PLF MAB

ICL
ONE PER DESK

Programmer's Reference

INTRODUCTION

INTRODUCTION

This chapter provides a general introduction to programming for the ICL ONE PER DESK. It describes the framework within which applications are written, under the following headings:

Concepts
Application development
Menus
Application structure
System calls

1 Concepts

The OPD has a number of special features that are essential background to the development of applications to run on it. Two are fundamental:

1 More than one application can run at the same time.

2 Only one application can use the screen and keyboard at a time.

Applications must therefore make arrangements for periods when they can and cannot use the screen and keyboard, and for sharing the resources of the OPD with other applications when they are running.

1.1 SELECTING APPLICATIONS

* The application that has the use of the screen and keyboard is called the foreground application.

* The user selects an application to run by first pressing the START key. This causes the display of the Top Level Menu, which lists the primary options on the OPD. The user chooses from these by pressing the key corresponding to the number on the list. One option is the Applications Menu, which lists the applications stored on capsule or Rompack currently plugged into the OPD, and any applications that have been loaded from cartridge and are still running. The Applications Menu has another option, the Cartridge Menu, which provides a list of the programs stored on the cartridges in the microdrives.

  Built-in applications, such as Telephone Directory, are accessed through other options on the Top Level Menu.

* The user can replace the foreground application with another application, by pressing the START key, as described above. The Top Level Menu is again displayed and the user can make a selection from it. Some applications may be able to continue processing without the screen and keyboard; if so, they do so, and are then running in the background. If not, they

must wait until they get the foreground again.

* The user can return to an application that is waiting or running in the background by pressing the RESUME key. A menu is displayed if more than one application is in these states, so that the user can make a selection.

* Thus the OPD can start new jobs in response to demand, and go back to unfinished jobs that were running previously.

* The user can take a quick look at the data associated with an application that is running (whether in the foreground or the background) by pressing the REVIEW key. This causes the Review menu to be displayed, and the user can choose what he wants to review from this list. (Further details of the use of Review are in the Handbook.) The resulting screen is only displayed while the selection key is held down: when it is released, the OPD goes straight back to the interrupted application.

  All applications must therefore be able to give up the foreground immediately when the user presses the REVIEW key, and then restore the display following the review.

* Applications are classified into two categories: extended and transient. Extended applications are returned to after an interruption (for example by START, RESUME or REVIEW). Transient applications are lost if they are interrupted by START or RESUME (although not by REVIEW). If the user wishes to return to an interrupted transient application, he must restart it from the beginning. The same application may be classed as extended and transient at different times.

1.2 SHARING RESOURCES

* Execution of code in OPD is in activities. An activity is an independent processing thread, with its own stack, registers, program counter etc.

* Activities are scheduled according to their priority by the system software, like a low level scheduler in a conventional operating system. Thus, activities appear to run in parallel and asynchronously, although where necessary they can be synchronised by the use of events and semaphores.

* Information is passed between activities by means of numbered events, which indicate that particular conditions have occurred; these may be predetermined by the system, or be defined within an application. Also, an activity can suspend itself until an (unspecified) event occurs.

* Semaphores can ensure that only one activity uses a particular resource at a time. The activity unlocks the

semaphore when it has finished using the resource, and the semaphore (and thus the resource) is then available to the next activity requiring it.

* The store is managed in a way that enables activities to obtain the quantity of store they need at the time they need it, and release it when they have finished with it. The dynamic store (RAM) is divided into segments. Some segments remain at the same address (for example, those containing activity stacks), but most can be moved within store except when being accessed by an activity. This is to prevent store fragmentation.

2 Application development

2.1 LANGUAGE

Programs can be written for the OPD in Assembler, C or BASIC; they can also be written in any high level language that can be compiled directly into machine code.

For a summary of the diferent development routes for OPD applications see the publication OPD Guide to Software Development.

2.2 SUPPLY MEDIA

Programs can be supplied on either of the following media:

1 A capsule to be inserted into a Rompack (see the Handbook).

2 A cartridge to be inserted into one of the microdrives.

For normal applications, the medium does not affect the actual code of the application.

Program formats for both media are given in Director Facilities for Application Writers, PSD 76.97.3.2.

2.3 HARDWARE

The hardware components that are relevant when developing applications are listed below. Some of them are represented for input/output purposes by more than one logical device.

Screen (represented by the Application Screen and Noticeboard)

Keyboard

Telephone (represented by one or two telephone lines)

Modem

Voice synthesiser

Printer

Two microdrive storage units (represented by the microdrive filing system)

Tone generator

Real time clock

Store (see section 2.4)

Permanent store

2.4 STORE

The store consists of the following elements:

1 128 Kbytes of dynamic RAM, known as the main store. The current screen image accounts for 32 Kbytes of this.

2 2 Kbytes of permanent RAM, used for long term storage of parameters. The contents of this element are not lost if the OPD is switched off or fails.

3 ROM containing system and other software.

These are supplemented by the following optional items of store:

1 Microdrive cartridges. These contain an endless tape loop, and provide storage for programs and conventional files. The OPD has two microdrives, each of which can hold a single cartridge.

2 A Rompack, which slots into the back of the OPD control unit. It provides storage for:

(a) The Xchange applications, if supplied (see Xchange)

(b) Capsules, which may store the Computer Access application (see the Handbook) and user-written applications

Further information about the management of the store is in the Kernel Specification, PSD 76.97.3.1.

2.5 SOFTWARE RULES

Applications written for the OPD should conform to a set of rules, to provide consistency of approach across applications and uniformity of user operation. Some of the rules are given in section 1.1, for example the way in which the START, RESUME and REVIEW keys work. Applications that conform to the rules are called trusted applications.

At present, untrusted applications are not supported, and it is recommended that normal applications should not use untrusted programs or untrusted activities. However, information on handling them is given in Director Facilities for Application Writers, PSD 76.97.3.2.

Applications cannot be implemented to affect the operation of the following:

1 The noticeboard area of the screen, except in a controlled way (see the Kernel Specification, PSD 76.97.3.1).

2 The system keyboard (see the Kernel Specification).

3 The telephone facilities, except as specified in the Telephone Handler document, PSD 76.97.3.3.

3 Menus

Standard OPD software uses menu screens to run applications, and it is recommended that user-written applications adopt the same approach. Each menu screen contains a list of options, with suitable explanatory information or data; the user selects an option from this list. A hierarchy of screens may be needed to achieve a particular function, and a single menu may spread over more than one screen. In the latter case, the 0 key is conventionally used to proceed to the next screen.

When the user is going through a series of menus in this way, there is a half second delay between the display of the menus. The experienced user may be able to remember the sequence of keys used to reach what he wants to do; if, having made one selection, he makes a further selection within the delay period, the intervening screen is not displayed. For example, he may need to press 2 in response to menu A to give menu B, and 3 in response to menu B to give menu C. If he has menu A displayed, and presses 2 and then 3 before menu B is displayed, the display will go straight to menu C, and menu B will not be displayed.

4 Application structure

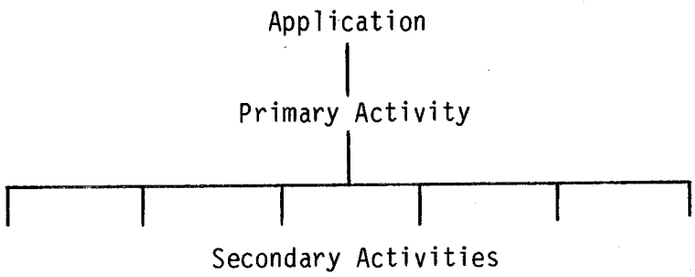
This section describes the structure of applications written for the OPD, and defines a number of concepts used. How to invoke and terminate applications is described in Director Facilities for Application Writers, PSD 76.97.3.2.

An application is the unit in which high level scheduling is performed by the OPD user. It corresponds to the job, or session, of a conventional operating system. High level scheduling on the OPD has two aspects:

1 Control of when the application shall run and when it shall terminate.

2 Control of when the application shall have the foreground.

The execution of an application comprises at least one actvity (see section 1.2), known as the primary activity. This activity is special in that it is responsible for coordinating the response of the application to the high level scheduling of work on the OPD. The execution of an application may comprise additional activities, known as secondary activities. The application designer may choose to exploit these where the application's function maps concurrently onto asynchronous tasks. Secondary activities can be created and destroyed by the primary activity, or by each other, to perform the application.



Activities communicate with the basic software and logical devices by means of system calls, each of which performs a limited and specific action, in most cases defined by parameters.

Secondary activities communicate with the primary activity by means of events, semaphores (see Kernel Specification, PSD 76.97.3.1) and shared data. They can handle devices whose actions are not synchronised with those of other devices used by the application. For example, the primary activity may handle the screen, while secondary activities handle the modem, which may be required to operate continuously, whether the application is using the screen or not.

5 System calls

System calls specify a particular action to be taken, or request information. In most cases information is returned following the call.

For example, system calls enable an activity to:

* Request and use various input/output devices
* Request store on a permanent or temporary basis
* Create and destroy other activities
* Synchronise and communicate with other activities
* Suspend itself or other activities until an event occurs

The calls all have the same general format. The generic type of system call is entered in the TRAP name instruction. The name is a symbolic form of the actual number that has to appear in the machine code. The mapping of names to numbers is given in an INCLUDE file (see Appendix 4).

Each activity has 16 registers, and the exact action required is specified in the least significant byte of the activity's D0 register. Other input information is entered in other registers of the activity: for example this may include the identifier of an input/output channel, or an address where further information can be found. Information is returned to the activity's registers or sometimes to locations in the store.

Following a system call, a successful return is indicated by a positive or zero value in the calling activity's 32-bit D0 register. A negative value indicates that the call has failed, and the register then contains a value associated with a name of the form:

ERR.cc

where cc defines the error detected. The codes are listed in the Kernel Specification, PSD 76.97.3.1. The condition code register is set according to the value returned in the calling activity's D0 register, as if a TST.L D0 instruction had been executed (see MC68000 16/32-Bit Microprocessor: Programmer's Reference Manual).

Descriptions and definitions of system calls are given in the documents included in this binder. Their names are given in capital letters, for easy identification. They are also listed in the index in this way.