O interrrupts is a good example. Because physical I/O operations are highly complex and, once initiated, are carried out independently of and asynchronous to the computational operations in the processor, one or two interruptions of program execution in the processor will be caused by each I/O operation. (These interrupts can signal either a normal or abnormal termination of the operation.)

When one multiplies the one or two interrupts per operation by the number of simultaneous I/O operations possible in a large-scale system (commonly on the order of 15 to 20), one can readily see that to allow each program to service its own I/O interrupts would quickly result in chaos, at least without an impossibly burdensome set of programming rules and regulations and interprogram communication requirements. However, by virtue of initiating all I/O operations, the operating system can monitor and service all resulting I/O interrupts in an orderly and methodical fashion.

It can be shown that a close analogy also exists with memory, mass storage, and peripheral management. It is precisely this centralized supervision-through-control of basic system functions that enables the central operating software to exercise its overall resource-management functions.

ROLE OF LANGUAGE AND UTILITY PROCESSORS

The role of language and utility processors is specifically to reduce user-programming effort, by radically reducing the amount and level of coding detail required to produce programs and to achieve file-maintenance operations, respectively. An additional role fulfilled by certain standardized, higher-level language processors (e.g., COBOL) is to facilitate program transferability. That is, they facilitate the shifting of existing programs from one computer environment to another.

Obviously, this greatly lessens the reprogramming effort involved in changing from one computer line (or one operating system) to another. But another significant benefit is that it also allows the sharing of programs among users with similar problems or interests, but who use different computer systems. Thus we have a second degree of reduction of programming effort.

Also included in the language/utility processor category is the Sort/Merge Package, which performs the commonly required functions of sorting and merging input data files.

Language Processors

Language processors translate source statements written in a well defined version of a given higher-level programming language (e.g., COBOL, FORTRAN, PL/I) into an object program. In normal practice, beside coding his source program the only other interaction the user has with the underlying system software is through the JCL. He uses the JCL to request compilation and/or execution of his program, to relate file codes defined in his program to specific types of file media or to specific permanent files, and to request creation of permanent files as required.

Utility Processors

Utility processors, on the other hand, perform a wide variety of common file-maintenance and file-conversion functions. The specific operations performed by a given utility processor are determined by "directives," or control-card statements similar to JCL but understood only by the utility processor in question. The number and complexity of the directives required varies in direct proportion to how standard or nonstandard the desired maintenance or conversion operation is (relative to system-standard file formats). A few simple directives suffice for the most-often-required utility operations.

No programming as such is needed to use the capabilities provided by the utility processors, which are powerful and varied, especially in the area of file maintenance. Often an operation that would otherwise require the coding of a nontrivial COBOL program can be accomplished almost automatically by intelligent use of the proper utility function.

Sort/Merge Package

The Sort/Merge package provides very flexible and highly efficient sorting and merging of input data files. The specific sort or merge processing performed for the user is controlled by user-supplied descriptive parameters, e.g., record size, size and type of sort key, type of key comparison desired. Relatively few descriptive parameters are required for the average sort or merge application.

The Sort/Merge package is similar to a utility processor in that it performs several frequently required data-processing functions. It is unlike the utilities, however, in that it is not an independent processor controlled by JCL directives, but is a highly adaptive system program that is invoked within a user's application program. The sort/merge functions may be invoked, for example, with a COBOL SORT verb, either in a COBOL program or in a COBOL subroutine within a program written in some other language.

ROLE OF THE INTERACTIVE SUBEXECUTIVES

The interactive-subexecutive category of software pertains to the areas of information retrieval, transaction processing, and dynamic data base management. Specifically, the role of the interactive subexecutives is to aid the user in implementing applications in the above-mentioned areas, to provide the manifold support functions required for such applications, and in some cases to provide direct, high-level, end-user language facilities. In the area of transaction processing, for example, the subexecutives allow the programming of application-specific processes in a common, higher level language such as COBOL. These programs are then readily integrated into the "open ended" user portion of the subexecutive for automatic invocation and execution as needed.

In the area of data base management (DBM), the subexecutives combine data definition (schema) facilities, application definition (subschema) facilities, and data base content-manager software (i.e., sophisticated access methods which "know" the structure of the data base), with appropriate communications management functions for remote access. The most advanced types of DBM subexecutives also provide a specialized procedural language for the end user (assumed to be a non-programmer). The intent of the direct end-user facility is to allow a quick-reaction capability for servicing unplanned and unanticipated information-processing requirements. In essence, they allow the end user to devise "on the spot" procedures for data base query and update as required in a dynamic operational environment.

In each of these areas, the system software underlying the application-specific programs (or user-defined processes) automatically provides many critical services, such as:

- Routing of messages between a terminal and an appropriate user-defined process.
- On-demand initiation and termination of such processes as required.
- Physical manipulation of the data base (i.e., data base content management).
- Dynamic storage-space management.
- Selectable levels of journalizing.
- Control of simultaneous-access conflicts, e.g., concurrent attempts to modify the same information item.
- Protection against minor disturbances of the data base resulting from random human error or a system error.
- Protection against potentially catastrophic damage to the data base due to unlikely-but-possible major system failure.

These services are provided in both the transaction-processing and data base management areas.

SECTION III

OPERATING SYSTEM (GCOS)

This section describes the user's interfaces to Level 66 GCOS services and facilities, and the major software structures that provide these services.

THE USER INTERFACE

The user's interface with GCOS takes one of two forms, depending upon the system-usage mode:

- Job Control Language (JCL) statements, for local/remote batch operations. (The JCL statements are also commonly referred to as "control cards," since the JCL statement format is based on the 80-column punch-card image.)
- Time Sharing System (TSS) dialog, for time sharing or "online" operations. (The TSS dialog consists of an interactive question/answer sequence between a remote-terminal user and the system.)

Due to the inherent differences in the man/machine interaction involved in batch vs. time sharing usage, the two interfaces seem quite different but, as we will show, they invoke most of the same basic system services.

Ease-Of-Use Characteristics Of GCOS JCL

GCOS provides a progressive, easily learned job-control language. New users can employ a limited subset of the full JCL in a very simple, straightforward fashion. In this elementary form of JCL usage, the system automatically supplies many default options. Relatively few JCL statements are required for basic use of the system.

As the user's familiarity with the system grows, he can exercise a wide variety of explicit options -- taking advantage of the system's flexibility -- simply by adding either keywords or additional statements to his original job-control sequences. Furthermore, for the experienced user's greater convenience and fullest utilization of system features, he can begin to use the JCL as a semiprocedural language, with features such as conditional statement-execution (branching), statement nesting (cataloged procedures), and dynamic modification of parameterized statements. These advanced features also build upon the same elementary-usage JCL sequences, but the elegance and sophistication of the user's "programming" capability in GCOS JCL is limited only by his inventiveness.

Ease-Of-Use Characteristics Of The TSS Dialog

The various GCOS Time Sharing subsystems are intended to accommodate secretaries, typists, technicians, accountants, managers, scientists, etc., as well as professional programmers. Therefore, the TSS command-level dialog is as simple and natural as possible, and is designed so that a user of only one or two end-user subsystems, e.g.:

- Text Editor and RUNOFF (A powerful text entry, editing, and display tool)
- ABACUS (A convenient algebraic "desk calculator")
- BASIC (A direct, easy-to-learn, mathematical problem-solving language)
- I-D-S Data Query (A very simple means of data base information retrieval)

need not learn the full extent of the TSS command language. (Much of the TSS command dialog involves program-preparation aids and file-system related operations, which are of great interest, however, to the professional application programmer.)

The overall organization of the time sharing user's interface is such as to encourage the user's growth and an easy, natural transition from simple to sophisticated usage of the system, while performing useful work along the way.

Interfaces To Job-Management Services

Job management essentially consists of the following actions performed by the system:

1. Recognition of a user's batch job or time sharing session (the two are conceptually identical) as an identifiable entity; that is, as a valid unit of work or a related set of such units associated with a given user.
2. Acceptance or rejection of the job (or session) based on verification of an authorized account number and/or user-ID and password, as determined by the installation, i.e., by the site-operations manager.
3. Scheduling of the job, based on its resource-requirement profile, and possibly on a relative priority value -- or 'urgency level' -- requested by the user (if he is so authorized).
4. Supervision of the interrelationships between the several activities, or steps, of the job (if multiple activities exist). These interrelationships include resources already allocated to a previous activity (e.g., memory, peripherals), normal or abnormal termination of the previous activity, etc.
5. Collection of system-generated output, i.e., execution reports and program listings, from the several activities.
6. Proper termination of the job after the successful or abnormal completion of its activities. This includes internal "cleanup" of system information concerning the job, and compilation of accounting information.

Since job management includes both the initiation and termination of a job (or session) and its between-activity transitions, it can be seen as the "umbrella" service for all of the other services provided by GCOS. In the batch world, the basic JCL statements -- or control cards -- that explicitly invoke job-management services are known as job-definition statements:

```
$       SNUMB  sequence-number, priority
$       IDENT  accounting-info,name,location,etc.
[$      USERID user-ID and password
 $      LIMITS resource-requirement values
 $      BREAK                                    ]
$       ENDJOB
```

The three statements outside the square brackets are mandatory; that is, they must be included as the first two and the last JCL statements, respectively, of every job deck. The first statement, $ SNUMB (for sequence number), supplies an arbitrary job identifier used by the system through the processing of the job, both internally and on listings, execution reports, operator messages, etc. (This statement is often supplied by the installation in the form of a prepunched, preprinted punch card, for the user's convenience and to forestall duplicate sequence numbers). Use of the optional priority field in this statement is controlled administratively by the installation, or automatically if the batch job is submitted via time sharing (the TSS CARDIN subsystem).

The $ IDENT card supplies an account number, as required by the user's site management. This statement also supplies a free-form name, e.g., "Joe Jones", and possibly the user's location, output-delivery station code, telephone number, etc. The first nine characters of the name field are used as a banner on printed or punched output for the job (i.e., Joe Jones).

The $ ENDJOB card signals the end of a job-control deck. The functions of the optional JCL statements shown within the square brackets are as follows:

$ USERID - This statement supplies the user's formal identification as it is known to the file system (e.g., "J.P.JONES"); it is required only when the user wishes to create and/or access permanent files, i.e., files cataloged within the file system. (The user-ID is identical to the one supplied by an authorized time sharing user during his TSS log on.)

$ LIMITS - This statement, if used only at the beginning of a job deck, describes the aggregate resource-requirement profile for the job. Except for permanent production-job setups, the use of this statement is required only occasionally, e.g., when an exceptionally large or long-running program is to be executed. (The system-default limits are normally adequate for program-development jobs.) In certain cases, however, it can optionally be used to effect more efficient job scheduling, allocation, and execution.

$ BREAK - This statement can be used to separate subsets of a job's activities into "mini-jobs" such that failure of a preceding activity will not force a deletion of subsequent activities. This statement also is seldom needed, due to automatic default conventions for conditional activity execution, based on types of activities.

In time sharing mode, the "job definition" information required of a user is radically simplified, as is both possible and appropriate in this mode of usage. After the user achieves a physical connection between his terminal and the system (normally by dialing an appropriate telephone number), the system identifies itself and requests a USER ID and PASSWORD. The user responds by entering his authorized time sharing user identification and password.

This is all that the time sharing user need do to identify the terminal session that is about to commence. He may, however, follow his user-ID with an account number to which this session is to be charged; otherwise the default account number associated with his user-ID is assumed. At the end of his session, he gives the command BYE, which causes accumulated accounting charges to be compiled and reported to him, and terminates the session.

During the session, he can use a "break" signal (e.g., the ATTN, BREAK, or INTERRUPT key, depending on terminal type) to "break off," or abnormally terminate, a process associated with a given activity, an overly long listing or a program loop, for example, without having to terminate the session.

In general, the items described above (the user-ID and password, possibly an explicit account number, and the "break" signal), comprise the user's interface with TSS job-management services. During a session, use of the STATUS command is occasionally helpful in determining various data about the user's resource usage up to that point in the session.

Interfaces To Program-Execution Services

An activity of a job or terminal session is defined most generally as the execution of a program, whether that program be a system program, a user program, or a Time Sharing subsystem. In batch usage, there are various ways of requesting a program execution, depending upon the nature of the program and, if a user program, its form. The JCL statements that request program execution are categorized as activity-defining statements. Typically these include:

```
$       COBOL     (COBOL-68  compiler)
$       CBL74     (COBOL-74 compiler)
$       CONVER    (Media-conversion utility processor)
$       FORTRAN   (ANS Fortran compiler)
$       FILEDIT   (Source/Object Library Editor program)
$       FILSYS    (File Management System processor)
$       GMAP      (Macro-assembly language processor)
$       IDS2      (Integrated-Data-Store II Translator, a translator
                    for data base definition statements)
$       PL1       (PL/I compiler)
$       UTL2      (Standard utility processor for file-maintenance functions)
$       EXECUTE   (Request for loading/execution of a user's object program)
$       PROGRAM   (Request for loading/execution of a program in
                    system-loadable format)
```

(This is not a complete list of activity-defining statements in respect to system-provided programs and processors. See Section IV for a description of the available language processors, for example.)

All of the statements listed above invoke the execution of a system program except for the last two. The system programs process either the user's source-language programs or user-defined directive statements recognized by a given system or utility processor. The last two statements, $ EXECUTE and $ PROGRAM, request loading and execution of user-prepared programs.

The $ EXECUTE statement invokes the General Loader, a system program, to load a user's object program (the output of a language processor) for execution. The program to be loaded may physically precede the $ EXECUTE statement, in the form of an object-program card deck, or may be obtained from a file identified in a variety of ways within the scope of the activity (i.e., by one of the file-defining statements discussed below). A $ OPTION statement must precede the execute-activity JCL for most higher level language programs, to identify the language processor that was used for compilation. This statement is also used to request one or more of the numerous nonstandard loading options recognized by the loader. The standard-default loading options are satisfactory for the majority of cases, however.

The $ PROGRAM statement requests the rapid loading of a user or installation program that is in a special system-loadable format, usually produced by a System Library Editor (SYSEDIT) process. The standard-system-loadable format is generally used for production programs or other programs with a high frequency of use; these programs usually reside on the system library, on an installation library, or possibly a user's private library. These programs are loaded by a small, high-speed system loader that does not perform any of the program-binding or relocating functions of the General Loader. (These functions are performed during the SYSEDIT processing.)

Interfaces To File Management Services

File management services comprise the creation, accessing, and release of all types of files: temporary files; user-managed, uncataloged "permanent" files; and system-managed, cataloged permanent files. (Note that these are not file media classifications in themselves. By virtue of standardized file formats, each of these file types can exist on several kinds of recording media.)

The JCL statements that request the various file-management services are categorized as file-definition statements. This category of the JCL contains the largest number of statement types, which reflects the power, variety, and importance of GCOS file management in the overall GCOS context. Nevertheless, the vast majority of these services are completely transparent to a new user who is utilizing GCOS in an elementary fashion. For example, standard system-file defaults are automatically exercised by the system for all activities in which the user supplies punch-card input and requests punch and/or printer output. In general, all normal temporary file requirements are automatically satisfied by the system, while any desired permanent file or private tape/disk file must be requested by the user.

Note that all JCL and TSS interfaces with file-management services concern file space management and/or peripheral-device handling, although file management includes basic file-content access services -- or logical access methods -- as well. (To the higher-level language user the invocation of logical access-method software, along with physical I/O services, i.e., IOS, is transparent, since this is done for him by his source-language compiler.)

The more-commonly used JCL statements in the file-definition category are as follows:

```
$       PRMFL
$       FILE
$       TAPE
$       400PK
$       451PK
$       SYSOUT
$       REMOTE
$       DATA
```

All of the JCL statements listed above must specify, as their first variable-field parameter, a file code. This file code, consisting of two alphanumeric characters, is central to the overall GCOS file-management strategy. Essentially it relates a logical file defined within some program (whether a system or user program), to some real-world physical file space or peripheral device. For example, a user's COBOL program will assign a two-character file code to a file named in a SELECT sentence and defined in its Data Division statements. By the use of a file-defining statement in his JCL for the execution of that program (after its compilation), the user associates that file code with a physical file or device. Thus the file code is the intermediary symbol used in both the program and "run time" definitions of a file.

The keyword of the JCL statement implies the general nature of the physical file being defined, e.g., PRMFL implies a permanent, cataloged file (usually on system-controlled mass storage), FILE implies a temporary file, TAPE a magnetic-tape file, 400PK a disk-pack file, etc. The variable field of the statement (following the file code) supplies further specifics about the actual file or file type desired, and about inter-activity relationships involving that file (if any).

Among the limited list of file-defining statements given above, $ PRMFL differs uniquely from the others. The $ PRMFL statement refers to permanent files, i.e., files known to and cataloged under the GCOS File Management Supervisor (FMS). As mentioned above, this implies that the permanent file thus referred to must have previously been created by the user. This file creation is achieved either by means of a $ FILSYS (File System) activity -- using a FCREAT directive or one of its several variants, or alternately by use of the Time Sharing ACCESS subsystem. It is important to note here that the file "creation" involved in either of these activities only names the file, describes some of its attributes, and reserves some initial amount of file space (with no meaningful content). Therefore, the first meaningful use of that file must be as an output file, to establish its initial content.

In time sharing mode an even simpler means of creating a permanent file is also available: the SAVE command. This command not only creates a named, permanent file, but also copies into it the contents of the TSS user's "current file" (the primary temporary file automatically supplied for a TSS user). That is to say, the SAVE command literally saves the user's TSS current-file content in a permanent file named by the user in the SAVE command. (A complementary RESAVE command is used to replace the contents of an existing permanent file.)

The $ SELECT statement also refers to a permanent file, but is not a true file-defining statement in that it does not define a logical file code. It defines, instead, a point in a JCL statement sequence at which the content of the referenced permanent file is to be copied during job input processing. Its intended function is that the $ SELECT statement itself is to be replaced by the contents of the permanent file during initial job-deck processing. Essentially it permits the cataloging of JCL segments (i.e., cataloged procedures), though it may be used for other purposes. Thus it can be seen that the following very simple JCL sequence:

```
$       SNUMB     ....
$       USERID    ....
$       SELECT    file-identification
$       ENDJOB
```

can be used to submit a complete and possibly very complex and voluminous job for execution. The file referred to would contain at least one $ IDENT statement and any number of activity-defining sequences, including other $ SELECT statements. (Nesting of $ SELECT statements is possible to a depth of 10 levels.)

GCOS file management is a very extensive subject, with wide-ranging implications. Rather than attempting to cover the subject exhaustively, the intent here has been to give you an idea of the essentially straightforward strategy employed and, especially, of its easy-to-use aspect for the novice.

Interfaces To Data Base Management Facilities

Data base management facilities under GCOS come in various forms. They fall into the following four general categories, however:

1. Batch data base manipulation languages for the data base application programmer (e.g., I-D-S/I or COBOL-74).

2. Data definition and application definition languages for the data-base administrator, provided by I-D-S/II or the Management Data Query System (MDQS).

3. Sophisticated data base access method software, which "understands" the structure of the data base. (This software is automatically invoked by the user's language facility.)

4. Interactive end-user facilities (e.g., MDQS, dataBASIC, I-D-S Data Query), which are normally invoked through the TSS command language. Each of these facilities provides a simple procedural or semiprocedural language for online query and/or updating of a data base. (A user's 'private' data base in the case of dataBASIC.)

As is implied by categories 1, 2 and 4 above, the user's interfaces with these data-management facilities consist essentially of the JCL or TSS program-execution services described earlier. For example, in the batch world the $ IDS2 statement would be used to invoke the I-D-S/II translator for processing a set of data base definition statements. The $ CBL74 statement would be used to invoke the COBOL-74 compiler to process a source program for manipulating an I-D-S/II data base.

In time sharing mode, the MDQ command would be used to invoke the end-user's interactive language facility associated with MDQS.

Interfaces To Data Display Facilities

GCOS data-display facilities are primarily associated with time sharing usage, and and include the following Time Sharing subsystems:

1. APRINT - A subsystem that prints an ASCII time sharing file on an ASCII high-speed printer at the central computer site.

2. BPRINT - A subsystem that prints an ASCII time sharing file on a standard BCD high-speed printer at the central computer site. (Automatic code conversion is performed, and special-character graphics are transliterated when possible.)

3. BPUNCH - A subsystem that punches an ASCII time sharing file (e.g., a JCL "deck" in one of the several convenient formats known to various TSS subsystems) on a card punch at the central computer site. (Automatic code conversion is performed, and special-character graphics are transliterated when possible.)

4. LIST - A very powerful and flexible subsystem that prints the contents of one or more ASCII time sharing files, or any portions thereof, at the user's terminal device. Many variations of the basic LIST command are available; their optional functions include:

   - o File-content concatenation
   - o File-content merging, by line numbers
   - o Concatenation and/or merging of multiple files
   - o Line-folding of overlong lines
   - o Listing with a date/time heading
   - o Conditional listing, by line numbers
   - o "Sampling," i.e., listing of every nth line of a file.

   In all of the variant cases described, the source files are not themselves modified or otherwise affected.

5. RUNOFF - A sophisticated subsystem that reformats a time sharing text file (e.g., a letter, a report, or a lengthy document) according to control commands recognized by RUNOFF that are imbedded or inserted in the file. (For example, the control command ".SPAC 4" will cause four blank lines in the reformatted file, in place of the command itself). Right and left margins can be automatically justified, page length and width is readily modifiable, automatic tabulation can be performed, etc. The reformatted file text is either printed at the user's terminal, or is saved on another file, e.g., for printing on a higher-speed output device.

   The RUNOFF subsystem is normally used on files built and edited via the Text Editor subsystem.

The user's interface for invoking any of these data-display facilities are TSS commands that are, in fact, the names of the subsystems listed above (or the first four characters of the name, i.e., RUNO for RUNOFF).

Interfaces To Language and Utility Processors

Many language and utility processors are provided under GCOS. The range of such processors includes:

- Highly developed compilers for all of the popular higher-level languages, e.g., ALGOL, BASIC, COBOL-68, COBOL-74, FORTRAN, JOVIAL, PL/I.
- A powerful macro assembler, GMAP, for assembly-language programming.
- A macro assembler and a simulator for programming either the DATANET 355 or DATANET 6600 Front-End Network Processors (355MAP and 355SIM).
- A Bulk Media Conversion (BMC) processor, which performs a wide range of file-media conversions in both standard and nonstandard file formats.
- A Source and Object Library Editor and a System Library Editor, for building and maintaining user libraries, production libraries, and system libraries.
- Two generalized file-maintenance processors (UTILITY and UTL2) which provide very convenient, flexible, and comprehensive file maintenance/display functions.

The user's JCL interface for invoking any of the language or utility processors consists essentially of the program-execution services (activity-defining statements) described earlier. In time sharing mode, a number of the widely-used language compilers can be executed directly via the TSS command language, but all of the processors except BASIC can be invoked as a batch activity through the TSS Terminal/Batch interface (CARDIN subsystem).

The full range of language and utility processors is discussed further in Section V.

Interfaces To Program Preparation Facilities

"Program preparation facilities" are features of the system that aid the user in (1) creation and modification of source-program text, and (2) maintenance of program libraries (source or object). Since program-library maintenance (item 2) is largely a function of the Source and Object Library Editor, a utility processor, it is dealt with under the latter heading. We will concentrate here on item 1, source-program creation and modification aids.

In the batch environment, the creation of source-program text in a system-readable form (a card deck, or a tape or disk file) is generally performed offline by data-entry personnel, on a keypunch or other form of key-data device. For convenient program modification -- during program development and testing, for example -- the system provides a standard Update/Alter facility that can be used in any language-processing activity. By use of the $ UPDATE and $ ALTER JCL statements in a compilation activity, the user can submit individual source statements or sets of such statements for automatic incorporation into either a normal or a compressed (COMDK) source file, prior to its recompilation. (At the point in time that the insertion, replacement, or deletion of source statements is desired, the original source program usually has gone through at least one compilation and has been placed on a permanent mass-storage file.)

In time sharing mode, however, it is assumed that the programmer (or a coding assistant) will usually "write" his source program directly onto a permanent file via his terminal device. Therefore TSS provides many ease-of-use features expressly designed to facilitate program-file building, manipulation, and display. A standard (or user-chosen) set of tabulation characters can be employed to allow source-statement entry in a convenient, blank-suppressed form; these tab characters and their values are recognized throughout the TSS program-preparation subsystems and facilities.

A majority of the program-preparation convenience features are based on the concept of line-numbered source files. (The line numbers can be automatically stripped, moved to a sequence-number field, or left "as is" prior to source-program processing, depending on the format requirements of the particular source language.) The line numbers can be automatically generated by the system, in several different forms, at the user's option. Line-numbered source files allow many convenient file-manipulation features, among which are:

- Concatenation of selected portions (or all) of several source files.
- Merging of several files, or selected portions thereof, based on line numbers.
- Any combination of the merging and concatenation processing.
- Insertion, replacement, or deletion of source lines (or sets of lines), by line numbers.
- Easy replication of common coding sequences at different points in a program.

All of these features are complementary to the powerful, generalized editing capabilities of the TSS Text Editor which allow, for example, the automatic 'global' replacement of a given parameter value, say "ENTRYA1", by another value, say "ENTRYB1", wherever it may appear throughout a program (or optionally, wherever it may appear within a specified context).

The user's interface with each of these facilities (and note that the above description is not exhaustive) is through the TSS command language, which provides intelligible and easily-remembered command mnemonics for the desired operations. For example, to request a concatenation of two permanent source files ('old' saved files) onto the user's new current file (a temporary working file); and then a resequencing of this current file with new line numbers, starting conventionally with 010 and incrementing by 10, the user would employ the following two TSS commands:

```
OLD FILE1;FILE2
RESE                (for RESEQUENCE)
```

This pair of TSS commands would be recognized and executed under any number of TSS subsystems, e.g., under EDITOR, under BASIC (an interpretive time sharing language processor), under FORT (the time sharing FORTRAN interface), or under CARDIN (the Terminal/Batch interface).

Many variants of the OLD and RESE commands (for sake of example) are available, all of which build upon the most basic form of the command. Thus, the basic form of the OLD command is

```
OLD filename
```

whereas the form

```
OLD filename1;filename2
```

is a simple variant requesting concatenation. As mentioned previously, the TSS command language is a progressive, easy-to-learn structure of system/user interaction.

SYSTEM SOFTWARE STRUCTURE

This subsection describes the major software components that provide the operating-system services discussed above. The component descriptions are organized by the categories of service which they support.

Job Management Services

BATCH JOB MANAGEMENT

The batch job-management services control the flow of batch jobs through the system and, as noted previously, are basic to all other system services. The system components that provide these services are best described by tracing the flow of a single job through the system, and identifying the components in a dynamic context.

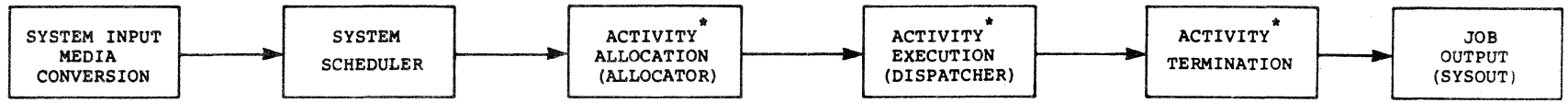
Job Flow

The description of job flow given on the pages following is also summarized in Figure 3-1. The reader should refer to this figure while reading the description. All batch jobs entering the system pass through six phases of job flow. These are

1. System Input Media Conversion
2. Job Scheduling (Scheduler)
3. Activity Allocation (Allocator)
4. Activity Execution (Dispatcher)
5. Activity (or Job) Termination (Termination-Module Group)
6. Output Media Conversion (SYSOUT)

Note that phases 3, 4, and 5 are activity oriented, rather than job oriented phases. Thus these phases are repeated for each activity of the job.

In each phase, special functions are performed which control the logical flow of the job through the system. The components named in parentheses are the system-software programs or modules primarily responsible for performing the functions involved in each phase.



- Performs gross control-card validation
- If valid, sends job to the System Scheduler

- Places job in a prioritized scheduling queue

- Selects job from a scheduling queue
- Performs detailed control-card analysis
- For each activity:
  - o Allocates needed peripherals
  - o Allocates needed memory
  - o Enters activity into the Dispatcher's queue

- Initializes hardware/software registers for program
- Loads program into memory
- Dispatches processor control to program, on cyclic basis
- Swaps program out to mass storage if required

- Initiated when activity terminates (normally or abnormally)
- Writes accounting statistics for activity on SYSOUT device
- Looks ahead to next activity
- Passes control to Allocation, Execution, or Output phase depending on type of next activity (if any)

- All activities of the job have been processed
- Output collected for each activity is disbursed to a printer or punch device

* Activity oriented phases

Figure 3-1. PHASES OF JOB/ACTIVITY FLOW

An overview of each phase of job flow follows.

System Input Media Conversion

The job is read into the system in its entirety by the System Input program and any requested character transliteration is performed. Local batch jobs come from the online card reader or from magnetic tape at the central site, while remote batch and terminal/batch jobs enter the system via the Front-End Network Processor (FNP). The System Input program validates the control cards containing

- Job defining information
- Accounting information
- Activity defining information.

If the job fails these tests, diagnostic information is provided, the job is deleted, and the user must correct his JCL errors and resubmit the job.

If the job passes the tests, the job is submitted to the Scheduler.

Job Scheduling

The Scheduler accepts a job from the input media conversion phase and places it in one of several queues. The particular queue into which a job is placed depends upon the resource requirements of the job, and on the specific scheduling queues that have been defined by the installation. Depending upon the total load on the system, a user's job can remain under the control of the Scheduler for a period ranging from seconds to hours. Eventually the job reaches a position in a queue which indicates it is a candidate for allocation.

Since the number and types of scheduler queues can vary from site to site, we cannot give details here. In general, however, the fewer and smaller the resource requirements for a job, the shorter will be the period of time that the job remains under the control of the system scheduler. (The user who wants to get into the fastest queues should avoid magnetic tape or private disk usage altogether, and should request as little memory and processor time as possible.)

Activity Allocation

When a job initially enters the Allocator, a detailed JCL analysis is performed for the total job. If control card errors are detected, the entire job is deleted.

Some common errors detected during this examination include:

- o An activity request for resources not available in the system configuration. (For example, an activity needs 16 tapes and only 12 are configured, or an activity has requested 150K words of memory and only 128K words are available on the system.)
- o Missing or incorrect control card parameters.

If the job passes the control card scan, it is then a valid candidate for peripheral and memory allocation. From this point on, each separate activity of the job flows from the Allocator to the execution phase and finally to the termination phase. This cycle continues until all of the activities of the job have been processed.

Each activity of a job is processed in the sequence submitted. For the current activity of the job, the Allocator:

1. Determines the eligibility of the activity for execution. Normally, because of the job/activity relationship, an activity is eligible for execution based on the results of a preceding activity. If a preceding activity did not complete successfully, all subsequent activities of the job except compilations will be deleted, unless the JCL specifically requests otherwise ($ BREAK usage).
2. Reserves specific peripheral devices (if any) for all defined files.
3. Informs the operator to mount requested volume on specific peripheral devices (tape or disk-pack files).
4. Reserves sufficient memory space, based upon JCL parameters (default or explicit).

During allocation, time delays are often experienced because the peripheral devices and/or the needed memory are not immediately available. If an activity proves difficult to allocate, eventually a point is reached where

1. Activities of other user jobs are not considered for peripheral allocation until the peripherals for the "blocked" activity can be allocated, and/or
2. Activities of other user jobs currently in execution may be temporarily removed from memory (swapped out) to make memory available for this activity.

When all requested peripherals have been allocated, all necessary files mounted, and sufficient memory has been reserved, the activity is ready for execution. Based upon control-card information, the allocator has technically tailored a hardware configuration to meet the specific needs of that activity. Thus a portion of the total hardware complement has been reserved for the activity. Control is then passed to the execution phase.

The allocation phase actually consists of two subphases: peripheral allocation, and memory allocation.

The type of activity requested by the user (a compilation, an object-deck load and execute, or a standard system program, e.g., FILSYS) is examined and the appropriate program is loaded into the memory reserved. When processor control is dispatched to this program, it begins execution. All programs loaded into memory are represented in a queue controlled by the Dispatcher. (Remember that it is activities of different jobs that are multiprogrammed, not multiple activities of the same job.)

The Dispatcher manages its eligible-program queue in such a way as to keep as much of the allocated system resources in effective, concurrent use as possible, essentially by overlapping processor and peripheral-subsystem usage as much as possible. During its execution phase, a program can lose and regain control of the processor many times. The executing program can be interrupted (i.e., lose execution control) for a variety of reasons, e.g., termination of an I/O operation for another, inactive program, or expiration of its current processor-time quantum (time slice). Alternatively, it can voluntarily relinquish control, usually to await the completion of a requested I/O operation. All activities concurrently in the execution phase of job flow, therefore, "take turns" using the processor and the I/O subsystems.

An activity will remain in the execution phase until any of the following conditions occur:

1. The program executes a sequence of instructions indicating that the activity is complete, e.g., a sequence compiled for a STOP RUN statement in COBOL or a STOP statement in FORTRAN. This condition is termed normal termination.

2. The user program violates a usage rule. In this case a termination routine removes the activity from execution. This condition is termed abnormal termination. Typical violations include:

   a. The user program has requested an I/O operation on a file which has not been defined by its associated JCL; therefore the file has not been allocated to the program and is not available.

   b. The user program has attempted to access a memory location beyond its allocated memory boundaries.

   c. The user program has attempted to access a mass storage file beyond the size defined.

   d. The user program has exceeded its maximum allocated processor time or print-line limits, either for the activity or for the job as a whole.

3. The Dispatcher removes the program (temporarily) from memory; that is, the program is "swapped" to disk to make memory space available for a higher urgency job. When the swapped program is read back into the memory, it reenters the execution phase.

During the execution of the activity, system output (SYSOUT) for printing and punching is collected on a mass-storage file for deferred printing and punching. Typical examples of system output include

- Source listings produced by a compiler
- Object-program decks produced by a compiler
- Load maps produced by the Loader
- Accounting reports produced by the operating system.

In addition, a user can direct any print/punch file separately defined within his program to a system-output file via either a $ SYSOUT or $ REMOTE control card. All information directed to SYSOUT is temporarily stored on disk.

Activity Termination (Termination-Module Group)

The execution phase passes control to the termination phase for both normal and abnormal termination conditions. This phase, handled by a group of GCOS modules known as the Termination Group, performs the following clean-up functions:

1. Collects accounting information for the activity on its SYSOUT file.
2. Writes a memory dump, if appropriate, on the SYSOUT file. The user may request a memory dump for any activity that terminates abnormally.
3. Looks ahead to the next activity (if any) in the job, and proceeds as follows:

   a. If the next activity is the same type as the terminating activity (for example, back-to-back COBOL or FORTRAN compilations), GCOS recognizes that the same system resources are required for the next activity of the job. Therefore, a return to the allocation phase can be bypassed since the required resources are already reserved. Control is returned to the execution phase for the next activity of the job.

   b. If the next activity is a different type (for example, a COBOL compilation followed by an execution), the system examines the file definitions of the terminating activity. The user can direct, via his JCL, that his files be:

      (1) Dismounted or de-accessed (saved for reuse by some subsequent job),

      (2) Saved for a subsequent activity of the job, or

      (3) Released back to the system (to become available to other users).

      The system informs the operator which files are to be dismounted, releases the designated files and/or devices, releases the allocated memory, and returns control to the Allocator for the next activity of the job.

   c. If there are no following activities (i.e., end of job), the Termination Group performs the same functions as described for subphase 3.b except that control passes to the output media-conversion phase.

Output Media Conversion (SYSOUT)

This is the final phase of job flow. At this point, all of the jobs's activities have been processed and system output for each activity has been stored on the SYSOUT mass-storage device. (Thus, output for the entire job is now ready.) The user has directed GCOS to either print or punch this output locally, or has specified that the output is to be transmitted to a remote station and should be held until the remote station requests such transmission. (Requests for remote transmission require that the remote user call back into the central system and identify his job number and remote station code).

The output phase for central-site output reads the output images stored on the SYSOUT device and disperses them to either the printer or to the card punch. Control information has been automatically appended to these card/print images which permit the system to recognize the peripheral media (card punch or printer) and the type of report. Printed system output for an entire job is generally produced in the following order:

1. A banner page (identifies the output for bursting and distribution purposes)
2. A complete listing of all control cards comprising the job
3. Accounting reports for each activity of the job
4. Output directed to SYSOUT (for printing) for each activity of the job
5. End-of-output banner page.

Note that the output phase is printing/punching information for all completed user jobs. Therefore, the printing/punching of your job may be delayed depending on the total amount of information from all jobs waiting to be printed/punched. (That is, you must wait your turn for system printer/punch output). If the job is submitted via time sharing, however, the user may monitor the progress and termination of each activity, and at end-of-job may scan (selectively inspect) his output at the terminal, prior to or instead of having it printed.

TIME SHARING JOB MANAGEMENT

The GCOS Time Sharing System (TSS) includes a central software component called the Time Sharing Executive (TSE). The TSE is a system program that is invoked (and released) by the operator, i.e., when a scheduled period of time sharing service is due to commence. The TSS Executive is essentially a highly-specialized (and dynamically optional) "mini" operating system that manages time sharing jobs, i.e., user's terminal sessions. (The TSS job-managmenent functions are kept physically separate from the basic GCOS software so the system-overhead costs associated with them are not incurred during periods when time sharing service is not provided, or are not incurred at all by batch-only installations).

When active, the TSE calls upon many of the same GCOS service functions that are used for batch-job management (e.g., temporary-file space and peripheral-device allocation modules), so that software structures are not duplicated for nonunique functions. The primary functions performed by the TSE are the following:

1. Initiation and termination of a user's terminal session.

2. Management of the TSS/user dialog; i.e., handling of all messages between any TSS subsystem and the user.

3. Recognition of requests for TSS subsystems (system-level TSS commands) and activation of the required subsystems. (This is analogous to the batch activity-allocation phase.)

4. Management of TSS memory space, and the dispatching of execution time among concurrently active subsystems. (This is analogous to the batch activity-execution phase.)

5. Management of transitions between one TSS subsystem and another for a given user, i.e., subsystem terminations or interruptions. (This is analogous to the batch activity-termination phase.)

Of the TSS-specific functions listed above, item (4) may be singled out for discussion as exemplifying a time sharing specific job-management function. The management of TSS memory space, i.e., that portion of memory reserved for allocation to TSS subsystems and administered by the TSE, is characterized by a great deal of "swapping" of active subsystems (relative to batch programs). The swapping process itself is the same for both batch and time sharing: it consists of moving an active but interrupted program and its 'context' from memory to a mass-storage device, and moving an already swapped-out program back into memory. The much higher frequency of TSS subsystem swapping (assuming a reasonably high time sharing load) is due to the inherent nature of interactive I/O: it consumes a greater amount of time than batch I/O by several orders of magnitude. This is so not only because the output speed of remote-terminal devices is very slow (relatively speaking), but because the reaction time of the terminal user (his input speed) is slower yet by far. Thus, during the period of time required for one interchange of messages between a TSS subsystem and a user, several other subsystems can have executed many thousands of instructions and initiated terminal messages of their own. Therefore, the general rule is that a TSS subsystem will be automatically swapped out upon initiation of any terminal I/O. (Again, assuming contention for CPU time among active subsystems.)

The combination of a high swapping rate and the generally smaller size of TSS-subsystem programs (relative to batch) dictates a different memory-allocation and time-dispatching strategy for time sharing operations. Therefore, the TSE incorporates its own optimal TSS-memory allocation and dispatching functions. (Several other comparable, but less instructive, time sharing vs. batch differences could also be shown.)

The preceding descriptions of job-management software structures and functions are highly simplified and presented from a dynamic, functional viewpoint. Therefore, from a strictly technical view certain fine points have been blurred. (For example, batch program loading is actually initiated by an Allocator module at the end of the activity-allocation phase, instead of in the activity-execution phase, but effectively it is part of the latter phase.) The major intent here has been to give the reader an understanding of the major functions of the GCOS job-management structures in as uncomplicated a fashion as possible.

File Management Services

The GCOS file-management services, and the corresponding software components, can be grouped into three categories:

- File space management
- File access-control/file protection services
- File access methods.

The first two service categories, space management and access-control/protection, are handled by the File Management Supervisor (FMS), which consists of the system program FILSYS and many associated GCOS modules.

The FILSYS program is invoked explicitly by the user (via $ FILSYS or the TSS ACCESS subsystem) when he desires to create or modify permanent files and file catalogs within the hierarchically structured GCOS File System. The symbolically-named file (or user's subcatalog under which a number of such files can be grouped) thus created can be given many attributes, or characteristics, among which are the following:

- Initial and maximum size (files only)
- Random or sequential file mode
- Password
- Access permissions: READ, WRITE, QUERY, EXECUTE, PURGE, MODIFY, etc,; either to specifically named users or to all other users. (The MODIFY permission allows another user to modify the cataloged attributes of a file.)
- Concurrent access allocation and control
- Incomplete update protection
- File duplication
- File journalizing
- Storage-device type
- I-D-S (integrated) file structure

The modification of files or catalogs through FILSYS mentioned above pertains to the modification of the attributes described, not of file content.

Various portions of FMS, including FILSYS, are implicitly invoked when the user requests access to a permanent file, e.g., via a $ PRMFL statement or a (TSS) OLD command. The functions performed in this case involve verification of:

- The file's existence
- The user's right to access the file (password and permissions checks)
- The file's current availability, i.e., whether it is 'busy' or not, whether concurrent access is allowed, etc.

These functions are performed during activity allocation.

When the file is actually in use, that is, during execution of an activity that affects the file, other portions of FMS are automatically invoked (by IOS) to perform any special concurrent access control or file-content protection procedures that were requested for the file upon its creation, e.g., the duplication or journalizing attributes. The file access-method software (for other than specially structured data bases) consists of three separate GCOS subsystems: the Unified File Access System (UFAS), the File and Record Control Facility, and the Indexed Sequential Processor (ISP.) Each of these subsystems provide software routines which allow user and system programs to perform I/O operations in logical rather than physical terms. (The GET or PUT next-logical-record level of operation, for example, as opposed to reading from or writing to a specific physical space on an I/O device). These routines effectively translate the symbolic I/O request into the machine-level code sequences that are recognized and executed by IOS, the Input/Output Supervisor.

The major distinctions between these three collections of access-method software for the higher-level language user are as follows:

1. Both UFAS and the File and Record Control Facility are explicitly invoked only in assembly-language (GMAP) programs, thus they are "invisible" to the user of a higher level language. (The language compiler causes the appropriate access-method routines to be included in the object program when the user requests file-level I/O in his source program.)

2. The Indexed Sequential Processor (ISP) routines can be invoked at any source-language level (e.g., in COBOL or FORTRAN), allowing the creation and very efficient manipulation of indexed-sequential files, an optimal data structure for many types of applications.

3. The File and Record Control Facility supports sequential and random-access file organizations, and standard Honeywell magnetic-file labeling. Thus it provides full file compatibility between the Honeywell Series 6000 and Series 60 Level 66 systems, for established Honeywell large-scale computer users. It also provides a Series 2000-to-Series 60 Level 66 magnetic tape interchange capability.

   The File and Record Control access methods are utilized by all higher-level language processors except the COBOL-74 compiler and the COBOL-68 compiler when used in full ANS mode. (These access methods are utilized by the COBOL-68 compiler operating in Commercial-Subset mode.)

4. The newer UFAS access methods provide an extensive range of capabilities, including as a subset the functionality of the File and Record Control software. A brief summary of UFAS capabilities is as follows:

- o Sequential, indexed, relative, and integrated file organizations.
- o Sequential, random, and dynamic access modes.
- o Unified File Format for all mass-storage files.
- o ASCII, EBCDIC, and BCD character-set processing, with either automatic or user-specified transliteration as required.
- o ANS and IBM magnetic-tape file formats.
- o Honeywell Series 6000 (File and Record Control) file-format compatibility for both magnetic tape and sequential mass-storage files.
- o Label processing for ANS, IBM, and Honeywell Series 6000 magnetic tape files.
- o Error checking and error processing according to the American National Standard for COBOL-74.
- o Honeywell Series 2000 magnetic-tape file interchange.
- o ISP 'file coexistence' feature.

The UFAS access methods are utilized by the COBOL-74 compiler and the I-D-S/II processor.

(Other highly specialized access-method software oriented specifically to data base management is discussed under the next heading.)

Data Base Management Services

A data base, for the purposes of this discussion, is defined as a mass-storage file -- generally quite large -- containing records, fields, and items of information which have a defined hierarchical or network type relationship to one another. In other words, a data base is a file that is physically structured in such a way as to reflect the logical relationships of its contents, i.e., of the various units of data stored within it. The intent behind such a highly-structured file organization is several-fold:

- o To provide an integrated, uniform collection of information that, to its end users, models the information structure of the business they are engaged in; as opposed to a set of separate data files that are ordered according to the convenience of the computer system and more traditional types of software supporting it.
- o To allow for the updating of many interrelated items of information in a single file, rather than by a sequence of processes in which each process updates its own master file(s) to reflect changes to files treated by a preceding process. (An example of such a traditional application processing sequence might be a product inventory run, followed by a production scheduling run, followed by a material-stockpoints run, followed by an accounts-receivable and accounts-payable run.)

- To allow information update and retrieval from geographically distributed locations, as and when required. (For example, a common data base might effectively link a factory, several warehouses, many sales offices, and corporate headquarters.)

- Finally, to provide a quick-reaction capability for the unplanned, unanticipated information processing and reporting requirements that have become increasingly common in today's dynamic business environment.

Obviously, a wide variety of very sophisticated software tools are required to aid the user in the structuring and initial creation of such data bases, and in the development of application packages (both batch and interactive) for manipulation of the data base.

The basic data base management facilities (i.e., those which underlie the Interactive Subexecutive end-user facilities described in Section V) can be divided into several categories:

1. Translators for data definition and application definition languages (DDLs and ADLs): these languages allow the user, normally a data base administrator, to structure the data base (schema definition) and to structure several or many application-specific, logical subsets of the data base (subschema definitions). The output of DDL and ADL translations are utilized by the following categories of software.

2. Data Base content managers: These are collections of logical I/O routines, analogous to the access methods described in the preceding subsections, which translate the data base manipulation requests (CREATE, RETRIEVE, UPDATE, PRINT, QUERY, etc.) of the user's data manipulation language. The source-language processor may be the I-D-S/I Translator (an extension of COBOL-68), COBOL-74 for I-D-S/II, or MDQS.

3. Specialized utility processors for initial building or logical restructuring of a data base.

4. Highly simplified time sharing interfaces, primarily for "browsing" of the data base by end users.

The software components that provide some of all of the facilities described above are:

- Integrated Data Store/I (I-D-S/I)

- Integrated Data Store/II (I-D-S/II)

- Management Data Query System (MDQS)

- I-D-S Data Query (A TSS subsystem for querying an I-D-S/I data base a. a remote terminal)

- dataBASIC (A BASIC-like TSS subsystem, primarily intended for the building and operation of private, individually maintained data bases).

I-D-S/I and I-D-S/II each incorporate a data base content manager for handling its own type of integrated (or chained) data base organization. MDQS, however, utilizes either the I-D-S/I content manager software or one of the access methods described in the preceding subsection, as appropriate to the organization of the particular data base it is processing (i.e., integrated, indexed sequential, or sequential). Although the MDQS 'package' includes a number of batch elements (e.g., the DDL and ADL translators and several utility programs), MDQS itself is primarily an interactive subexecutive supporting an end-user oriented procedural language for interactive data base manipulation. Therefore, it will be described further in that aspect in Section V.

Data Query is a TSS subsystem that permits online retrieval of information from an existing I-D-S/I data base in a simple and convenient fashion.

The TSS dataBASIC subsystem is totally self-contained, and manipulates only those data bases built by it or by its utility program. It operates on a very efficient multiply-indexed (inverted) file structure.

A significant distinction between I-D-S/I and I-D-S/II (or MDQS) is that in the former, the data base structuring facilities are not separate from the data-manipulation language; i.e., the DDL function is not a separate language but is incorporated into the COBOL-like host language.

MDQS provides DDL and ADL capability for each of the data base organizations that it handles, and initial building of the data base is source-language independent. While I-D-S/I has a time sharing interface for easy report generation and online "browsing," both I-D-S/I and I-D-S/II are basically batch oriented (though full-capability processing runs can always be initiated through time sharing). The newer I-D-S/II software couples the functionality inherent in I-D-S/I with the state-of-the-art features contained in the CODASYL 1974 proposals. Key among these features are:

- The data-manipulation language (DML) is an integral part of COBOL-74.
- The fully implemented schema and subschema description capability provides a high degree of independence between data structure and DML programs, and includes data dictionary functions.
- A set of advanced mechanisms for privacy and security, data integrity assurance, recovery/restart, and concurrent access control provide maximum dynamic protection of the data base.
- A very large data base capability: Up to 68 billion records can be accommodated.

The facilities provided specifically for interactive information processing are discussed further under "Interactive Subexecutives" in Section V.

Communications Management Services

The communications management services can be conceptually divided into two functional areas:

1. Local Physical I/O
2. Network Communications.

The latter area can be further divided into two categories:

a. Remote device handling

b. Message switching.

The first category (a) pertains to the transmission, reception, and routing of messages between the central computer (or "host") and a terminal device or satellite processor on its network.

The second category (b) pertains to the routing of messages between individual terminals on the network.

LOCAL PHYSICAL I/O

Local physical I/O services handle the physical input/output operations between the processor(s) and the local I/O devices: disk units, magnetic tape units, card readers, punches, line printers, and system-operator's console(s), and any other I/O devices that might be configured at the central site. These services are provided by two GCOS subsystems: the Input/Output Supervisor (IOS) and the Exception Processing Subsystem. The former handles the initiation, control, and termination of all successful I/O operations (through the hardware intermediaries of input/output multiplexers and peripheral-subsystem controllers). The Exception Processing Subsystem provides all exception processing for I/O operations that terminate unsuccessfully. Exception processing consists primarily of error analysis, I/O retry in the case of possibly recoverable errors, and related communications with other portions of the system. (Notably, communication with the operator and with statistics-gathering routines.)

Since the normal user has no direct interfaces with this level of software, it is not discussed further here.

NETWORK COMMUNICATIONS

Network communications is performed by either of two major software subsystems: the General Remote Terminal Supervisor (GRTS) or the Network Processing Supervisor (NPS). The primary distinction between these two subsystems is that while each supports communications between the central processor and remote devices (e.g., TSS terminals) on the network, NPS also provides:

- Message Switching -- Essentially a facility that allows nodes of the network other than the host to intercommunicate (e.g., terminal-to-terminal), without routing the message through the host's central processor.

- Generalized Terminal-Handling Capability -- A facility that allows individual sites to add to the range of terminals and line disciplines that are acceptable to NPS. This is done by adding a new "routine" (composed of NPS Micro-operations, or MOPs) to the open-ended NPS line handling structure. This process does not involve any reprogramming of NPS as such, since the NPS software is expressly designed to be expandable in regard to the types of terminals that can be supported. (This extension of NPS capabilities would ordinarily be performed by the site's technical support personnel.)

Either of these subsystems, GRTS or NPS, is utilized by any of the following types of software:

- o TSS, for routing of messages between the TSE or a TSS subsystem and a time sharing terminal.
- o An interactive subexecutive, e.g., MDQS, TDS, or TPS, also for message routing.
- o A user program that is programmed for, and privileged to perform, direct program access (DAC) communication between itself and a remote-terminal device. Such programs can be written in several of the higher-level languages provided by the Level 66 Information System, e.g., COBOL-68 or COBOL-74.

The only direct system interface that the normal user has with this level of software is that in the last-mentioned case, i.e., a direct-access user program, the $ DAC statement must be included in the JCL for the program. This statement performs two functions:

- o It defines a specific program file (by file code) as being a remote terminal device, and
- o Provides a unique identifier by means of which the terminal operator can request connection with the program.

All other interfaces with the message-handling software (GRTS or NPS) are provided automatically, through compilation of the user's source-language program.

System Management Services

The system management services comprise the software tools and aids provided for the site manager and his operations and technical-support staff. The purposes of these tools and aids fall into the following categories, listed in their nominal order of use:

- o System-software definition and installation
- o System monitoring and performance evaluation
- o Dynamic system-administration interfaces
- o Online hardware test facilities
- o System-software maintenance facilities

Since this overview document is intended primarily for the software user, only a cursory description of the system-management software subsystems/elements and their functions will be given.

SYSTEM DEFINITION AND INSTALLATION

The primary software tools for defining a software system, i.e., tailoring it to a site's specific needs and "installing" it (getting it into an operable form), are the Startup program and the Patch Editor (PAED) program. The basic strategy of this process for the Honeywell Level 66 differs significantly from that for some competitive systems. The process for certain other comparable systems is referred to as system generation (SYSGEN), a long and complex procedure involving the selective and conditional assembly and program binding, or linking, of all desired system software from its source-language form.

The software for the Honeywell Level 66 Information System, however, is supplied in a system-loadable format, i.e., already assembled and linked, in the form of a "total system tape." (Several reels of magnetic tape are actually involved.) The total system tape includes all standard system software except that which is separately priced. The Startup program loads from the total system tape an operable but not-yet-tailored system, which is then used to initialize necessary system devices, files, catalogs, and libraries, and to run the total system tape against a file of patches for "known errors" in the particular release (or version) of the system being installed, via the Patch Editor.

From this point on, the definition/installation process consists of "pruning" the total system tape, i.e., selecting desired software, and inserting, deleting, or replacing installation-dependent modules as circumstances require. This latter procedure is directed by a "startup deck" which is processed by the Startup program, effectively reloading the system in a tailored form. The aim of this process is to achieve an image of the desired system on the system-disk device, from which it can be reloaded for subsequent operation.

The Startup program is divided into a number of functionally separate sections, any combination of which can be utilized to achieve desired modifications of the system, such as redefining the hardware configuration, adding an installation-written module (e.g., an accounting routine), or changing site-determined operational parameters via patch cards. Normal "bootloading" of the system is also achieved through the loading function of the Startup program.

SYSTEM MONITORING AND PERFORMANCE EVALUATION

The system monitoring and performance-evaluation facilities comprise the following software subsystems or elements:

- The VIDEO (Visual Information Display for Efficient Operation) program, which displays system-operating statistics on a large-scree video display. This allows operations personnel and interested user to monitor the system's performance on a minute-by-minute basis.

- HEALS, the Honeywell Error Analysis and Logging Subsystem, which operates "invisibly" alongside other software and user programs to intercept and track hardware-detected error conditions; attempt to retry instructions for recovery from transient processor or memory faults; and issue warnings to the operator whenever the error rate of any hardware component approaches an abnormal point. The intent of HEALS is twofold: (1) to enhance system availability and integrity, and (2) to ensure timely online testing and/or maintenance of marginal system components.

- The Statistical Collection File (SCF) and the Summary Edit Program (GSEP). The SCF is an online repository of operating-statistic records generated by all major operating-system components. GSEP periodically compiles and prints a statistical report summarizing the information contained on the current SCF. The GSEP reports can be used to analyze and evalute the effectiveness of past and current system-operation parameters, and software/hardware configurations. In effect, the reports constitute a detailed history of system performance. The installation can add to or modify the SCF records by means of a site-supplied module provided for in the system.

DYNAMIC SYSTEM-ADMINISTRATION INTERFACES

The system-administration interfaces that allow dynamic control of system operation are twofold:

- The console-operator command interface
- The TSS Master-User interface

The Operator Command Interface

The system operator can exercise a considerable degree of control over system operation by use of various console commands. During system operation, he can:

- Initiate and terminate the Time Sharing System
- Increase or reduce the amount of memory available for time sharing allocation
- Initiate and terminate the Transaction Processing System
- Initiate and terminate MDQS (for interactive operations)
- Change the system "sieve" criteria, i.e., the resource-requirement values that, if exceeded, will cause a job to be held for deferred processing.
- Take marginal or defective system components, e.g., memory modules or peripheral devices, offline for servicing or maintenance.
- Logically switch peripheral devices to provide a "backup" for a device that was placed offline.
- Modify certain Scheduler parameters so as to 'tune' system performance for nontypical system loads.
- Terminate specific jobs, because of abnormal or adverse conditions affecting system throughput or the job in question.
- Disconnect and reconnect specific communication lines.
- Reinitiate jobs in 'hold' or 'sieve' status.

While the above list is not exhaustive, it does represent most typical system control procedures.

The TSS Master-User Interface

The Time Sharing System provides for an authorized TSS master user who can monitor and control time sharing operations, and can modify the GCOS File System with respect to authorized users, their privileges and resources. The interface for these capabilities is provided by the TSS MASTER subsystem. Legitimate access to this subsystem is essentially protected by a multiple-password scheme.

During time sharing operation, the authorized MASTER user can:

- Obtain detailed status reports on current TSS activity
- Monitor any other user's terminal session
- Issue "all points bulletins," or messages to specific users
- Take "snapshots" of, and apply temporary patches to, TSS software
- Alter certain TSS operating parameters
- Communicate with the console operator
- Modify entries in the system-master-catalog (SMC) of the GCOS File System, e.g., to add or delete users, and to change passwords, permissions, and resource-usage limits. (The SMC effectively controls all authorized file-system and time sharing users.)

In summary, the MASTER subsystem facilities, in conjunction with VIDEO and the system operator's capabilities, offer the system administrator a full range of control functions either on-site or at a remote location.

ONLINE HARDWARE TEST FACILITIES

The Total Online Testing System (TOLTS) operates under GCOS supervision -- concurrently with user programs if desired -- and provides a very extensive range of diagnostics that can be controlled from either the system console or a remote terminal. TOLTS is composed of the following modular subsystems:

- POLTS - The Peripheral Online Test Subsystem, which can be used in a special "slack" mode so that peripherals can be checked for maintenance requirements during normally scheduled production time.
- MOLTS - The Mainframe Online Test Subsystem. Individual main-storage modules can be allocated to MOLTS so as to test memory and memory/processor interaction with minimal interference with user operations.
- COLTS - The Communications Online Test Subsystem. Tests Front-End Network Processor (FNP) operations in an online environment (i.e., coexistent with either the GRTS or NPS subsystems).
- SOLTS - The System Online Test Subsystem, a combination of POLTS, MOLTS, and COLTS functionality, but with special emphasis on processor integrity and the "business" instruction set.

Note that TOLTS complements the full array of offline Test and Diagnostic tools available to field engineering personnel for regularly scheduled system maintenance.

SYSTEM SOFTWARE MAINTENANCE FACILITIES

"Software maintenance" means primarily the incorporation of software changes available from Honeywell between releases of new versions of the system.

For this purpose, the facilities already described for initial system definition and installation (Startup and the Patch Editor), are utilized, plus:

- The Source and Object Library Editor (FILEDIT)
- The System Library Editor (SYSEDIT)

FILEDIT and SYSEDIT are used, in conjunction, to update either the total system tape or an existing system library file. These two facilities permit incorporation of program modifications (generated by Honeywell) that cannot be handled by patches applied via Startup or the Patch Editor. They also permit inclusion of additional program modules to satisfy a site's need for increased processing capabilities (e.g., addition of transaction processing software).

The process of system editing is dealt with in detail in the System Management Guide: Software Installation and Maintenance document (Order No. DE62). It may be noted here, however, that the system-editing process avoids the total reassembly and regeneration of the system (i.e., the SYSGEN process) that is characteristic of some competitive systems for purposes of system maintenance.

The functions of FILEDIT and SYSEDIT as they relate to the user are described in Section IV, under "Utilities". (These user-related functions include the building and maintenance of a user's source and object program library, and of a site's production-program library.)

SECTION IV

LANGUAGE PROCESSORS AND UTILITIES

This section describes the salient features of the programming languages and utilities available for the Honeywell Level 66 Information System. It also describes any significant characteristics of their respective processors.

LANGUAGES

The Level 66 programming languages are covered below in alphabetical order.

ALGOL

The Level 66 ALGOL language conforms fully to the ALGOL 60 international standard, but also provides improvements and extensions to that standard, including enhanced I/O.

The ALGOL language is a well defined, algorithmic language primarily intended for solution of problems involving numerical analysis. It has a hierarchical block structure which facilitates true structured programming. (It is more commonly used by the scientific community than by business users.)

ALGOL programs can be developed and executed in time sharing as well as local and remote batch environments.

The language extensions include:

- EXTENDED REAL type attribute for extended-precision real number operations
- An extended-integer division operator
- Input/output implemented according to the Knuth proposal published in the Communications of the ACM, May, 1964.

The improvements and special features of the Level 66 ALGOL implementation include:

- A flexible set of input/output functions that provide formatting, logical and physical record processing, and character handling capabilities

- o I/O procedure statements for logical record handling between memory and external devices: statements are provided for transferring data in physical record blocks between memory and external devices, and for logical record character processing
- o STACK tracing routines
- o Machine level double-precision, floating-point word implementation of EXTENDED REAL type
- o DEBUG option.

BASIC

Level 66 BASIC is an interactive language designed and developed expressly for the time sharing user. It is quite easy to learn and to use, and is especially useful for (but not limited to) solutions of day-to-day technical problems by users who are not professional programmers. This version of the original Dartmouth-designed BASIC language contains many extensions and improvements which significantly extend its power and range of applicability.

The BASIC processor (a TSS subsystem) is a fast, fully conversational compiler. The many extensions and improvements incorporated in Level 66 BASIC include:

- o Both ASCII and binary file I/O
- o Extensive character-string handling capabilities
- o The ability to save and re-execute object programs
- o Subroutine and function statements.

COBOL-68

The Level 66 Standard COBOL-68 language represents a full (maximum level) implementation of the American National Standard for COBOL-68 (X3.23-1968). The language also includes extensions for capabilities beyond the requirements of this standard, and special features that relate to other Level 66 software. (These are briefly described below.)

The Standard COBOL-68 compiler offers a number of features that are outstanding by any industry criterion. The fully annotated source listings provide clear, plain-language warning messages and error diagnostics. I addition, suggestions for potential source-code optimization are produce wherever possible. Detailed cross-reference information for data and procedure names is provided.

The compiler operates in either of two modes, normal ANS or Commercial Subset, at the user's option. The purpose of the Commercial Subset mode is to allow compilation of certain older COBOL programs which would be invalid if compiled according to the standard, either with respect to syntax or to desired compilation results. Compilation in ANS mode supports not only the standard language and extensions thereto, but also certain other language elements defined in earlier CODASYL specifications that are not contrary to the current COBOL-68 standards (ANS or CODASYL). In the normal compilation mode, the compiler utilizes its own logical I/O routines, which are expressly designed for ANS magnetic-tape file format and labeling compatibility.

Salient features of the implementation include the following:

- Full ANS implementation of the SORT verb.
- The MERGE verb implemented in accordance with the CODASYL Journal of Development, 1970 specifications.
- Full ANS implementation of the Report Writer facility.
- Single and double-precision floating point, decimal precision, and single and double-precision binary integer data representations, with packed-decimal format provided on all Level 66 models.

Extended language capabilities and other special features include the following:

- COPY - ANS COBOL-68 provides for COPY only from a library file. Level 66 COBOL-68 allows file renaming and COPY of portions of the same source program containing the COPY, as well as COPY from the library file. Both methods are allowed at the user's option.
- DEBUG - The PROCESS statement specified under the SPECIAL-NAMES paragraph provides a source language debug feature. This enables the programmer to place debug statements in the COBOL source program and, at compile time, to indicate the debug levels to be included in the generated program.
- Level 66 COBOL-68 provides user interfaces with the GCOS Transaction Processing System through the ACCEPT and DISPLAY statements. Direct terminal communication is also provided.
- The programmer can use the powerful syntax checking features of the compiler and optionally bypass the code generation phase of the compilation process, dependent upon the incidence of fatal errors.
- Level 66 COBOL-68 provides excellent facilities for the processing of common data between independently compiled subprograms. The assignment of all files to Labeled Common storage, plus the ability to assign any or all items in WORKING-STORAGE to Labeled Common permits flexibility for referencing common data with COBOL or other language subprograms. Consequently, subprograms can be compiled separately and then combined to perform as one operational program.

The COBOL-68 compiler is operable on all Level 66 and Series 6000 models, thereby ensuring maximum compatibility and program interchangeability for established Honeywell large-scale sites.

The Level 66 COBOL-74 compiler and its runtime package is a complete implementation of the maximum level of the American National Standard X3.23-1974 for the COBOL-74 language. It also implements extensions beyond the capabilities specified in the standard.

The COBOL-74 language has a number of new and powerful features:

- An integrated Communications Facility permitting communication between program and terminals
- An improved Report Writer
- Interprogram communication, featuring a LINKAGE SECTION and CALL/CANCEL statements
- A powerful Debug capability with compile/object time switches
- Expanded input/output capabilities: sequential, relative, and indexed file organizations with sequential, random, or dynamic access modes
- MERGE statement
- Powerful new verbs - STRING, UNSTRING, and INSPECT
- Expanded arithmetic and ACCEPT statements
- SIGN clause
- Program COLLATING SEQUENCE clause

The Level 66 COBOL-74 compiler and runtime package are highlighted by the following:

- ASCII native mode
- Byte orientation
- Supports the data manipulation language (DML) for I-D-S/II data bases (an extension)
- Utilizes the new Unified File Access System (UFAS) logical I/O routines
- File coexistence/conversion capability for COBOL-68 sequential files, ISP files, and IBM files with mixed data types (an extension)
- Half-byte/byte aligned packed decimal; 16, 32, and 36 bit binary integer
- A CONTROL DIVISION, which provides for modifying program source and altering compiler defaults (an extension)
- Optimized object code -
    - New PERFORM mechanism
    - Improved subscript calculation
    - Compile-time calculation of constant values

- Extensive in-line code generation
- Extensive register and common subexpression management
- Direct generation of object code

- Symbolic literals permitting entry of non-printing characters (an extension)
- Transaction Processing System interface (an extension)
- Block Common (an extension)
- Honeywell Series 2000 magnetic tape support (an extension)

FORTRAN

FORTRAN is a very widely used language for mathematical and scientific problem solving, and has many minor variants. The Level 66 FORTRAN language is essentially an extended version of ANS FORTRAN IV. It includes most of the significant extensions developed by other manufacturers and a number of features unique to itself. (ANS FORTRAN IV is a proper subset of the language.) Thus, the Level 66 FORTRAN implementation is among the most comprehensive available in the industry.

The Level 66 FORTRAN processor is composed of three integrated software entities: the compiler; the library of run-time modules that support execution of FORTRAN object programs; and a specialized time sharing interface. The compiler is directly executable in time sharing mode, as well as in the local or remote batch environments. Therefore, compatibility between source programs developed in one environment and used in another is ensured since one compiler is doing the job for all environments. A collection of source programs can be compiled - some through time sharing, some through batch - and the object modules combined for execution in either environment.

Advantages include:

- A common library
- Files in standard format
- Free-form format, with or without line numbers
- Multiple compilations within an activity (provided the options are the same for the collection of subprograms).

Features include:

- Memory-to-memory conversion (ENCODE/DECODE)
- List-directed formatted I/O
- Random file I/O
- Mixed-mode arithmetic
- Subscripts can be any expression

- DATA initialization in any type statements
- END= clause in READ statements
- ENTRY, CHARACTER, PARAMETER and IMPLICIT statements
- T and R format specifiers
- ABNORMAL statement
- PAUSE and STOP with printout
- Quoted character constants
- ERR= clause in READ and WRITE statements
- Switch variables
- Type statements with size-in-bytes notation
- FLD function - a built-in function that provides bit string and field capabilities
- XOR function - complements the Boolean functions with an "exclusive OR" capability
- Argument validation for built-in functions
- Null label fields in the arithmetic IF statement.

JOVIAL

The Level 66 JOVIAL compiler is equivalent to the J3 version JOVIAL compiler of the U.S. Air Force, as described in AFM 100-24. The JOVIAL language allows the programmer to make use of specific characteristics of Level 66 hardware, while imparting all the advantages of a higher level language. It is intended for application areas that require end-user 'command and control' facilities as well as numerical computation.

JOVIAL programs can be developed and executed in time sharing as well as local or remote batch environments. Programs developed in one dimension can be executed in any other. Various data formats are provided, including integer, floating-point, fixed-point, Boolean status, and literal. Special data formats include bit-string, byte-string, characteristic, mantissa, table entry, LOC (in address), and odd (for even/odd tests).

JOVIAL offers many benefits:

- Mathematical subroutines are shared with FORTRAN.
- Hardware registers may be loaded, tested, and stored through the ASSIGN statement.
- Data files may be maintained using either the BCD or ASCII character sets.
- Powerful table-handling language permits manipulation of either serial or parallel table arrangements.

Macro Assembler (GMAP)

The Macro Assembler program is a two-pass symbolic language assembler that provides the programmer with the convenience of coding in open-ended (macro) language or directly in machine-oriented symbolic instructions. The principal functions performed by the assembler are:

- Translation of control and assembly-edit formatting pseudo operations
- Recognition and translation of addresses that are absolute or relative to subprogram origin, to common storage, to labeled or block common storage, and to externally defined symbols
- Production of relocatable or absolute-binary subprograms that can be combined at load time
- Allowance for programmer-defined macro instructions at assembly time
- Provision for accepting compressed symbolic decks plus any desired alter cards as input, and producing an updated compressed deck as output
- Provision for a complete listing of the assembled program, plus a symbol reference table
- Provision for descriptors and multiword instructions.

PL/I

Level 66 PL/I is a language designed for commercial, scientific, and system-programming application. The PL/I compiler requires a Level 66 system with at least 131,072 words (524,288 bytes) of main memory.

PL/I offers a number of advantages:

- The language is designed to be useful for the widest range of programming applications, reducing the costs of programmer training and program maintenance.
- PL/I is designed to be relatively machine independent, reducing the costs associated with changes in either machine environment or computer systems.
- The block structure and separate compilation features of the language reduce effort when rewriting or changing portions of a program.
- The data manipulation capabilities of PL/I are superior to those offered with other commonly used programming languages.

The many capabilities of PL/I include the following:

- PL/I is a block-structured language that permits both internal (local) and external (global) variables and procedures. These language characteristics facilitate the development and maintenance of structured programs.
- PL/I has a comprehensive set of data formats. These include 16 distinct types of arithmetic data, character string, bit string, locator, label, entry, file, and area data. These formats give PL/I considerable data descriptive power.
- Structure Variables (similar to COBOL hierarchical descriptions) let the programmer explicitly define data structures in combinations of PL/I data formats.
- Dynamic allocation for scalar, arrays and structure variables is provided by automatic, controlled, and based storage.
- PL/I has powerful bit-string and character-string handling capabilities. Operations and functions are performed on either fixed- or variable-length strings.
- Condition testing mechanisms allow the programmer to construct flexible program logic with a minimum number of labels.
- The ON statement permits the programmer to respond to most error conditions that may arise during program execution.
- Dynamic locator variables and memory utilization characteristics facilitate the use of list processing techniques.
- Declaration of arrays and data structures with initial values is permitted.
- Both fixed- and floating-point arithmetic data are supported by software. Binary and decimal arithmetic data are similarly supported.
- Uses the full ASCII character set defined in American National Standard X3.4-1968. This offers the user greater flexibility in data file content and communications. The Level 66 BCD character set may also be used for some types of data files.

Special Purpose Languages and Simulators

DATANET-355/6600 MACRO ASSEMBLER (355MAP) AND SIMULATOR (355SIM)

The 355MAP language processor and the 355SIM simulator are provided for programming the Front-End Network Processor (FNP) associated with the Level 66 Information System, i.e., the DATANET 6600 or the DATANET 355 FNP. Thus they are of very limited interest to the average system user. Briefly, 355MAP is a Level 66 macro-assembler for the FNP machine language. Thus it allows a program for the FNP to be assembled on the Level 66 system.

355SIM is a complementary program that simulates the operation of the FNP in the Level 66 central processor, for purposes of testing FNP object programs on the Level 66 (prior to their actual installation in the FNP).

A PROGRAMMING LANGUAGE/66 (APL/66)

APL/66 can be characterized as a line-at-a-time algorithmic calculator with many sophisticated operators and a stored-program capability. Originally developed as a mathematical notation for discussion of the theory of algorithms, APL was refined as a means for expressing an algorithm to the computer. The user needs little or no prior acquaintance with digital computers to use it. After invoking APL, the user types an expression to be evaluated. The APL interpreter performs the calculations, prints the result, and awaits a new input line. The result of an expression evaluation can also be assigned to a variable and remembered from line to line.

In addition, there is a capability for storing input lines by an assigned name, so that a later mention of the name causes the lines to be recalled and interpreted as if they had been entered from the terminal at the time. Finally, there is the ability to save the entire state of an APL session, complete with all variable values and stored programs, so that the user may continue at a subsequent APL session.

NOTE: APL/66 is not part of the system software (in the strict sense) but is available from the Honeywell applications software library. Since it is more a general-purpose than application-specific language interpreter, however, it is included in this overview for the sake of completeness.

GENERAL PURPOSE SIMULATOR SYSTEM (GPSS)

GPSS, one of the most widely used simulation languages, produces accurate simulations of real processes. By establishing mathematical models and furnishing simulation results for subsequent analysis, GPSS provides extremely fast solutions to a variety of business simulation problems. It reduces the need for experimenting with costly prototypes or physical installations.

Any physical process that is already in operation, or under consideration, or envisioned by management - and that can be flowcharted for modeling - can be converted readily for GPSS simulation by the use of a simple activity block diagram. A user of GPSS does not need previous experience in programming, as the rules and practices are tailored to the experience and needs of the practical planner.

NOTE: GPSS is not part of the system software (in the strict sense) but is available from the Honeywell applications-software library. Since it is more a general-purpose than application-specific simulation language processor, however, it is included in this overview for the sake of completeness.

UTILITIES

"Utilities" is a generic term for the set of processors and system programs that provide a wide variety of general-utility functions for the system user (and for system-management personnel).

Bulk Media Conversion

Bulk Media Conversion (BMC) is a powerful program that converts data files from one high-speed or low-speed peripheral device to another. Multiple files and multiple report codes can be processed, and mixed printer/punch images can be present on the input device. Character transliteration for various character sets can be performed on input or output.

BMC performs media conversion for card readers, magnetic tape, disk storage, printers, and card punches. BMC is generally used when the data volume exceeds that permitted by system media conversion routines such as System Input and System Output (part of the automatic functions of GCOS).

BMC has no volume restrictions, however. It processes successive files on a device but does not unscramble mixed files, as does the System Output program. BMC can be used as a free-standing program or as a preprocessor and/or postprocessor of a main program. It can also be used to create multiple copies of output files.

Bulk Media Conversion offers:

- Choice of standard or nonstandard data formats
- Character transliteration (e.g., from various IBM character-code sets)
- Multiple input and output files, sequential and nonsequential
- Ability to spin off BMC activity as a separate job to improve multiprogramming density
- Printer form control
- Creation of tape from input job stack
- Character substitution (card reader data)
- Restart
- Error recovery

Sort/Merge

The Sort/Merge program is a highly efficient tool for sorting and merging files. The program accepts a wide variety of data and task descriptions and adjusts itself dynamically to the individual task to provide the most efficient processing possible. The SORT function also features dynamic adjustment to the operating environment and maximization of resource usage, especially during multiprogramming operations.

Sort/Merge uses mass storage devices, tapes, or combinations of both depending on the configuration of the system and the desires of the user.

The Sort/Merge program is controlled by a set of macro statements through the Macro Assembly Program (GMAP), or by the SORT verb through the COBOL compiler. It requires few descriptive parameters for the average sort or merge application. Unless otherwise restricted, it assumes a standard set of functions and file descriptions. Flexibility of the program is enhanced by optional parameters in the macro statements or in the SORT statement.

This flexibility is further represented by the characteristics listed below:

- Record size from 1 to 16,384 bytes (4,096 words or 24,576 characters)
- No restriction on key size
- BCD, binary, word, double-precision floating point, or single-precision floating-point fields possible as keys
- Three to 16 collation files
- One to 16 input files per execution
- Sorting of short files in memory without intermediate storage files
- Single input merge possible for checking data sequences
- Automatic spill-out to tape as disk areas become full
- Recognition of key fields in any position within the record
- Automatic key transliteration to commercial collating sequence
- Automatic record selection/deletion
- Own-code interface
- Ability to allocate free memory dynamically at execution time
- Mixed alphabetic and numeric keys possible.

UTILITY

UTILITY is a generalized program package within the broad service-program assortment provided for Level 66. UTILITY provides storage-device and file maintenance capabilities, including complete facilities for copying, comparing, positioning, and dumping of device or file content. For example, it performs the following:

- Processes magnetic tapes and sequential or random disk files as specified by user-supplied directives.
- Copies, compares, and dumps a number of files or records.
- Positions a storage device - rewind or skip.
- Repeats specified parameters.
- Processes user-coded instructions added to Utility, e.g., to modify file content, set delimiters, or specify error conditions.
- Processes seven-track or nine-track magnetic tapes.
- Handles mixed-mode and mixed-density tapes.
- Processes any size record if given sufficient memory storage.
- Handles both multireel files and multifile reels.
- Handles nonstandard formatted files.

The UTILITY processor is used mainly for operational and debugging purposes. It is called unconditionally by the $ UTILITY control card, or conditionally by the $ ABORT control card in the event of an activity abort.

UTILITY processes all file formats, storage media, and data representations supported by the File And Record Control Facility, and uses that facility for its I/O operations.

UTL2

The UTL2 utility processor provides essentially the same kinds of storage-device and file maintenance capabilities as does UTILITY (described above), but it handles a wider variety of file formats and is ASCII rather than BCD oriented.

UTL2 is primarily intended for support of sequential, relative, and indexed files in the Unified File Format (UFF), as produced by the Unified File Access System (UFAS). In addition, however, it provides

- Sequential processing of magnetic-tape files in both ANS and IBM file formats, and
- Sequential processing of standard-format files produced through the File And Record Control Facility, on both magnetic tape and mass storage (sequential organization only).

Thus UTL2 provides facilities for file transferability as well as full utility-processing support for UFF files and ANS magnetic tapes.

UTL2 uses the UFAS access methods for its I/O operations.

Source And Object Library Editor

The Source And Object Library Editor (FILEDIT) is specifically designed for the creation and convenient maintenance of program-library files, in either source or object form. Such libraries may be created and maintained by an individual user for himself or his project, or may be set up by the installation for general operational use.

If corresponding source and object libraries are created, FILEDIT provides a very convenient, semiautomatic means of updating the same program in each library simultaneously, in conjunction with the recompilation activity used to make the object-program changes. Programs on the source library can be used as input for recompilations. Programs on an object library can be readily called for execution by means of the $ EXECUTE JCL statement. The programs so executed are loaded by the General Loader, and thus are subject to all of the General Loader facilities and options (program-subroutine linking, relocation, etc.).

An object library can also be used to contain object-form (load time) subroutines, also for input to the General Loader. Such a subroutine library is identified to the loader by means of the $ LIBRARY JCL statement. (A random-library format is more commonly used for this purpose however, e.g., by compiler runtime packages, in the interest of greater efficiency.)

Both source and object libraries are system-standard format sequential files, with individual programs thereon delimited according to system-wide conventions (known to all language processors and to the loader). The files are usually assigned to permanent mass storage, but can also be kept on magnetic tape. An object library is essentially the compiled or assembled counterpart of its corresponding source library (assuming both exist).

Source library programs may be either in full card-image form or in compressed deck (COMDK) form, or in a combination of both.

The main advantages offered by FILEDIT are as follows:

- Elimination of bulky, hard-to-handle card files by use of magnetic storage in an efficient, standardized form.
- Maintenance of corresponding source and object library files in one pass, or independently.
- Provision of storage within the system for source and object programs, and for subroutines, that is conveniently assignable as language-processor input or output, and as loader input.

FILEDIT also produces a very useful record-of-changes file for each object library update.

Object library files are sometimes used as input to the System Library Editor (described below), e.g., for creation of a site's fast-loading production library.

System Library Editor

The System Library Editor (SYSEDIT) is provided for two related but distinctly separate functions:

- Creation and maintenance of program libraries in a special system-loadable format designed for fast program loading
- Editing of a Total System Tape, as part of the procedures involved in system-software installation and maintenance.

The latter function (editing of a system-software tape) is generally of concern only to the technical support personnel responsible for system-software installation and maintenance. The former function (building of system loadable libraries) may be utilized by a group of users or by the site-operations staff, however. SYSEDIT can be used to convert an object library of fully developed "production" programs into system-loadable form. In this form, the programs can be loaded by a specialized, high-speed loader (via the $ PROGRAM statement), rather than by the General Loader. The benefits of this form of program library is increased loading efficiency for frequently executed programs.

SYSEDIT achieves the system-loadable format essentially by using the General Loader to form a fully bound memory image of a program on the object library and then saving this image on the system-loadable file. Thus for "finished" programs the General Loader need perform its linking and binding functions (e.g., combining of program segments, resolution of external address symbols) only once rather than each time a given program is executed. (Also, the General Loader must be called into memory each time a program is to be loaded by it, whereas the high-speed system loader is permanently resident.)

Note that not all of the programs on an object library need be selected for inclusion in the system loadable library.

SECTION V

INTERACTIVE SUBEXECUTIVES

Three major software packages can be classified as Level 66 interactive subexecutives:

- The Management Data Query System (MDQS)
- The Transaction Driven System (TDS)
- The Transaction Processing System (TPS)

Each of these is a subexecutive inasmuch as it operates under the control of GCOS and utilizes common operating-system services, but offers "executive" type services to the application programmer or the end user. Further, each is characterized as interactive since it is designed to support applications that involve direct end-user interactions with the computer system through some type of communications terminal.

Depending upon the nature of the terminal equipment and the application, the interaction may be one-way (user to system only), or may be two-way (user to system and system to user). In the application area generally considered to be transaction processing, the end-user interaction is often one-way, e.g., in the case of an inventory control application where the terminal equipment is a specialized, automated cash register, for example. In application areas considered to be data base management or information retrieval, however, the interaction is generally two-way, and the terminal device is some type of keyboard/display unit.

MANAGEMENT DATA QUERY SYSTEM (MDQS)

End User Facilities

MDQS is oriented towards a somewhat knowledgeable end user in that manipulation of the data base is determined by end-user devised procedures (for updating and/or query), rather than by fixed-form transactions. MDQS provides a very high level, procedure-oriented language (POL) for direct use at a remote keyboard/display terminal. This language is designed so that it can be used in a very elementary, highly automatic fashion (requiring little learning time), or in a very flexible, full-capability fashion by more knowledgeable users.

Generalized "canned" procedures may also be prepared for a group of end users; these can be executed by an end user with parameters that satisfy his momentary processing needs. The canned, or precoded, procedures provide the equivalent of fixed-form transactions for end users with little need for operational flexibility. Consequently, little or no knowledge of the procedure language is required to execute them.

An online reference subsystem is provided, as a supplement to regular documentation.

Data Base Administrator Facilities

The choice of data base structure, and the type of access-method or content-manager software used to manipulate it, is controlled by the data base administrator. The data base administrator uses the MDQS Data Definition Language (DDL) and Application Definition Language (ADL) to define the entire data base, and various application-oriented virtual "views" of the data base, respectively.

MDQS Structure

MDQS is a hybrid structure that utilizes both time sharing and batch facilities to achieve a well balanced, powerful, and cost-effective blend of capabilities. The MDQS interactive-user interface is a TSS subsystem (MDQ); the POL procedures submitted via this interface automatically generate a batch-world program that performs the requested data base processing. (The end user is totally unaware of any implicit batch-world activity, however.)

The net result of this type of structure is that both the interactive subexecutive and the processing programs are invoked dynamically and neither requires dedicated memory space.

MDQS is available in two forms: MDQS/II for a query-only (information retrieval) capability, and MDQS/IV for full update-and-query capabilities. The MDQS/II language facilities are a proper subset of those provided by MDQS/IV. (That is, MDQS/II and MDQS/IV are fully upward-compatible.)

TRANSACTION DRIVEN SYSTEM (TDS)

TDS is designed to support the development of high-volume transaction processing and data base management applications. It is intended for applications involving the processing of fixed-form transactions entered from remote terminals, but can also be used for processing batched transactions entered at the central computer site (or through a remote processor). No programming knowledge or ability is assumed or required of the end users, i.e., the individuals who initiate the input transactions, since no procedural language is involved at the end-user level.

The form of input transactions to be processed (and of the corresponding output messages, if any) is determined by user-written Transaction Processing Routines, which operate under the control of TDS. Thus, transaction formats can be specifically tailored to the application; that is, designed to fit naturally into the end user's working environment. The structure of transactions can be as simple or as complex as desired. Several (or many) different types of input transactions can be defined for an application. Each transaction type is identified by a unique keyword. (All transaction messages must begin with such a keyword.)

Transaction Processing Routines

The function of the user-supplied Transaction Processing Routines (TPRs) is to determine and direct the processing that must be performed in response to the receipt of a message initiating a transaction. This processing often involves the updating and/or retrieval of information in a data base associated with the application, and may result in the generation of an output message for a destination terminal.

TDS software components provide all of the dynamic support functions required by the TPRs:

- Message Manager -- Receives, verifies, and (optionally) transliterates input messages; routes transactions to appropriate TPRs; routes output messages to recipient terminals; and journalizes input/output messages, if required.
- Data Base Manager -- Initiates and monitors the data base I/O operations requested by a TPR; controls concurrent access to the data base by multiple transactions; journalizes "before" and "after" images of data base fields during data base update operations; and journalizes data base "checkpoint" records.
- Transaction Manager -- Initiates and terminates individual TPRs and their associated data base access tasks; coordinates sequential execution of related TPRs for multiphase transactions; and terminates transactions when all related processing has been successfully completed (including output-message transmission).
- Recovery and Restart Manager -- Utilizes 'historical' data from journal files to effect either a recovery of TDS operation after a minor error or malfunction, or a restart of TDS following a more serious error. It also maintains data base integrity in the event of incompleted transactions due to a TDS, TPR, or system failure.

With all of the above-mentioned support functions performed for him automatically by TDS, the application programmer is free to concentrate on the purely logical requirements of processing a transaction when coding a TPR (or a related set of TPRs). Furthermore, TDS is very flexible with respect to the relationship of individual TPRs and transaction types. A single TPR may, if so designed, provide all of the logical processing required for one type of transaction, or, possibly, for several transaction types. Alternatively, a TPR may process only one phase (i.e., one 'piece') of a transaction, and several TPRs may be required in sequence for processing of a single transaction instance. (Therefore, the TPR/transaction relationship may be one to one, one to many, or many to one.)

The TPRs are coded in an extension of COBOL or I-D-S/I COBOL. This language (called TDS/COBOL) includes statements unique to TDS. TDS/COBOL source programs are processed by the TDS Translator. The compiled TPRs are then placed in the TDS Library, in a special TDS-loadable format created by a support program called the TDS Librarian.

TDS Orientation and Structure

TDS is essentially oriented towards the management of a large data base associated with either single or multiple applications. TDS is also oriented towards the use of an I-D-S/I data base organization, but ISP, random, or sequential file structures can be used as well. TDS is especially suitable for development of a dedicated, high-throughput transaction processing system. TDS exhibits the following performance characteristics:

- Fast interactive response time
- Excellent data base protection features, including comprehensive journalizing and dynamic rollback/recovery capability.

TDS is a self-contained GCOS subsystem that is somewhat similar to TSS in that it

- Manages its own dedicated memory space
- Schedules, allocates, and initiates TPRs (as TDS subtasks, not GCOS jobs)
- Performs common message-handling functions on behalf of the TPRs.

Structurally, TDS is a privileged slave program consisting of the TDS Executive and several other modules whose functions were described above, i.e., the:

- Message Manager
- Transaction Manager
- Data Base Manager
- Recovery and Restart Manager.

The TDS Executive controls and coordinates all phases of TDS operations, and is functionally subdivided as follows:

- Startup -- Initializes the TDS Executive with run-time parameters at the beginning of a TDS processing session.
- Dispatcher -- Allocates processor time to various TDS tasks on a priority basis.
- Interrupt handler -- Supervises transfers of execution control between TDS tasks, and services I/O-completion interrupts.

TDS Support Programs

In addition to the dynamic TDS software components described above, the total TDS package includes the following system-support programs:

- System Generation (SYSGEN) Program -- Configures TDS for a specific application, using information provided by the site's TDS administrator. This information is supplied in the form of COBOL-like SYSGEN source statements. One significant category of information that must be supplied to TDS SYSGEN is a description of the installation's data base structure (the data base schema).

- TDS/COBOL Translator -- Preprocesses a TPR source program written in TDS/COBOL. The function of the translator is to translate TDS-specific statements based on information supplied to TDS SYSGEN concerning the data base structure and TPR operating parameters. The translator then invokes the appropriate language processor for full compilation of the program. (During the TDS SYSGEN process, the translator is "tailored" for the installation's applications.)

- TDS Librarian -- Creates and maintains a library of Transaction Processing Routines in a special TDS-loadable format. The Librarian program accepts the user-written TPRs in compiled (object) form, converts them to a memory-image form, and places them on the TDS library file. TDS loads TPRs from this file as required during a transaction-process session.

TESTBED Facility

TDS includes a TESTBED program that allows newly developed TPRs to be tested and evaluated in a batch-mode environment that simulates normal TDS operational conditions. TESTBED facilitates testing of a new TPR against an existing, operational data base, but without danger of impairing that data base because of faulty updates. (All data base updates can be automatically captured on a separate collection file.) Selective trace and dump information can also be generated for debugging purposes.

TRANSACTION PROCESSING SYSTEM (TPS)

TPS is designed to facilitate development of transaction processing applications in a multi-use environment. Its operational characteristics as seen by the end user are basically similar to those of TDS: fixed-form transactions which can be tailored to the end user's business context, no end-user procedural language, and identification of transaction type by leading keyword.

TPS and TDS differ considerably, however, from the application programmer's viewpoint. TPS is in certain respects more flexible and generalized than TDS, and it is somewhat simpler to program for. It can readily handle multiple unrelated applications using different data bases, and is particularly well suited to the development of low-to-moderate volume applications. Such applications can be "gotten on the air" under TPS with a minimum of programming time and effort.

On the whole, TPS requires a less demanding data base administration effort than does TDS for multiple, concurrent applications, but on the other hand does not offer the same throughput potential for a very large volume application.

TPS consists essentially of three elements:

- The Transaction Processing Executive (TPE) -- the controlling element of TPS
- Transaction Processing Application Programs (TPAPs) -- The user-supplied, application-oriented portion of TPS
- The GCOS Interslave Communication Facility (INTERCOM) -- A passive element which serves as a channel of communication between the TPE and the TPAPs.

Each of these elements is described briefly below.

Transaction Processing Executive

The Transaction Processing Executive (TPE) receives transaction input from the terminals and routes the transactions to the appropriate Transaction Processing Application Programs (TPAPs) for processing. When the transaction has been processed, the TPE directs the output back to the terminal or, if the terminal is not available, holds the output for later transmittal.

The TPE executes as a privileged slave program under GCOS, and is started and terminated by the system-console operator.

During normal transaction-processing operation, the TPE performs several functions:

- Checks incoming transaction messages for validity
- Journalizes both input and output data to ensure against loss of a transaction or its output and to provide a base for TPS restart should the need arise
- Queues input transactions according to user-established priorities
- Initiates TPAP execution as required
- Monitors TPAP execution to ensure an orderly flow of transactions and to determine the status of the TPAPs at all times.

The modular design of the Transaction Processing Executive program permits easy incorporation of the user-supplied parameters (TPAP profiles) required for system operation.

Transaction Processing Application Programs

Under TPS, the application programmer writes a TPAP that defines and processes a specific type of transaction, or several related types if desired. The TPAP can be written in any of the general-purpose programming languages supported GCOS (e.g., COBOL-68, COBOL-74, FORTRAN, GMAP, PL/I). Thus, any file or data base organization can be utilized, including I-D-S/I, I-D-S/II, and UFAS organizations, and any appropriate access-method or content-manager software can be employed to manipulate it.

Normally, one TPAP will contain all of the processing logic and I/O procedures required to process a given transaction, but several sequentially executed TPAPs may be utilized if desired. (Sequential execution of multiple TPAPs can be made conditional, dependent on transaction content and TPAP processing logic.)

The TPAPs are stored as independent programs in the GCOS file system, to be spawned (called into execution) by the TPE as needed to process transactions submitted to the system. Very frequently used TPAPs can be held in memory, to be enabled as needed to process transactions. Most TPAPs can be swapped in and out, however, to accommodate a wide variety of incoming transactions. (The memory residence characteristics of a given TPAP are controlled by parameters in the TPAP profile. This profile information is supplied by the programmer, for incorporation into the TPE.)

Each TPAP is invoked by TPS as a separate GCOS job, consisting of one execution activity. A wraparound provision in the TPAP-TPE interface permits the output of one TPAP to be used as input to another TPAP for further processing. This provision facilitates the sequential TPAP execution mentioned above.

Interslave Communication (INTERCOM) Facility

The GCOS INTERCOM facility provides a simple, standardized means of intercommunication between the TPE and TPAPs, for both input and output messages. The INTERCOM facility essentially simulates logical input/output operations within main storage, for the high-speed transfer of small amounts of data between two independent programs (in this case between the TPE and a TPAP). That is, INTERCOM provides pseudo files -- actually memory buffers -- which one program can "write" to and another can "read" from by means of a standard I/O-like interface.

Thus, transaction input from a terminal is sent to a TPAP (by the TPE) via one INTERCOM file, and transaction output is placed in a second INTERCOM file by the TPAP for transmission (via the TPE) to a terminal. Optionally, output from a TPAP can be passed to another TPAP through the same INTERCOM mechanism, again via the TPE.

For TPAPs written in COBOL, the INTERCOM interface is implicit and virtually transparent, since normal COBOL-68 or COBOL-74 syntax is employed, with only minor extensions where necessary. (For example, in COBOL-68, extended forms of the ACCEPT and DISPLAY statements are used, in conjunction with a COMMUNICATIONS-DEVICE clause in the SPECIAL-NAMES paragraph.)

For compiler languages other than COBOL, two standard interface subroutines are provided: COMMI for input, and COMMO for output. These subroutines are invoked, and supplied with required arguments, by the CALL statement that is common to all Level 66 compilers.

Note that TPAPs can process transaction data in either BCD or ASCII, as appropriate to the programming language and the data base organization utilized. The TPE will provide any code transliteration required.

TPAP Test Features

New TPAPs can be checked out during the development cycle, i.e., prior to normal incorporation into TPS, through the use of a "built in" TPAP profile especially provided for this purpose. (Several such checkout profiles are included in the TPE.)

Another feature that can be utilized by a TPAP during its development is the training/testing mode "switch", provided by the TPE. This software switch can be turned on or off by the terminal operator. A TPAP can be coded to ascertain the state of this switch, so as (e.g.) to bypass or modify its file update procedures. Thus it can avoid faulty updates when run against an actual data base. (The switch can also be used by an operational TPAP to enable tutorial logic when the terminal operator requests "training mode".)

SUMMARY

The range and variety of the major Level 66 interactive subexecutives is such that an appropriate tool is available to fit the particular characteristics of each installation's interactive-processing needs. (Lesser retrieval-only facilities, such as I-D-S Data Query, are not covered here.)

Each of the major subexecutives has advantages and drawbacks relative to the others. That is, each offers a particular "mix" of capabilities oriented towards a general class of applications. A careful choice of the tool that "best fits the job in hand" can forestall most if not all of the undesirable tradeoffs that have characterized many recent large-scale interactive applications.

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE: SERIES 60 (LEVEL 66) SYSTEM SOFTWARE OVERVIEW

ORDER NO. DE61, REV. 0

DATED JULY 1976

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

19372, 7.5C1177, Printed in U.S.A.

DE61, Rev. 0