Chapter 3

PTS SOFTWARE

3.1 General Description

As mentioned previously, there are two operating systems on PTS, one each for the production and development environments. The production system, TOSS, allows programs to be tested and run, while the development system, DOS-PTS, allows programs to be written, updated and linked.

3.2 The Production Environment

The production environment is the one most commonly employed for PTS systems. The application program is written at a central computer department, then distributed to all the branches that need to run it. Each branch, therefore, has a completely written and tested program, and no further development need be carried out.

As written at the central site, the application may have to cater for widely differing sizes of branch, from a remote rural office to a large city branch. However, the greatest difference between the requirements of various branches is the number of terminals to be connected. As this value is supplied when the system is loaded (as part of the Configuration Data), the application program itself is identical in every branch. The only difference will be in the configuration data. This makes the system extremely flexible, and if, for example, a branch adds an extra terminal during the life of the application, all that needs to be changed is the configuration data file.

3.2.1 TOSS

The operating system used in the production environment is TOSS (Terminal Operating System Software). TOSS is supplied as part of the system software of the computer. The parts necessary to satisfy the requirements of the particular application program can be selected and linked together to form the TOSS Monitor. As it is unlikely that two different systems will make identical demands on the system software, it can be seen that no two Monitors are likely to be identical.

The TOSS Monitor is generated at the central site in parallel with the application and distributed to the individual branches in the same way. Again, as long as the types of device in the system do not change, the Monitor will not need to be altered during the life of the application. This means it is possible to change the number of terminals in a branch without affecting the Monitor in any way.

3.2.2 Tasks

Both the Monitor and the application program are generated without regard to the number of terminals present in the final system. This information is supplied as part of the configuration data. However, in the running system, each connected terminal must be controlled in some way, for instance to prevent two operators trying to access the same account record at the same time. To enable this kind of control to be exercised, a method of identifying the operations performed by each terminal has been developed.

The actions of each terminal are controlled from within the Monitor. The part of the Monitor responsible for controlling one terminal is known as a task. Thus, there will be one task in the system for each terminal connected. Each task has its own unique identifier.

Other tasks will, however, also be present, as the Monitor uses tasks to perform certain system functions, such as accessing disk files. These tasks are not related to any particular terminal. Their number and types depend upon the way the Monitor was generated.

3.2.3 Terminal Classes

It has already been stated that when the application is written, no account is taken of the number of terminals, and hence the number of tasks, that will exist in the final system. However, those tasks will execute the coding written as the application program, so some way of relating the tasks to the coding of the application program must be available. The Terminal Class is the method of doing this.

All similarly-configured terminals running similar transactions form one terminal class. The application is then written as if only one terminal exists for each terminal class. At run time, when the system is loaded, the configuration data tells the system how many tasks exist for each terminal class. The relevant parts of the coding can then be duplicated to enable each terminal to function correctly.

3.2.4 Data Communication

PTS computers are often connected to other systems by data communication lines. Standard software modules may be included in the Monitor to handle many different data communication protocols from a number of different mainframe manufacturers. These protocols range in scope from a simple point-to-point connection to full networking capabilities.

3.3 The Development Environment

The development environment is the one in which application programs are written. It therefore has different requirements from the production environment and uses a different operating system.

3.3.1 DOS-PTS

The development operating system is DOS-PTS, the Disk Operating System. This operating system allows programs to be written, translated and linked to form an interpretable program. Testing has to be carried out under TOSS.

DOS-PTS is a single-user system that contains all the processors and utilities necessary for application program development and for generating the TOSS Monitor. The various system programs are called into execution using simple commands from the operator's console. The most important commands are those dealing with the language processors.

3.3.2 CREDIT

CREDIT is a language dedicated to the PTS system. As such, it provides a level of flexibility in handling the range of input/output devices in the system which would be difficult to achieve with traditional high-level languages.

The language is interpretive, which means that the object code produced by the CREDIT translator (compiler) is not directly executable, but has to be processed by an interpreter at run-time. This interpreter decodes each instruction to an executable form. This can save considerably on memory size, since the executable code for each instruction need only be held once, in the interpreter, and not every time the instruction is coded.

The application program is written as a series of modules which are subsequently linked together to form the working application. These modules may be available to all tasks in the system, or they may be dedicated to the needs of an individual terminal class.

The translator produces reentrant coding, shared by all the tasks that need to include that particular sequence of instructions. This also reduces memory size, as common coding is only held in memory once.

One module contains a Data Division, where all the data areas for the application are located. The other modules all refer to the data areas

defined in this module. Thus, once the application's data areas have been defined, there is no possibility of invalid data references occurring in the final application.

CREDIT supports four types of data:

Boolean data	one-bit fields, used for holding flags, etc.
Binary data	one-word fields, holding numbers in the range -32768 to +32767 in pure binary format
BCD data	(Binary Coded Decimal data) numeric values of from 2 to 512 digits, each digit held as a four-bit packed decimal value
String data	alphanumeric values of from 1 to 4095 characters, each character held as an 8-bit byte in ISO-7 code

Data items used by a CREDIT program may be either global (available to all tasks) or task local (available to a single task only). Thus, security of data between tasks can be maintained while data needed by all tasks is freely available.

As CREDIT is a language dedicated to PTS, it offers a very wide range of instructions to control the special devices necessary in a banking environment. As well as the instructions for reading and writing information which may be found in any language, CREDIT allows greater control to be exercised over such things as positioning documents in a teller terminal printer, positioning the cursor on a screen and so on.

3.3.3 Screen Management

Since virtually every application written in CREDIT will make extensive use of terminal equipment, and in particular screens and keyboards, a standard package has been written to simplify the programming of such applications. This package is called Screen Management.

Screen Management treats each screen of data as a complete unit and relates all the data fields on the screen to a corresponding data item in memory. The operator can key in the values to each field, either in a predetermined sequence or at random, depending on the requirements of the application. The package performs all the validation necessary. Some checks are incorporated in the package itself, others are included in user-written routines called by the package.

All the statements necessary to accept input from the keyboard and display to the screen are included in the Screen Management routines. This reduces considerably the programming effort required and makes for much easier screen and keyboard handling.

3.3.4 Work Station Management

The Work Station Management (WSM) package is another software package to facilitate workstation handling. As well as the facilities provided by Screen Management, WSM allows the user to have fixed and removable parts on the screen during one transaction, and to display guiding messages for the operator.

The major difference between WSM and Screen Management is that WSM is not included in the user application but runs as a separate task alongside it. The format coding defining the screen layouts is held on a disk file, allowing several user tasks to use the same formats concurrently. Special WSM instructions are included in CREDIT to enable the application to request the functions available.

Formats to be used by WSM are defined with the WSM Creator tool. This is an interactive program which enables the user to design the screen layout while working at the screen, and define the operator input required by specifying a number of parameters for each input field. Error and guiding messages that will be displayed when the related input field is processed may be defined. In addition, a validation routine may be defined for each input field to perform calculations on the input data before it is passed to the application. Validation routines are written in BASIC.

A set of WSM utilities is used for creation and maintenance of the files holding the format code.

3.4 Debugging

To help test and debug CREDIT programs, a special debugging program is available. This debugging program runs as an interactive task alongside the application tasks, and may be used to insert break-points in the coding. When one of these break-points (known as a trap) is reached, execution of the application is halted and the contents of memory locations may be examined and changed, if necessary. This allows small changes to be made to the program during testing without having to go through the lengthy process of updating and retranslating source modules, relinking the application, etc. After the changes have been checked, the necessary updates to the source code must of course be made and the process repeated.

The debugger is not normally included in a tested and running production system.

3.5 Utilities

Two sets of utilities are available for use under the two operating systems.

3.5.1 DOS-PTS Utilities

The DOS-PTS utilities are primarily concerned with keeping the files of each user in a consistent state during the process of developing an application. They allow such precautions as dumps of each user library or of complete disks for backup purposes, and provide the routines necessary to restore a corrupted disk.

A standard utility under DOS-PTS allows four TOSS utilities to be run, to allow a TOSS disk to be formatted, individual files to be created and deleted, and the VTOC to be printed. This facility is necessary when converting an application from DOS-PTS format to TOSS format so that it may be loaded and run.

3.5.2 TOSS Utilities

Whereas the DOS-PTS utilities are mainly concerned with handling groups of files, the TOSS utilities work mainly on individual files. These files are those needed in the production environment and so are mostly data files.

Many of the utilities are used for Data Management files, to maintain these files in a usable condition.