Director Facilities for Application Writers

PSD 76.97.3.2

0. DOCUMENT CONTROL

0.1 Contents

0.2 Changes Since Previous Issue

The document has been revised both to provide an accurate specification applicable to all existing and anticipated releases of the component, and also to define additional rules and facilities which need to be taken into account by writers of applications that are intended to run compatibly on both early and later releases of the software and hardware.

The changes are as follows:

1. Rules are defined for writing programs that may be used in an environment that supports paged store. A new property to control page switching is defined in the program header.

2. The requirement for registration of use of a program by an activity is clarified. LOAD PROGRAM and RELEASE PROGRAM are extended to allow registration and de-registration of a program identified by descriptor instead of by name.

3. The program header is extended to accommodate additional fields, and a new interface GIVE PROGRAM HEADER FIELD is provided to read such fields.

4. A new interface GIVE SYSTEM VERSION NUMBERS is provided.

5. The format of programs in ROM is clarified, and extended to include ROM checksums.

6. Minor clarifications and corrections have been made throughout the document.

0.3 Document Predecessors

OPD/DIR/1

0.4 Changes Forecast

Issue 4/0 of this document is intended to be the stable

definition of the component for use by writers of applications that are intended to run compatibly on both early and later releases of the software and hardware. No changes are expected apart from clarifications and correction of errors.

Further issues may be made to describe facilities provided in later releases.

0.5 Document Cross References

[1] PSD 76.97.3.1 OPD Kernel Specification
[2] R51002 OPD Handbook

1. GENERAL

1.1 Scope

This document defines the function of the Director component of the OPD software, and the interfaces it provides for the support of OPD applications. It states the rules for writing applications so that they fit into the OPD system architecture. It does not set out to specify how Director works: this is covered in internal design documents. Certain intimate system applications and activities will require information beyond the scope of this document.

1.2 Introduction

OPD can be regarded as two devices integrated into a single system: a conventional telephone with advanced telephony features and a personal computer capable of running applications, especially those that exploit telephony.

The lowest level of OPD software is the Kernel, which is responsible for functions such as resource allocation, device access, event handling and low level scheduling. Director is a further level of software, whose function is to provide a suitable control interface for the end user, and to ensure consistent and disciplined use by applications of the low level Kernel facilities. Applications will interface both with Kernel and with Director.

Director is a generic term applied to that part of the intimate system software lying outside Kernel but not necessarily forming part of any application directly perceived by the user. It comprises a number of subsystems, in particular:

1. Application Handler

   This subsystem is concerned with the control and support of applications with respect to the computer aspect of OPD. It acts as a high level scheduler, allowing the end user to decide which applications shall be run, and which of these shall at any time be using the screen and keyboard. It interfaces with applications so that they respond appropriately to built-in OPD architectural features such as the REVIEW mechanism. It handles access by contending applications to shared facilities such as the screen, keyboard, noticeboard and sound generator.

2. Telephone Handler

   This subsystem controls conventional use of the telephone by the end user, and provides facilities for applications to use the telephone. Details of the facilities are outside the scope of this document.

OPD applications are constrained to obey certain rules in order to support the OPD architecture. The rules are concerned particularly

with the way in which the screen and keyboard are used and with the response to use of special OPD keys such as START and REVIEW. All machine code applications must obey the architectural rules; the system software provides no environment for enveloping machine code applications that do not obey the rules. Such environments could be provided by higher levels of software. In particular, interpreters or run time packages for applications written in languages such as BASIC are expected to support the OPD architecture on the application's behalf.

1.3 Terminology

See section 4.1.

2. SUMMARY

This document defines the interfaces provided by OPD Director software for the support and control of conventional applications, and states rules and conventions for writing such applications to conform to OPD architecture and philosophy.

3. DIRECTOR INTERFACE

3.1 General

As far as possible the mechanics of the Director interface are designed to be compatible with the Kernel interfaces.

The bulk of the Director interface comprises a set of procedures, which are called by obeying the instruction:

```
TRAP #T.DIRECTOR
```

The particular Director interface required is specified by an action value in D0.B. Each procedure defined in this document has a full name, used in descriptive material in the text, and an abbreviated name. The formal definition of each procedure includes an action value of the form D.abbreviated-name. These action values are provided in the INCLUDE file D9VALUES. The variable names are intended to be used in calling code for documentary reasons. For example, the procedure REQUEST FOREGROUND is called by setting D0.B to D.REQFG.

In all cases where a call parameter is specified as of byte or word length, the value of the remainder of the register is immaterial.

The caller of a Director procedure must ensure that at least 128 bytes of stack space exists below the location addressed by A7(SP) in (frozen or immobile) store owned by the caller.

On exit from a Director procedure, D0.L contains a response code, which in general is negative if the call has failed and positive or zero if the call has succeeded. The discrete negative response codes which may be returned are identified in this document by names of the form ERR.cc (or just cc); the ERR.cc values are supplied in the Kernel INCLUDE file ERRORS, and a consolidated list of them is given in [1].

Further data and/or address registers may be defined to contain parameters and/or return values. Any register that is not defined to return a value is preserved. In particular A7 is always preserved. Following a failure (negative) response, any register used to return a value is undefined unless stated otherwise.

The value of CCR (the Condition Code Register) on entry to a Director procedure is immaterial. On exit, CCR is set as follows:

N	Set if D0.L is negative. Cleared otherwise.
Z	Set if D0.L is zero. Cleared otherwise.
V	Always cleared.
C	Always cleared.
X	Undefined.

4. APPLICATIONS AND ACTIVITIES

4.1 Terminology

A program is a named piece of executable or interpretable code. It is the unit of code storage in ROM or on a file store medium such as a microdrive cartridge. See Appendices 1 and 2 for details of the formats and properties of programs.

A program can have the property that it is an application. Application Handler acts as a high level scheduler for the execution of applications, enabling them to be started by the user or by another application, and to be moved to and from the foreground (where they have use of the screen and keyboard) as the user requires. A program that is not an application cannot be separately scheduled in this way, but can be called by applications.

An activity is one of a number of concurrent processing threads, with associated registers etc. See the Kernel specification [1] for further details. The low level scheduling of work on the OPD is in terms of activities. The execution of an application comprises one or more activities.

A segment is a slice of the RAM, and is the gross unit in which Kernel provides RAM space for use by applications. A segment has properties, and in some cases a name, which are controlled partly by Kernel facilities (see Kernel specification [1]) and partly by Application Handler (see section 7).

4.2 Application structure

Each machine code program is either trusted or untrusted. A trusted machine code program is one that obeys the rules contained in this document. The rules are largely concerned with the way the screen and keyboard are used, in order to maintain the architectural features of OPD.

(The 'trusted' concept does not apply to non-machine code programs. The interpreter for the language in question will be a trusted machine code program and will obey the OPD architectural rules on the program's behalf.)

An activity executing a trusted program is a trusted activity; an activity executing an untrusted program is an untrusted activity. An untrusted activity created by a trusted activity runs under the control of that trusted activity. An untrusted activity created by an untrusted activity runs under the control of the controller of the creating untrusted activity. See the Kernel specification [1] for details of the control mechanisms for untrusted activities.

A given execution of an application comprises, as a minimum, a trusted activity called the primary activity of the application. This activity is responsible for coordinating the response of the

application to changes in OPD system status. Only one instance of a given application can be executing at any time.

An execution of an application may additionally include any number of trusted secondary activities. These activities are created and destroyed by the primary activity or by each other as necessary to perform the application. They must communicate (via events, semaphores or data) with the primary activity to achieve any required response to changes in OPD system status. They are likely to be used to handle devices whose actions are unsynchronised with those of devices owned by other activities in the application. For example, the primary activity might be responsible for handling the screen while a secondary activity handles the modem, whose actions may be required to continue whether or not the application is currently in foreground mode.

Each primary or secondary activity may also control any number of untrusted activities. It is not intended that applications written specifically for OPD should normally need to use untrusted activities, except in specialised cases such as language interpreters.

4.3 Application types

The end user perceives the actions of the OPD computer in terms of applications; the application is the unit which the user invokes, terminates, moves to and from the foreground etc.

The intimate system software (such as Kernel and Director) and certain standard applications such as Telephone Directory are built into the OPD.

Further applications are provided in the Rompack, or in capsules, or on file store media such as microdrive cartridges.

Once an application has been loaded into the machine, the facilities it can use and the way it is controlled by the user are not affected by the medium from which it was loaded. The application writer must note, however, that applications in capsule or loaded from files generally occupy unpredictable addresses, and hence must be written in a position-independent manner.

Programs (in the sense of units of code in ROM or on file store media) written for OPD may be marked as not representing applications. They are intended to be loaded only by an application (e.g. as an overlay or as code shared between applications), and are not presented to the end user as applications.

Programs representing applications may however be marked as "invisible". Such applications are concealed from the end user. They can be started only by auto-entry or START APPLICATION. This facility is intended to support add-on device drivers, and similar mechanisms for extending or customising the basic machine. Such

software is expected to exhibit the same level of integrity as the intimate system software.

4.4 Application invocation

An application is invoked in one of the following ways:

1. By selection from the Top Level Menu (or one of its subsidiary menus)

2. By selection from the Review Menu

3. By an existing application calling START APPLICATION

4. By Auto-Entry (see section 4.4.1)

The required application is identified by the name of the program that is to be executed. If the program is in machine code and is marked as trusted, it is directly entered as the primary activity of an application named after the program (except in the Review case, see below). See Appendix 1 and Appendix 2 for the way in which program properties are recorded in ROM and on file store media. An untrusted machine code program cannot be invoked as an application.

If the target program is not a machine code program, a special system program is entered as the primary activity of an application named after the target program. The system program determines and invokes the appropriate interpreter or other execution environment for the target program. The details of this mechanism are beyond the scope of this document.

The initial state of the new primary activity is defined in section 4.4.2.

An application invoked by START APPLICATION or Auto-Entry will appear in the user's view in the same way as applications invoked via the system menus (unless the application is marked as "invisible").

An application invoked by selection from the Top Level Menu (or one of its subsidiary menus) is immediately moved to the foreground. An application invoked by START APPLICATION or by Auto-Entry will normally run initially in background mode, although if it uses the screen it will become a contender for the foreground on the Top Level Menus or Resume Menu. (See also section 4.4.1.)

Options are provided with START APPLICATION to enable an application running in foreground mode to transfer the foreground (the right to use the screen) to a newly started application, or to an application already running in background mode. This provides a way of CHAINing applications without user intervention.

A call on START APPLICATION can specify that this is a

'tele-start', i.e. the request to start the application was initiated from outside this machine (for example, a request received via T-Link). The call will fail unless the target application has the bit set in its program header (see Appendix 1) or, for a program in a file, in its file type qualifier (see Appendix 2), indicating that the application is suitable for tele-starting.

When an application is entered to perform a Review, the program is entered within a permanent Director activity that handles such occurrences, rather than an activity of its own. Details of Review processing are given in section 6.

4.4.1 AUTO-ENTRY TO APPLICATIONS

There are two ways of causing automatic entry to applications on power-up or following a System Reset:

1. An application in ROM can be marked as having this property (see Appendix 1 for layout conventions for programs in ROM). Such applications in ROM which are addressable at power-up or reset time will be entered automatically, but not moved to the foreground.

2. An application name can be configured into permanent store. If this application cannot be found in ROM, a standard load sequence will occur to load it from the available file storage devices. The application will be entered automatically, and will be moved to the foreground without user intervention unless an error has occurred during the initialisation sequence. This application is termed the First Application. It does not itself need to be marked as "auto-entry".

An application on a file store medium cannot be auto-entered unless it is configured as the First Application.

The order in which the applications are invoked is undefined.

[If it becomes possible to insert capsules without causing a reset, applications marked as "auto-entry" in a newly inserted capsule will be auto-entered following the insertion.]

4.4.2 STATE OF ACTIVITY ON ENTRY

This section defines the initial state of an activity invoked by the Director functions START APPLICATION and START ACTIVITY.

For an activity started by direct use of the Kernel interface CREATE NEW ACTIVITY, the initial state of the activity is defined in the Kernel specification [1].

4.4.2.1 Priority

For activities invoked by use of START APPLICATION or START ACTIVITY the priority is set by the invoker.

For an application invoked via the Top Level Menu (or one of its subsidiary menus) or auto-entered, the primary activity has an initial priority of 63.

The activity in which a review entry is made has a priority of 63.

A priority greater than 63 should only be used where there is a specifically evaluated requirement to achieve a faster than average response to events. This might apply to activities handling the lower access levels of commmunications systems. The use of priorities greater than 100 is likely to degrade system control and telephony functions.

A primary activity must never be given a priority of 0, because it would then be unable to respond to system events.

4.4.2.2 Registers

D0.L : 0 : entry by START ACTIVITY
1 : entry from Top Level Menu (or one of its subsidiary menus)
2 : auto-entry
3 : entry by START APPLICATION
4 : entry to perform review

(Other values are used for ephemeral processes other than Review. Such details are beyond the scope of this document.)

D1.L : (Review entry only)
Review screen channel identifier (see section 6.3)

D5.L : (START APPLICATION and START ACTIVITY only)
Activity identifier of invoking activity

A2-A4 : (START APPLICATION and START ACTIVITY only)
Values as in the corresponding registers of the invoking activity at the time of invocation

A7(SP) Stack pointer (see section 4.4.2.3)

Other data and address registers and CC: Undefined.

4.4.2.3 Stack and dump area

Each activity must have a 72-byte dump area (see Kernel Specification [1]. Below the dump area is a space which the activity uses as its initial stack space. On entry to the activity, A7(SP) addresses the base of the dump area, and hence

is appropriately set up for accessing the descending-address stack.

Except in certain cases in the intimate system software, stack space is always set up by Director. In the START APPLICATION call, the space requirement is determined from the program header of the target program. In the START ACTIVITY call, the requirement is specified by the caller; the value will often be that returned by a preceding call of LOAD PROGRAM. Director creates an immobile normal segment of size sufficient to contain the stack space plus the 72-byte register dump area. This segment becomes owned and frozen by the new activity. On entry to the activity, the contents of the stack and register dump area are undefined.

When an application is entered to perform a Review, the only assumption that can be made is that 512 bytes below the location addressed by A7(SP) are available for use as a stack. If further space is required, the reviewed application should acquire it (and destroy it) in the standard way.

4.4.2.4 Rank

A primary activity (started by selection from the Top Level Menu (or one of its subsidiary menus), or by Auto-Entry, or by START APPLICATION) is of 'trusted' rank.

A non-primary activity (started by START ACTIVITY) is of 'trusted' or 'untrusted' rank, determined by the caller.

The activity in which Review (or other ephemeral) invocation occurs is of 'trusted' rank.

4.4.2.5 Events

On entry to a primary activity, bits corresponding to the following events are already set in the activity's Event Request Register:

Foreground Allocated
Suspend Foreground
Terminate
Abandon
Foreground Mode

One or more of these events may already have occurred. A primary activity should never clear these events from its Event Request Register.

On entry to a non-primary activity, the Event Request Register is clear.

On entry to an application for review (or other ephemeral process), the bit corresponding to the Terminate event is set in

the activity's Event Request Register. The application should not clear this event from the Register.

4.5 Application termination

An application may either terminate normally or be abandoned.

An application terminates normally under the following circumstances:

1. It naturally completes its work, according to its specification.

2. It is requested to terminate via its normal control interface with the end user. All visible extended applications should provide such an interface. A transient application may also do so, or may rely on the effect of the START/RESUME keys (see below).

3. The end user presses START or RESUME while the application is in foreground mode and its style (see section 5.2) is transient. In this case a Terminate event is caused in the primary activity of the application.

The application should proceed as follows in normal termination:

1. If it is in foreground mode (suspended or unsuspended), the primary activity should call RELEASE FOREGROUND as soon as any necessary screen interaction is complete. Following a Terminate event it must do this at once.

2. It should complete its scheduled tasks according to its specification. This may take some time if access to slow devices is required.

3. It should release locks and resources as appropriate.

4. The primary activity should ensure that any secondary activities are destroyed.

5. The primary activity should call DESTROY APPLICATION. This deletes Director's records of the application invocation, and causes an implicit call on the Kernel procedure DESTROY THIS ACTIVITY. This Kernel procedure should never be called directly by a primary activity. Any programs loaded into RAM that are being used by an activity are deleted from RAM when the activity is destroyed (unless another activity is using them).

The user can choose to abandon an application using facilities associated with the Store Report feature (see [2]). In this case, an Abandon event is caused in the primary activity. The application should proceed in the same manner as for normal termination, except that it should not complete its scheduled

tasks. Instead it should do the minimum necessary to leave data in a consistent state, for example, by destroying a partially written file.

Following an Abandon event, the end user is not able to re-invoke the application or move it to the foreground until the termination process is complete. (The status 'Abandoned' appears against the application on the Top Level (or subsidiary) Menu.) The ability to review the application may also be affected: see section 6.3 for details.

An application may also be instructed to abandon via its normal control interface with the end user. Unless the abandon can be achieved very quickly the application should call INHIBIT APPLICATION to cause the 'Abandoned' status to appear in the system menus. Access to the application is then restricted in the same way as if the sequence leading to an Abandon event had occurred.

During normal termination (which may be a long process) the end user may again select the application for foreground mode (a Foreground Mode or Foreground Allocated event occurs) or may Review it, and the application should respond in the normal way. If there are particular technical difficulties in coping with this, the application should call INHIBIT APPLICATION with the parameter value which causes the 'Unavailable' status to appear in the system menus, and restricts access to the application in the same way as in the 'Abandoned' case. The end user is still able to cause an Abandon event in the application, in which case normal termination must be replaced by an abandon sequence. The 'Unavailable' state (unlike the 'Abandoned' state) is reversible, and its use may be appropriate at times other than during termination. INHIBIT APPLICATION should be used with discretion since it locks out the end user.

In the case of an application entered to perform a review (or other ephemeral process), the application invocation ends when it completes naturally and calls EXIT REVIEW, or when it receives a Terminate event, whereat it immediately tidies up and calls EXIT REVIEW. There is no Abandon event in this case, and the procedure INHIBIT APPLICATION is not used. Full details are given in section 6.

Terminate and Abandon events may exceptionally be caused by system software, for example on certain System Errors or capsule manipulations.

4.6 Loading and calling programs

4.6.1 LOADING PROGRAMS

An activity may wish to enter a distinct program within the same activity, or to enter part of its original program or subsequently loaded program as a secondary activity (see section 4.7).

The Director interface LOAD PROGRAM returns a descriptor to a named program, by which that program may be called (see section 4.6.2). If the program is not in ROM, nor already loaded into RAM, options are provided to search for it on file store media.

The LOAD PROGRAM facility is provided to allow an application to be overlaid in RAM or be distributed over different units in a capsule (see Appendix 1), or to allow several applications to share a common library.

A program loaded from a file is placed in an immobile normal segment, which is frozen on behalf of the activity requesting it. While a given program remains in ROM or RAM, new users gain access to that same instance of the program. A program loaded into RAM is deleted when it no longer has any users.

LOAD PROGRAM does not create a stack. It returns the size of stack required by the loaded program, which may be used as input to START ACTIVITY.

System software maintains a record of which activities are using which programs. This is for two reasons: so that RAM can be released when there are no longer any users of a program loaded into RAM, and so that, on any future hardware which supports insertion and removal of capsules with the power on, the effects of capsule exchange can be controlled. LOAD PROGRAM registers the calling activity as a user of the nominated program. When the activity no longer requires the program, it should de-register it by a call on RELEASE PROGRAM. De-registration occurs implicitly when an activity dies. The program in which execution commences in a new activity is implicitly registered to that activity. LOAD PROGRAM provides an option whereby an activity can register a program identified by a descriptor instead of by name; the descriptor may have been passed from another activity that has already loaded the program. START APPLICATION registers the target program on behalf of the new primary activity, and de-registers it on behalf of the caller of START APPLICATION. Registration has no effect if the program in question is already registered.

A program written for OPD can be marked as being suitable for entry as an application, or suitable only for loading within an application. In the latter case the program will not appear in the user's Application Menu. LOAD PROGRAM will return an error response if the target program has the 'application' property, and if the target program is already running as an application, LOAD PROGRAM will return its primary activity identifier.

LOAD PROGRAM cannot be used to load a non-machine code program, but it will nevertheless return the primary activity identifier if the target program is a running application.

4.6.2 CALLING PROGRAMS

Calls within a program can be made without difficulty using conventional methods. Calls between different programs are complicated by the possibility that, in certain environments, programs (and data in the program area) may be in paged address space. The system and the application program have to co-operate to ensure that, whenever a particular activity is running, the correct page (if any is required) is selected for the code being executed and the data being accessed.

The complexity is largely handled by the system software, provided that the application code obeys the rules set out in this section.

When a program is to be called, LOAD PROGRAM is used to obtain a descriptor to the target program. A descriptor is a 32-bit value that specifies both the address of the entry point to the program and its page requirements. Details of the descriptor format will be found in the version of the Kernel specification [1] applicable to paged environments. Normal applications should not need to know the format, and should not seek to modify a descriptor (except possibly to add a small even displacement if entry at an offset from the normal entry point is required).

To call the program, the descriptor is loaded into one of the address registers A0 to A6, and the instruction JSR (An) is executed. No other instruction or operand form is allowed. This causes entry to the target program with the activity's page requirements modified as appropriate. A 32-bit link is stacked in a special format. Details of the link format will be found in the version of the Kernel specification [1] applicable to paged environments, but normal applications need only know that execution of an RTS instruction will return control to the caller in the expected way, with the activity's page requirements restored to those of the caller. Return by RTS is the only method allowed for normal programs.

When a program is called, it will often be necessary to pass to it the address of data which may be in the caller's program area, and hence in the caller's page. No means is provided of executing in one page and accessing data in a different page, and an attempt to call from one page to another will normally fail: the effect of the JSR (An) instruction will be exactly as if a MOVEQ #ERR.PE,D0 had been executed instead. A callable program should, by convention, return a response value in D0.L (and set the Condition Code Register accordingly) so that the caller may reliably detect the error PE (page error) condition.

This error does not arise if the programs are in the same page, or if at least one of them is in unpaged store. The error can also be inhibited by asserting the 'forced page switch' property for the called program (a zero value in bit 5 of byte 25 of the program header: see Appendix 1); this property should be asserted

if the called program does not access data passed by the caller (except in the registers), or can reasonably require that such data be in unpaged store (for example, in RAM). If the property cannot justifiably be asserted, then the user documentation of the program must explain that it and any program that calls it cannot both be plugged into paged ROM slots.

Calls on system programs (such as Telephone Directory) using the TRAP mechanism can in principle fail with error PE for the reason described above. In practice, such programs will usually be in unpaged store or will have the 'forced page switch' property asserted. Calls on Kernel and Application Handler procedures cannot fail in this way (although certain Kernel procedures can return error PE as a result of an invalid attempt to manipulate pages).

A called program should not return an address of data in its own program area, which might be paged, and hence not easily (or at all) accessible to the calling program. [System-oriented programs will be able to use the Kernel procedure SET ACTIVITY'S DATA PAGE to handle return of such addresses in restricted circumstances. If a data page is set by this means, it should be cancelled before any inter-program call or return. This technique cannot conveniently be exploited by programs that are required also to run on early releases of the system software.]

Simple addresses of data in the program area, constructed for example by the LEA instruction, can be used within that program and passed to called programs subject to the rules above. They should not be passed to another activity, where they might be used in a page context other than that required.

The system RAM, visible above Kernel as segments, is not paged, and is not constrained by the rules above.

4.7 Secondary activity invocation

The procedure START ACTIVITY is used to start execution in a new activity (secondary or untrusted) of code forming part of the invoking program or of a program that the invoker has previously loaded. The initial state of the new activity is defined in section 4.4.2.

The code entry point at which execution is to start can be specified in one of two ways:

1. By a descriptor returned by LOAD PROGRAM. The descriptor defines the page (if any) required by the new activity.

2. As the address of an instruction within the current program. If the address is in paged address space, the new activity's page is set to that of the invoking activity.

START ACTIVITY creates a stack for the new activity and freezes it

on behalf of the new activity. If the code entry point to the new activity is in RAM, the segment containing the code is frozen on behalf of the new activity; otherwise the new activity is registered as a user of the capsule containing the code. The code segment or capsule is not de-registered in the invoking activity; if this is required, RELEASE PROGRAM should be used.

The Kernel procedure CREATE NEW ACTIVITY should be used directly only by intimate system software, in circumstances which are outside the scope of this document.

4.8 Asynchronous loading

A call to START APPLICATION or LOAD PROGRAM may not be completed synchronously because of device or other long system actions. If the call cannot be satisfied or failed at once, it will return with response NC, and the bit corresponding to the 'Loading Complete' global event will have been set in the activity's Event Request Register. When the requested action has been completed (successfully or not), that global event will be signalled. The activity can then repeat the request, and will normally then receive a success response or a failure response other than NC (although a further NC response may occur if the Loading Complete event referred to some other load request).

An outstanding request can be cancelled by use of RELEASE PROGRAM. This does not clear the Loading Complete bit from the Event Request Register. The caller should in all cases explicitly clear this bit when he has no further load requests outstanding, by means of the Kernel procedure CLEAR EVENT REQUESTS.

4.9 Interfaces

4.9.1 START APPLICATION

Trap Name: T.DIRECTOR
Action Value (D0.B): D.STAPP

Additional Call Parameters:

A0 : address of buffer containing 12-byte name of target program
D1.W : chaining control:
- 0 new application is to start in background mode
- 1 new application is to take over the foreground
- 2 new or already running application is to take over the foreground (see section 4.4)

The value 4 added to any of the above values indicates that this is a 'tele-start' request (see section 4.4)

D2.W : initial priority of new primary activity (1 to 127)
D4.W : specifies the devices to be searched if the program is not found in ROM or already loaded into RAM. The following (binary) values are allowed:

0 search no devices
2 search the 'left microdrive'
4 search the 'right microdrive'
6 search all available file storage devices
The parameter can alternatively be set to the less significant word of the value returned in D4.L by a previous successful call on START APPLICATION or LOAD PROGRAM. This requests a search of the device (if any) on which the previous program was found.

Return Parameters:

D0.L : activity identifier of primary activity of new application

D2.L activity identifier of primary activity of existing application. The value of this parameter is defined only when error IU is returned in D0 (See description of error IU.)

D4.L : load source
bits 0 to 2 may contain the following values:
1 target program is in ROM
2 target program was loaded from 'left microdrive'
4 target program was loaded from 'right microdrive'
6 target program was loaded from some other device. This value is provided for forwards compatibility. Its precise significance is not yet defined, but the value returned in D4.W can be used compatibly as a call parameter to a subsequent call of START APPLICATION or LOAD PROGRAM.
bits 3 to 31 are not yet defined, but should not be assumed to be zero.

Error Returns:

BP : bad parameter

NF : program not found (this response can also occur in the case of a non-machine code application if the system program that handles entry to such applications cannot be found)

NA : calling activity is not allowed to do this (attempt to transfer foreground by non-primary activity)

DT : director tables full

NP : target program unsuitable for this operation (attempt to tele-start a non-tele-startable application; untrusted machine code program; application is in Abandoned or Unavailable state; program does not have 'application' property)

OM : out of memory

NO : attempt to transfer foreground (D1.W=1 or 2) but caller does not have foreground or has been requested to release it

IU : target program already running as an application. If D1.W was 0 or 1 (or 4 or 5), the target program is unaffected and no implicit RELEASE FOREGROUND occurs in

the invoking activity. If D1.W was 2 (or 6), foreground is transferred to the target application, and RELEASE FOREGROUND occurs in the invoking activity.
Response IU can also result from a failure to open a program file because it is already open for writing. In this case, zero is returned in D2 and no RELEASE FOREGROUND occurs

UF : unformatted or no volume in specified drive
NV : volume removed or unserviceable
PF : file read failure
NC : operation not complete (see section 4.8)

This call invokes the nominated program as an application.

If the program is not in ROM or already loaded into RAM, it is sought on the specified device(s) (if any). If a search of all devices is requested the devices are searched in the same order as for a Kernel OPEN CHANNEL call (see [1]).

If the program is not already running and is a non-machine code or trusted machine code program, it is invoked as a new application. The values in the caller's A2 to A4 are transmitted to the new application.

If transfer of foreground is requested (which can only be by a primary activity that currently has the foreground) the new application is moved to the foreground, and RELEASE FOREGROUND (see section 5) implicitly occurs in the calling activity.

If the target program is already running as an application, the call fails except in the case D1.W=2 or 6 (see description of error IU). In the latter case, the foreground is transferred, but not the registers A2 to A4.

4.9.2 START ACTIVITY

Trap Name: T.DIRECTOR
Action Value (D0.B): D.STACT

Additional Call Parameters:

A0 : entry point to new activity: the permitted values are defined in section 4.7.
D1.W : rank of new activity (1 = untrusted, 2 = trusted)
D2.W : initial priority of new activity (0 to 127)
D3.L : size of stack space to be provided (number of bytes)

Return Parameters:

D0.L : activity identifier of new activity

Error Returns:

BP : bad parameter

OM : out of memory
NA : invalid rank specified
NS : invalid entry point specified

This call creates a new activity and associated stack, and enters the activity at the specified entry point.

4.9.3 LOAD PROGRAM

Trap Name : T.DIRECTOR
Action Value (D0.B): D.LOADPR

Additional Call Parameters:

A0 : address of buffer containing 12-byte name of target program
A1 : descriptor to program to be registered to calling activity
D4.W: specifies the devices to be searched, in the same way as in section 4.9.1.

In addition, bit 6 specifies the function of this call:

= 0 : the name is specified by A0; A1 is ignored

= 1 : the name is specified by A1; A0 is ignored

Note: early releases of Director assume that bit 6 is zero. An application that is to run on all versions of the system software should supply both A0 and A1 when bit 6 = 1.

Return Parameters:

A0 : descriptor to target program (see section 4.6)
D1.L : 1 = target program is untrusted, 2 = trusted
D2.L : activity identifier of primary activity executing target program. The value of this parameter is defined only if error IU is returned, indicating that the target program is an application which is already running. If the target program is an application which is not running, error NP is returned

D3.L : stack space requirement of target program (number of bytes)

D4.L : load source (see section 4.9.1)

Error Returns:

BP : bad parameter
NF : program not found
DT : director tables full
NP : target program unsuitable for this operation (program has 'application' property; program is not machine code)

OM : out of memory
IU : target program already running as an application (see description of return parameter D2.L).
Response IU can also result from a failure to open a program file because it is already open for writing. In this case, zero is returned in D2.
UF : unformatted or no volume specified drive
NV : volume removed or unserviceable
PF : file read failure
NC : operation not complete (see section 4.8)

The function of this call depends on the value of bit 6 of D4.

If bit 6 is zero, the call searches for the nominated program in the same way as START APPLICATION (see section 4.9.1). If the call is successful, it returns a descriptor to the program, which may be used subsequently to call the program (see section 4.6). The calling activity is registered as a user of the program.

If bit 6 is 1, the call registers the calling activity as a user of the program identified by the descriptor (which will have been passed from another activity that already has the program registered).

4.9.4 RELEASE PROGRAM

Trap Name : T.DIRECTOR
Action Value (D0.B): D.RELPR

Additional Call Parameters:

A0 : address of buffer containing 12-byte name of target program
A1 : descriptor to target program (returned by a call on LOAD PROGRAM)
D4.W: bit 6 specifies which parameter identifies the target program:

= 0 : the name is specified by A0; A1 is ignored

= 1 : the descriptor is specified in A1; A0 is ignored

bits 0 to 5 and 7 to 15 should be set to zero

Note: early releases of Director assume that bit 6 is zero. An application that is to run on all versions of the system software should supply both A0 and A1 when bit 6 = 1.

Error Returns:
None

This call de-registers the calling activity as a user of the specified program, as described in section 4.6.

4.9.5 DESTROY APPLICATION

Trap Name: T.DIRECTOR
Action Value (DO.B): D.XAPP

Error Returns:

NA : caller is not a primary activity
Unless error NA is returned, no return is made.

This call deletes Director's record of the application invocation, and destroys the primary activity that makes the call.

4.9.6 INHIBIT APPLICATION

Trap Name: T. DIRECTOR
Action Value(DO.B): D. INHAPP

Additional Call Parameter:

D1.W : bit 0 : 1 put application into 'Abandoned' state
0 do not
bit 1 : (actioned only if bit 0 is 0 and application is not already in 'Abandoned' state)
1 put application into 'Unavailable' state
0 remove application from 'Unavailable' state
bit 2 : (independant of bits 0 and 1)
1 inhibit entry to application for Review
0 allow entry to application for Review
(see section 6 for details of when Review may occur)
bit 3 to 15 : reserved (zero)

Return Parameters:

DO.L : foreground state (see section 5.4)

Error Returns:

NA : caller is not a primary activity

This call sets the calling application into the specified state. In the 'Abandoned' or 'Unavailable' state, the application cannot be moved to the foreground by the end user. The 'Unavailable' state is reversible. If the 'Abandoned' or 'Unavailable' state is requested, the application's foreground phase (if any) is ended by an implicit call of RELEASE FOREGROUND.

4.9.7 GIVE SYSTEM VERSION NUMBERS

Trap Name : T.DIRECTOR

Action Value (DO.B) : D.GIVVER

Additional Call Parameters:

D1.L : this register is reserved for future use; at present it should be set to zero

Return Parameters:

D1.L : base software version number

D2.L : base software variant identifier

D3.W : telephony module firmware version number

D4.L : undefined

Error Returns:

BP : call is not supported by current version of system software. This error will be returned by early releases of the software.

This call returns information about the version of software and firmware in use.

The generic version number of the base software is returned in D1.L as the four graphic ASCII characters supplied when the system was built, for example "F1.2". (Note that, in development builds, the value is usually set to binary zero.)

The variant identifier returned in D2.L also consists of four graphic ASCII characters supplied when the system was built. This identifier may be used to identify variants of the generic base software for use in particular territories etc..

The telephony module firmware version number is returned as two binary numbers in the bytes of D3.W. For example version 2.1 is returned as $0201.

The more significant word of D3.L and the whole of D4.L are returned with undefined contents. The values may be defined in later releases.

The significance of the version numbers, and the use that may be made of them, are outside the scope of this document.

4.9.8 GIVE PROGRAM HEADER FIELD

Trap Name : T.DIRECTOR

Action Value (DO.B) : D.GIVPROG

Additional Call Parameters:

D1.B : identifier of required field

D2.W : length of buffer addressed by A2

D4.W : bit 6 specifies which parameter identifies the target program:

= 0 : the name is specified by A0; A1 is ignored

= 1 : the descriptor is specified in A1; A0 is ignored

A0 : address of buffer containing 12-byte name of target program

A1 : descriptor to target program (returned by a call on LOAD PROGRAM)

A2 : address of buffer (in RAM) into which the program header field is to be copied

Return Parameters:

None

Error Returns:

NP : the specified program is not in ROM nor currently loaded into RAM

NF : the specified program does not have a field with the specified identifier

BP : call is not supported by current version of system software. This error will be returned by early releases of the software

This call returns one of the optional information fields from the program header of a specified program. The format of these fields is defined in Appendix 1.

The program must either be in ROM or currently loaded into RAM. It need not be in the same page as the caller of this interface.

The information returned into the buffer addressed by A2 consists of one byte containing the length (in bytes) of the specified field as it appears in the program header, followed by that number of data bytes copied from the header. If the supplied buffer is too large, the spare bytes are unchanged. If the supplied buffer is too small, the excess bytes are not returned; the length byte indicates the number that could have been returned (and no other warning of truncation is given).

5. SCREEN/KEYBOARD HANDLING

5.1 Introduction

This section describes the standards and interfaces by which applications conventionally access the screen and keyboard in a manner that correctly supports the OPD functional architecture. The objectives are as far as possible to conceal the complexities of the architecture, to simplify the writing of applications, and to encourage a consistent approach across the range of applications as perceived by the end user.

5.2 Architectural Overview

OPD can perform several applications simultaneously, subject to resource constraints. An application that needs a resource that is currently unavailable must stop (or not start), or wait until the resource becomes available, possibly meanwhile performing other work not dependent on the resource.

The most critical resource is the screen. Only one application can be using the screen at any time: this is called the foreground application while it retains the screen, and is said to be running in foreground mode. Any other running applications are for the time being background applications running in background mode. Since the user's perception of what the OPD is doing is principally through the screen, he can always choose which application is to be in the foreground, and his choice is never pre-empted (except on a System Error). The choice is made through the START and RESUME keys, as described in the User Specification [2].

For ergonomic reasons, the foreground application is also the (only) application to which the application keyboard can be currently connected. The application keyboard is that part of the keyboard that can be read by an application running under Director, and comprises the whole keyboard excluding the dedicated OPD system keys, the dedicated telephony keys, and, while the OPD is in a dial state, the keys used for telephone dialling. Details of the sub-allocation of the keyboard are given in the Kernel specification [1]. The application itself does not need to be aware of the current suballocation of the keyboard, but the application designer must take this into account when designing the user interface.

Within an instance of an application, several user-visible "tasks" can be run simultaneously, and the screen and keyboard may be shared between them. This subdivision is the responsibility of the application; as far as the interaction with Director and the end user's control of applications is concerned, it is a single application. This approach is not in general recommended for OPD applications, since it involves the user in two levels of control.

The user chooses which application is to be in the foreground by means of the START and RESUME keys. The design intention of these

keys is to enable the user to select a new foreground application (START) or to return to what he was doing previously (RESUME). The effect of these keys on the current foreground application depends on the application. To achieve a measure of consistency across the range of applications, the OPD philosophy is that most applications should fall into one of two styles:

1. Transient applications. Such applications are typified by short access to the screen to perform single updates or enquiries on information in the OPD, and are likely to be invoked at unpredictable times while longer applications are running. When the user selects a new foreground application by START or RESUME, the assumption is that he has finished with the transient application, which accordingly terminates.

2. Extended applications. Such applications are typified by lengthy user interaction via screen and keyboard. The user is likely to want to break off to perform other urgent tasks while retaining the ability to return to the extended application. When the user selects a new foreground application by START or RESUME, the extended application will go into background mode, relinquishing the screen and keyboard until it is again selected as the foreground application, but if appropriate still accessing other devices (such as the modem).

The terms 'transient' and 'extended' are part of the user view, and can be used in user documentation of applications. The terms describe broad styles, and cannot encompass every case. There may be applications that are sometimes transient and sometimes extended (e.g. a Viewdata application transiently handling stored pages or handling an extended connection to a Viewdata service).

A further broad class of application is the spasmodic application. Such an application is typically quiescent, waiting for the occurrence of an event of interest to it, for example, the start or end of a phone call, the arrival of an electronic message, or a preselected time being reached. The application may then wish to interact with the end user, and will become temporarily like a transient, or more probably, extended application, before returning to its quiescent state.

A general rule applicable to all applications (except those marked as 'invisible'), and in particular to spasmodic applications, is that they must present a display when selected as the foreground application. Even if no user interaction is relevant at the time, the application may allow the user to change its control parameters or at least display a status screen. (An application can become 'non-interruptible' during a termination or abandon sequence: see section 4.)

The foreground application's use of the screen may also be interrupted by use of the REVIEW key or one of the built-in ephemeral functions which acts like REVIEW, e.g. SHIFT/RECALL and

SHIFT/REDIAL. Part of the definition of the function of these keys is that they have no permanent effect on the application whose foreground phase is interrupted by their use. The foreground application, whether transient or extended, will therefore go into background mode, but will continue normally when (and if) the foreground is restored at the end of the Review sequence.

As well as expecting to be interrupted by a Review sequence, each application must also normally be prepared to be reviewed itself. This aspect is covered in greater detail in section 6.

5.3 Mechanisms of foreground control

The logic necessary to give a correct and consistent response to the occurrences which influence the allocation of the foreground is complex. The approach in the OPD system is that such occurrences are seen directly only by Director. Director then causes appropriate events in the affected applications, some of which must be acknowledged by the application making a call on Director. The events are caused in a sequence which ensures a correct and consistent response to the end user's commands.

The events are caused only in the primary activity of the application. Acknowledgements, where required, are also made by the primary activity. It will usually be most convenient if screen handling is peformed by the primary activity. (See section 5.5 for recommendations concerning keyboard handling.)

A period during which an application potentially uses the screen is called a foreground phase. The application indicates that it wishes to enter a foreground phase by calling REQUEST FOREGROUND. A parameter to this procedure specifies whether the request is Active or Passive. With an Active request, the application is actively soliciting the end user's attention, perhaps because of some external occurrence which the end user may not have foreseen, such as unsolicited communications traffic. The application will normally output an Event Report to the Noticeboard (see section 8) when it makes an Active foreground request (if it does not immediately acquire the foreground). With a Passive request, the application is merely indicating that it has a screen to display when the user selects it as the foreground application. An Active request causes the status 'Attention' to appear against the application on the Top Level Menu (or its subsidiary menus); a Passive request causes a 'Waiting' status.

Following any successful call of REQUEST FOREGROUND, the Foreground Allocated event will occur when (and if) the user selects the application for foreground mode. The event will occur within the REQUEST FOREGROUND call if the application is already entitled to use the foreground.

The user may select an application for foreground mode when there is no outstanding foreground request from the application. In

particular this happens when a new application is started through the Top Level Menu. It can also happen when an application is chained by START APPLICATION, or when a spasmodic application is quiescent, or when the foreground has been released and has not been inhibited by INHIBIT APPLICATION and an Abandon event has not been caused. A Foreground Mode event occurs in the application, which must immediately respond by calling REQUEST FOREGROUND to enter a foreground phase. However the application is still not entitled to use the screen and keyboard until the Foreground Allocated event occurs. Exceptionally, the invitation implied by the Foreground Mode event can be refused by means of a call on RELEASE FOREGROUND (or on one of the procedures that performs RELEASE FOREGROUND implicitly); this technique should be used only to cope with an event that arrives during a brief time when the application has not yet made an appropriate call on INHIBIT APPLICATION.

Once the foreground is allocated, the application is entitled to write to the screen. The application can have screen channels open at any time, but it is required not to use them in any way that causes a display unless the foreground is allocated to it. The primary activity must liaise with any secondary activities to ensure that the application as a whole obeys the screen access rules. Any untrusted activity can be unsuspended (given a priority other than zero) only during a foreground phase, unless it is known that it obeys the rules; this is the responsibility of its owning trusted activity.

When the application has completed its foreground phase, it calls RELEASE FOREGROUND. Before this call, the application must have ceased issuing screen transfers, completed any outstanding transfers, and suspended any untrusted activities as appropriate. RELEASE FOREGROUND occurs implicitly when START APPLICATION is successfully used to "chain" a further application, and in certain other cases defined in sections 4 and 5.

Before the application naturally completes its foreground phase, it may be instructed to yield the foreground to some other application as the result of the end user pressing a system control key. The effect on the current foreground application of pressing START or RESUME depends on the current processing style of the application: Transient or Extended, as described in section 5.2. A transient application will terminate, whereas an extended application will suspend screen processing until it is restored to the foreground.

The application declares its style as Transient or Extended in the call on REQUEST FOREGROUND. The application can change its style from time to time if appropriate by further calls of REQUEST FOREGROUND. The declared style has no significance except during a foreground phase.

There are other keys that always imply suspension of screen processing, irrespective of style (e.g. REVIEW), and there may be system occurrences that imply termination, irrespective of style.

The application does not directly see these special keys and occurrences. Instead it sees the Suspend Foreground and Terminate events.

The Terminate event may occur at any time during the execution of an application. The response of the application to this event is described in section 4.5.

The Suspend Foreground event can occur only while the foreground is allocated to the application. The application must immediately stop initiating screen transfers and writing to the screen store (in all its activities). It must suspend any untrusted activities in the application that are liable to be accessing the screen. Around the time of this event, screen transfers are liable to be cancelled by Kernel (this can in theory occur at any time). The application must ensure that it is capable of restoring the screen display when the foreground is eventually restored.
Purpose-written OPD applications must be able to do this using their stored data. Where untrusted activities are involved, the only way of restoring the screen may be to copy the screen store to a screen save area. The 30Kb save area is reserved by Director (when necessary) during the call on REQUEST FOREGROUND If sufficient space is not available, the foreground request is rejected. For sizing purposes, the store usage of such an application is 30Kb greater than the store it directly uses.

When the application has completed all the actions necessary to suspend its use of the foreground, it must call SUSPEND FOREGROUND. (The next user of the foreground will now be allowed to proceed.)

While the foreground is suspended, the application can continue to perform other tasks, such as accessing the modem, provided it does not access the screen (nor write to the screen store). It is for the application designer to decide what processing is appropriate while the user is not interacting with it.

While the foreground is suspended, the Active/Passive choice associated with the original foreground request determines the classification of the application in the Top Level and Resume Menus.

When (and if) the foreground is eventually restored to the application, the Foreground Allocated event occurs again. The application must immediately restore the screen display. It can then resume normal foreground phase processing. If a screen save area is in use, the display will already have been restored by Director.

During the foreground phase suspension the application may be reviewed. Details of review handling are given in section 6.

Depending on the speed and sequence in which the user types system control keys, more than one of the system events may appear to the

application to occur simultaneously. The application should act on events in the following order of priority, ignoring any 'simultaneous' event of lower priority: (event numbers are given in parentheses):

Abandon	(5)
Terminate	(4)
Suspend Foreground	(3)
Foreground Allocated	(2)
Foreground Mode	(6)

When the Foreground Allocated event first occurs in a foreground phase, the hardware display mode of the screen is set as specified in the call of REQUEST FOREGROUND that introduced the foreground phase. On subsequent occasions in the foreground phase when the foreground is restored (after having been suspended), the hardware display mode is restored to its state at the corresponding suspension. In either case, the entire application screen will have been cleared to black (unless the display is automatically restored from a screen save area). The foreground interactions with Director have no effect on the application's screen channel(s).

5.4 Foreground state response

The procedures that manipulate the foreground return a standard response in DO.L when the call is successful. This is defined as follows:

bit 0 : Set to 1 if there is a current (satisfied or unsatisfied) foreground request

bit 1 : Set to 1 if foreground is currently allocated to the application (whether or not there is a current foreground request)

bit 2 : Set to 1 if a request to suspend the foreground is outstanding

bit 3 : Set to 1 if a request to terminate is outstanding

bit 4 : Set to 1 if application state is 'Abandoned'

bit 5 : Set to 1 if application state is 'Unavailable'

bits 6 to 30 : undefined

bit 31 : zero

At most one of bits 2 to 5 is set.

This response is intended to be used as a 'reminder' in applications of modular structure, and as a debugging aid during application development. It is not intended to provide an alternative mechanism for manipulating the foreground: the

procedures and events previously described should be used.

5.5 Keyboard handling

While the application is in unsuspended foreground mode, it is entitled to read the application keyboard (i.e. those keys that are not dedicated to telephony, or to OPD system control, or temporarily being used for telephone dialling).

The application in this mode is permitted to call the Kernel procedure SELECT NORMAL KEYBOARD CHANNEL (see [1]) to select an open keyboard channel as the Current Normal Keyboard Channel, or to open a keyboard channel using the Select option. Thereafter, the codes corresponding to keys pressed will be fed into the buffer associated with that channel, from whence they may be read by conventional Kernel I/O calls. This will continue until

(a) the channel is closed, or
(b) a new channel is selected by the foreground application, or
(c) a system control key is depressed implying the suspension or termination of the current foreground phase.

Following a system control key, key depressions are seen only by the system software until a new foreground user is established. The application that receives the Foregound Allocated event is then entitled to make a keyboard channel current and receive subsequent key depressions.

A keyboard channel can be read using Kernel I/O calls even if it is not the Current Normal Keyboard Channel. No new characters will be arriving in the channel's buffer, but the application can successfully read characters which had been typed, but not processed, before the system control key was pressed. Thereafter, by suitable use of I/O events, the activity can wait until the channel is again selected and characters start arriving again. Thus, for example, one part of the application could be responsible for reading the keyboard, while another part ensures that the keyboard channel is reselected on each entry or re-entry to foreground mode. It is, however, for the application designer to decide whether such offline processing of keys is sensible, or if the keys should be ignored.

The system control and telephony keys are not visible via normal keyboard channels.

The handling of the keyboard by an application entered to perform a Review (or other ephemeral process) is discussed in section 6.4.

When an application is entered via the Top Level Menu, (or one of its subsidiary menus), the expert user may have typed the keys that perform selection from the first application-specific menu(s) in quick succession, before the application is sufficiently active to read them directly. These key

depressions (up to a small number) are stored by Director and can be read by the procedure READ MENU KEY. The action of the application when the Foreground Allocated event occurs should first be to select its own keyboard channel as the Current channel. The application should then read keys using READ MENU KEY, reverting to its own keyboard channel when READ MENU KEY reports no characters available (or possibly when a point is reached at which the user could not sensibly have pressed the keys in advance). However, when the foreground is granted to an application that was already running, it is assumed that the user cannot sensibly press keys until he sees where he had got to; Director will have ignored any keys pressed before the application selected a Current Normal Keyboard Channel, and READ MENU KEY will not return any keys.

Such stored keys are lost when the foreground is suspended or released.

READ MENU KEY will return a Buffer Overflow (BO) response if more keys were pressed than could be stored. This response indicates that keys read from the application's keyboard channel are not consecutive with those obtained by READ MENU KEY; the application may wish to flush its own keyboard channel so that the user may make a clean restart.

5.6 Interfaces

5.6.1 REQUEST FOREGROUND

Trap Name: T.DIRECTOR
Action Value (D0.B): D.REQFG

Additional Call Parameters:

D1.W : bit 0 : 0 application can restore screen after suspension
1 application cannot (Director will allocate store to contain copy of screen image)
bit 1 : 0 request is Passive (see section 5.3)
1 request is Active
bit 2 : 0 application style is Transient (see section 5.2)
1 application style is Extended
bit 3 : 0 initial hardware display mode is 512-pixel
1 initial hardware display mode is 256-pixel

Return Parameter:

D0.L : foreground state (see section 5.4)

Error Returns:

BP : bad parameter
OM : insufficient memory (for screen image copy)

NA : caller is not a primary activity

This call establishes a requirement for foreground mode on behalf of the application whose primary activity makes the call.

Once the foreground phase has been established (whether or not the foreground is actually allocated), REQUEST FOREGROUND can be used to change the Active/Passive and Transient/Extended values associated with the application. (The screen-restore and display-mode values cannot be changed within the foreground phase.)

5.6.2 RELEASE FOREGROUND

Trap Name: T.DIRECTOR
Action Value (DO.B): D.RELFG

Return Parameter:

DO.L : foreground state (see section 5.4)

Error Returns:

NA : caller is not a primary activity

This call terminates the foreground phase of the application whose primary activity makes the call. If an unsatisfied foreground request is outstanding, the request is cancelled.

5.6.3 SUSPEND FOREGROUND

Trap Name: T.DIRECTOR
Action Value (DO.B): D.SUSPFG

Additional Call Parameter:

D1.W : 1 Review screen for this application is the current screen as copied to screen save area (applicable only if such a save area has been requested)
0 application will construct its own Review screen

Return Parameter:

DO.L : foreground state (see section 5.4)

Error Returns:

BP : bad parameter
NA : caller is not a primary activity

This call acknowledges that the application whose primary activity makes the call has ceased issuing screen transfers and writing to screen store, and will continue thus until foreground is restored.

5.6.4 READ MENU KEY

Trap Name: T.DIRECTOR
Action Value (DO.B): D.READMENUKEY

Return Parameter:

DO.B : byte read from menu key buffer

Error Returns:

NB : no byte available
BO : buffer overflow (see section 5.5)

This call reads an initial application menu selection key which the end user may have pressed before the application was ready to receive it. The intended use of this facility is described in section 5.5. There is no event facility on this interface. If NB is returned, the application should read its (already selected) normal keyboard channel. Buffer overflow (BO) is reported in the circumstances described in section 5.5 after all the stored keys have been read; further calls will yield No byte available (NB).

6. REVIEW

6.1 Purpose

The REVIEW feature allows the user to make an unpremeditated inspection of data associated with an active application or an existing reviewable segment. The REVIEW action has no permanent effect on the application or segment reviewed, nor on any application active when REVIEW is invoked.

The choice of what to display during the Review is up to the application. Typically it will be the current screen of an active application, or the most recently accessed logical item in a segment. It might instead, or as well, be a status report on the area covered by the application. The general objective is to provide the user with as useful a reminder as is possible (in a single display) of what he was last doing with that application. In some cases, there may be several displays which might be appropriate. Wherever possible, the application should make the choice itself; if it cannot reasonably do this, it will have to present a menu to the user (possibly on a screen already containing some basic data). However the use of a complex hierarchy of screens is contrary to the philosophy of review, which should act as a quick aide-memoire, and review capabilities should not be over-engineered. Complex data review should be handled by entering the application formally.

6.2 User Interface

The user can press the REVIEW key at any time once the OPD power-up sequence is complete. The existing foreground user of the screen is suspended from the foreground, and a Review menu is displayed. The user selects from the menu, and then sees the selected application's Review screen. Exceptionally this may be preceded by intermediate menu-screens where there is an unavoidable user choice.

The expert user may press a sequence of Review selection keys in quick succession following REVIEW. The choice screens to which these keys are the response are not then displayed.

The final screen in the Review sequence normally remains displayed only while the user contains to hold down a key on the application keyboard. When the key is released, the application that was interrupted by the Review sequence will be restored to the foreground. (If that application has meanwhile ended its foreground phase, the Top Level Menu will be displayed instead.)

The user may wish to hold the Review display without keeping a key pressed, for example to study the display at length, or to use the keyboard for dialling, or to apply the SHOW or PRINT facilities to the Review display. This can be achieved by making the initial selection from the Review Menu using a function key (f and numberpad-digit) instead of the corresponding ordinary digit.

The Review Menu screen and any intermediate selection screens remain displayed without the need to keep a key depressed.

While the Review Menu or an intermediate selection screen or a 'held' final screen is displayed, the Review sequence can be terminated by one of the following system keys:

RESUME	the application or system menu that was interrupted by the Review sequence is restored to the foreground (The Top Level Menu is displayed if that application has ended its foreground phase.)
START SHIFT/START	the effect is as if that key had been pressed while the interrupted application or system menu was using the screen
REVIEW	a new review sequence is entered
LIST LAST LOOK	these keys have the same effect as if they had been pressed to interrupt the already interrupted application or system menu
SHOW PRINT	these keys have their normal effect, and operate on the current display of the Review sequence, but a subsequent RESUME will return to the application or system menu interrupted by the review sequence, not to the review sequence itself

Telephony functions are not impaired by Review, provided that the final Review screen is 'held' if the keyboard is required for dialling.

6.3 Handling of Review by applications

Applications are selected to appear on the Review Menu according to the following rules:

1. The application must be 'visible', and must be in ROM, or already loaded into RAM.
2. If it is a non-machine code application, it must already be running.
3. If it is running, it must not have inhibited Review by use of INHIBIT APPLICATION (see section 4.9.6).
4. At least one of the following must apply:
   (a) the application has the 'always reviewable' property (bit 6 in byte 24 of the program header, see Appendix 1)
   (b) the application is running, and is not in the 'Abandoned' or 'Unavailable' state
   (c) there exists a reviewable segment (see below) of the same name as the application

An application may declare to SUSPEND FOREGROUND that its review screen is its current display as copied to the screen save area; such an application will be offered on the Review

Menu (subject to the rules above), but no Review entry to the application will occur.

An application that expects no Review entry may nevertheless be so entered at a moment when it has not established the appropriate status; in this case it should immediately call EXIT REVIEW and establish the status.

A reviewable segment is a permanent segment that contains data that the user may wish to Review, for example a set of stored viewdata screens. It is marked as "reviewable" by the application that owns it, using the segment handling facilities described in section 7. The name of the segment is the same as that of the application that owns it and can Review it. (There is an exception to this rule in the case of OPD system software; this is outside the scope of this document.) It is sensible for the user to ask to Review this data whether or not the corresponding application is currently running. If the application is running at the time of the Review, it may be appropriate to offer the user a choice of reviewing the current action of the application or the stored segment.

(Note that it is irrelevant whether the reviewable segment actually contains the reviewable data. The application could choose to hold the data in another segment. It is merely the existence of the reviewable segment that is used in constructing the Review Menu.)

When a Review sequence occurs, the selected application is entered at its normal entry point. The code executes within a trusted Director activity dedicated to Review processing. The state of the registers etc. is defined in section 4.4.2; in particular the value 4 is set in D0.L to indicate Review processing.

A dedicated screen channel is provided for use by the Review activity (and other ephemeral processes). Its identifier is supplied in D1.L on Review entry to the application. At that time the channel display mode is 80-column; the channel properties are otherwise undefined. The hardware display mode is 512-pixel and the entire application screen is black. [On entry to SHOW and PRINT, the hardware display mode and screen contents are as left by the preceding user of the screen.] On Review entry, the reviewed application is already entitled to use the screen, and the normal actions of requesting the foreground etc are not required (nor allowed). The dedicated screen channel must not be closed.

The application will display any intermediate selection screens, and read the keyboard response to these using the facilities described in section 6.4. When it comes to display the final Review screen (which may be at once), it must first call DECLARE FINAL REVIEW SCREEN to activate the (Director) test on continued key depression.

Having completed display of the final screen, the application must release any resources etc that it has acquired (but not the

dedicated screen channel), and call EXIT REVIEW. In particular it must thaw or destroy any segments that it has frozen or created.

Throughout the Review process, the application must continually look for a Terminate event. In particular it should do this after output of every line or two to the screen. As soon as the Terminate event occurs, the application should tidy up and call EXIT REVIEW as if it had completed its processing.

The application must not destroy the stack space with which it is provided, nor destroy the activity in which it is invoked. It is not allowed to start an untrusted activity, and it should start an additional trusted activity only where this is essential to achieve the desired effect (in which case it must ensure that the additional activity is destroyed at the end of the Review sequence). It is permissible for an application being reviewed to start another application (or even a non-review instance of itself), but it cannot pass the foreground using the 'chaining' facilities of START APPLICATION.

6.4 Use of Keyboard during Review

In principle, the handling of the keyboard during a Review entry to an application is analogous to that during a normal foreground phase: the reviewed application can open a keyboard channel, select it as the Current Normal Keyboard Channel, read any keys already pressed using READ REVIEW KEY (rather than READ MENU KEY), and revert to using its own channel when READ REVIEW KEY reports that no key is available.

In practice, however, most Review sequences will not require any keyboard input, or at most one or two menu selection keys. In such cases the application should not open a keyboard channel, but should use the channel underlying READ REVIEW KEY. To support this usage, READ REVIEW KEY provides a facility whereby a local event can be caused when a key becomes available (in the same way as the Kernel procedure GET BYTE IMMEDIATE).

6.5 Ephemeral System Applications

Certain internal applications forming part of the system software are invoked via the same mechanism as for review of a visible application. These internal applications include those that handle the following dedicated keys:

LIST	(Priority telephone directory)
LAST	(Recent number redial)
SHOW	(Transmit current screen image)
LOOK	(Receive screen image transmitted by SHOW)
PRINT variants	(Output current screen image to printer)

Ephemeral processes (other than Review) should not call DECLARE FINAL REVIEW SCREEN unless the specification of the ephemeral process states that the user can maintain the final display by

continued depression of a key.

The specification of the ephemeral process will also state the circumstances (if any) in which the process will terminate (by calling EXIT REVIEW), other than on receipt of a Terminate event. (The Terminate event is caused by use of any System Control key, or by key release following a call of DECLARE FINAL REVIEW SCREEN.)

On receipt of a Terminate event, the ephemeral process will normally complete processing of any outstanding keys available via READ REVIEW KEY or a specifically opened keyboard channel, but will not perform further displays.

[The above ephemeral processes are not necessarily all provided at first release. An attempt to use an unimplemented ephemeral process will have no effect except to cause the screen to flicker and to terminate Review or other ephemeral process if current.]

6.6 Interfaces

6.6.1 READ REVIEW KEY

Trap Name: T.DIRECTOR
Action Value (D0.B): D.READREV

Additional Call Parameter:

D3.W : event number of a local event that is to be caused when a byte becomes available, if no byte is available immediately. An event number of -1 indicates that no event is to be caused

Return Parameter:

D0.B : byte read from Review key channel

Error Returns:

NA : caller is not Review activity
NB : no byte available (the event, if any, specified in D3 will occur when a byte becomes available.)
BO : buffer overflow (more keys pressed than Director can store)

This call reads the next byte from the keyboard that is intended for the current Review process. The intended use of this facility is described in section 6.4. The significance of Buffer overflow (BO) is as described for READ MENU KEY (see section 5.6.4).

6.6.2 EXIT REVIEW

Trap Name: T.DIRECTOR
Action Value (D0.B): D.EXITREV

Error Returns:

NA : caller is not Review activity
Unless error NA is returned, no return is made.

This call causes exit from a reviewed application (or ephemeral system application).

6.6.3 DECLARE FINAL REVIEW SCREEN

Trap Name: T.DIRECTOR
Action Value (DO.B): D.FINALREV

Error Returns:

NA : caller is not Review activity

This call declares that the application is about to display its final Review screen, and that it does not intend to read any further bytes from the keyboard.

7. SEGMENTS

7.1 Segment Names

The primitive concept of the segment, as an allocation of RAM, is explained in the Kernel specification [1].

Each segment is either permanent or transient. A permanent segment is used if it is required to survive when no application is accessing it. A transient segment ceases to exist when there are no applications accessing it (i.e. when its owning activity dies or deletes the segment and no other activity has it frozen).

A permanent segment has a name. An application that wishes to access a permanent segment that may already exist uses the Director procedure GET SEGMENT IDENTIFIER to discover the segment identifier of the named segment. The segment identifier must not be remembered over a period when the application has not had the segment in use, since the segment could have been saved and subsequently loaded with a different segment identifier. Transient segments do not have names.

The name of a permanent segment is 12 characters long, the characters being chosen from ASCII $20 to $7A inclusive. (System software may use names containing other characters to avoid confusion with user application segments and to achieve special system effects.) To avoid name clashes between segments belonging to different applications, and to make the purpose of the segment apparent to the end user, it is suggested that segment names be derived in some obvious way from the name of the associated application.

A segment is created by the Kernel procedure CREATE NEW SEGMENT (see Kernel specification [1]). A newly created segment is transient. A segment can be made permanent, and possibly subsequently again made transient, by the Director procedure CHANGE SEGMENT PROPERTIES. A permanent segment is destroyed by the Director procedure DESTROY SEGMENT. The Kernel procedures CHANGE SEGMENT OWNERSHIP and DESTROY SEGMENT must not be used for the above purposes, and it is recommended that they be not used at all, all manipulation being effected by the Director procedures.

7.2 Reviewable segments

A permanent segment may be declared to be reviewable by means of the procedure CHANGE SEGMENT PROPERTIES. The significance of the reviewable segment is described in section 6.

7.3 Segment usage

The OPD system software includes a Data Record feature, which enables the user to Save permanent segments to a file, and subsequently to Load those segments from the file. The full details of Data Record and its effect on applications

are outside the scope of this document, but the Director procedures associated with Data Record are described below.

A successful Save cannot be performed while any application is (potentially) writing to a permanent segment (or changing its cell structure). A successful Load cannot be performed while any application is (potentially) writing or reading a permanent segment. To control this, Director maintains a record of activities that have registered themselves as readers or writers of a particular permanent segment.

REQUEST READ ACCESS TO SEGMENT registers the calling activity as a reader of the segment. REQUEST WRITE ACCESS TO SEGMENT registers the calling activity as both a reader and a writer. RELEASE WRITE ACCESS TO SEGMENT deregisters the calling activity as a writer, but retains registration as a reader. RELEASE ACCESS TO SEGMENT deregisters the calling activity both as a writer and as a reader. There is an option in CHANGE SEGMENT PROPERTIES to request registration as a reader or reader and writer.

An application must achieve registration of the appropriate type before accessing a permanent segment. A request may be rejected if the Data Record feature is active at the time.

Note that registration is on an individual activity basis, but where there are cooperating activities (for example, within a single application) it may be sufficient for one of them to register on behalf of them all. Coordination between activities or applications concurrently accessing a segment is their own responsibility.

A permanent segment cannot be destroyed while any activity is registered as a reader or writer of that segment (even if the only registered activity is the one attempting the deletion). Nor can a permanent segment be destroyed if it is being Loaded or Saved by Data Record. Registration thus prevents a permanent segment being destroyed by another application or by the user while it is temporarily thawed.

Registration and de-registration have no effect on the frozen/thawed state of the segment.

There is no automatic de-registration when a registered activity terminates: de-registration must be done explicitly.

7.4 Store Report

The user can request a Store Report, by a facility within the Housekeeping subsystem (see [2]). This report lists the names and sizes of existing permanent segments, and the names of running applications with the total size of the transient segments owned by each (including stack segments and code segments).

The user can choose to destroy a permanent segment listed in the

Store Report or to abandon an application listed therein. He cannot, however, destroy a permanent segment that has any registered reader or writer.

7.5 Interfaces

7.5.1 CHANGE SEGMENT PROPERTIES

Trap Name: T.DIRECTOR
Action Value(DO.B): D.SEGPROPS

Additional Call Parameters:

D1.L : segment identifier
D2.W : new status of segment:
bit 0 : 0=transient, 1=permanent
bit 1 : 0=non-reviewable, 1=reviewable(only allowed if bit 0=1
bit 2 : 1=request read access (only allowed if bit 0=1)
bit 3 : 1=request read and write access (only allowed if bit 0=1)
bits 4 to 15 : reserved (zero)
D3.L : identifier of new owning activity (transient segment only)
A0 : address of buffer containing 12-character name of segment (permanent segment only)

Error Returns:

NA : invalid activity identifier
NS : invalid segment identifier
BP : bad parameter
OM : name table full
EX : another segment already has the specified name
IU : action not possible because of existing read or write registrations
SL : action not possible because of current Data Record actions

This call establishes the specified status for the specified segment, and sets/changes the owning activity identifier or segment name. If any error is reported, no change is made to the properties of a valid segment.

7.5.2 DESTROY SEGMENT

Trap Name: T.DIRECTOR
Action Value (DO.B): D.XSEG

Additional Call Parameter:

D1.L : segment identifier

Error Returns:

NS : invalid segment identifier
IU : the segment is frozen by another activity or there are existing read or write registrations
SL : action not possible because of current Data Record actions
BP : bad parameter

This call destroys the specified segment in the same way as the corresponding Kernel procedure (see [1]).

If the segment is successfully destroyed, the segment name (if any) is also destroyed.

7.5.3 GET SEGMENT IDENTIFIER (OF PERMANENT SEGMENT)

Trap Name: T.DIRECTOR
Action Value (D0.B): D.SEGID

Additional Call Parameter;

A0 : address of buffer containing 12-character segment name

Return Parameters:

D1.L : segment identifier
D2.L : bits 0 to 6 = count of activities registered as writers
bit 7 = 1 if segment is reviewable
bits 8 to 15= count of activities registered as readers
bits 16 to 31 undefined

Error Returns:

NF : no such segment name is registered

SL : segment is not available because Data Record is performing a Load

This call returns the specified values for the nominated segment.

7.5.4 REQUEST READ ACCESS TO SEGMENT

Trap Name: T. DIRECTOR
Action Value(D0.B): D.READSEG

Additional Call Parameters:

D1.L : segment identifier

Error Returns:

NS : invalid segment identifier
SL : segment is not available because Data Record is performing a Load
OM : name table full

This call requests registration of the calling activity as a reader of the identified permanent segment.

7.5.5 REQUEST WRITE ACCESS TO SEGMENT

Trap Name: T.DIRECTOR
Action Value (DO.B) D.WRITESEG

Additional Call Parameters:

D1.L : segment identifier

Error Returns:

NS : invalid segment identifier
SL : segment is not available because Data Record is performing Load or Save
OM : name table full

This call requests registration of the calling activity as a reader and writer of the identified permanent segment.

7.5.6 RELEASE WRITE ACCESS TO SEGMENT

Trap Name: T.DIRECTOR
Action Value (DO.B) D.RELWRITE

Additional Call Parameters:

D1.L : segment identifier

Error Returns:

NS : invalid segment identifier

This call cancels the registration (if any) of the calling activity as a writer of the identified permanent segment. Registration (if any) as a reader is unaffected.

7.5.7 RELEASE ACCESS TO SEGMENT

Trap Name: T.DIRECTOR
Action Value(DO.B): D.RELSEG

Additional Call Parameters:

D1.L : segment identifier

Error Returns:

NS : invalid segment identifier

This call cancels the registration (if any) of the calling activity both as a reader and as a writer of the identified permanent segment.

8. NOTICEBOARD

8.1 Description

The Noticeboard is an area at the bottom of the screen which is used to display status information about OPD and its applications which cannot be accommodated in the main screen area. Most of the Noticeboard is used for telephony status information. The parts that concern ordinary applications are the Flags and the Output Area. The format of the Noticeboard is described in the user Specification [2].

8.2 Flags

Character positions 5 to 14 of the bottom row of the Noticeboard are reserved for flags (the left hand character is position 1). These are (usually) single character mnemonic indicators of aspects of OPD system status or reminders of events that have occurred (such as "electronic message received" or "save store failed").

Each flag is an upper case letter (or, exceptionally, a digit). The display attributes (PAPER and INK) are selected by the application that causes display, and may be set so that the flag is effectively cleared. A given flag position may display different characters and intensities at different times. Flags do not flash. Display is effected by calling the procedure DISPLAY NOTICEBOARD FLAG. After power-up or reset all flags are clear. The flag at a given position remains displayed until it is explicitly changed, whether or not the application that caused its display is still running.

Tones are not generated to accompany changes in flags. Typically a report will be placed in the Output Area accompanied by a tone, and the flag will then be set as a reminder of the condition.

Flag positions and characters are allocated by ICL, and are normally reserved for use by standard system applications provided by or procured by ICL. Other applications must rely on the facility to send a report to the Output Area and to put an application into a state where it is actively soliciting user attention (see section 5.3).

8.3 Output area

The Output Area occupies the first 20 character positions of the upper row of the Noticeboard, and is capable of displaying upper case letters, digits, and other graphic characters which yield a meaningful display at Noticeboard height (see Kernel specification [1]). This set does not include the "curly f" character (function key qualifier).

The purposes of the Output Area are as follows:

1. For an application in background mode, to advise the user of

an incident which may require his attention, such as the receipt of an electronic message or a preset "alarm" time being reached.

2. For an application in foreground mode, to report a similar incident when there is no room on the main screen or when use of the main screen would be confusing to the user.

3. For an application in foreground mode, to display a status report which cannot be accommodated on the main screen, which lasts for a long time in user terms (greater than 1 second, say) and where it may not be obvious to the user what is happening. This might apply during long device or communications actions. It is not necessary for a normal application to report the progress of dialling (manual or auto): this is dealt with by Telephone Handler in the telephony parts of the Noticeboard. The status report facility should not be used unnecessarily since it inhibits display of the real time clock.

4. For system software to display status and event reports.

5. For the display of a real time clock when the Output Area is not needed for a report.

Director provides interfaces designed to resolve, as far as possible, the conflicts that potentially arise for access to the Output Area. There is a Current Status Report associated with the current foreground application. This is null at the start of each new use of the foreground, and can be set, changed and cleared by the current foreground application. It is displayed whenever the Output Area is not required for an Event Report. It must be restored (if non-null) by the application following a suspension of the foreground.

Any application can also specify an Event Report, which may be Urgent or Normal. An Urgent Event Report is one directly related in real time to the current interaction with the end user, and is likely to be generated only by the current foreground application. The report is displayed at once, and persists until

(i) another Urgent Event Report is received, or
(ii) the current use of the foreground ends, or
(iii) the application cancels the report, or
(iv) five seconds have elapsed and there is a non-null Current Status Report or an outstanding Normal Event Report, or
(v) one minute has elapsed, when the clock display is restored

A Normal Event Report is one for which immediacy of display is not critical within a few seconds. It is displayed as soon as any preceding Event or Status Reports have each been displayed for five seconds or have been cancelled by their owning applications. It persists until

(i) an Urgent Event Report is received, or
(ii) the application cancels the report, or
(iii) five seconds have elapsed and there is a non-null Current Status Report or an outstanding Normal Event Report, or START has been pressed since the start of the five second period (enabling the user to recover the real time clock display).

(No compensation is provided for a report deprived of its five-seconds-worth by an Urgent Event Report.)

It will be seen that a Normal Event Report can in theory remain displayed indefinitely. The application must cancel such a report as soon as it no longer applies, and should usually cancel it in any case after a reasonable time (perhaps half a minute), so that the real time clock display is restored.

The display attributes (PAPER and INK) of Output Area reports are selected by the application that causes the display. Normally the defaults provided by DISPLAY NOTICEBOARD REPORT should be used (see section 8.5.2).

An application can specify a sound to be generated in conjunction with an Event Report. The sound is output at the instant that the report is displayed. The sounds are the same as the 'standard sounds' that can be generated directly using the facilities described in section 9. No sounds should normally be required to accompany changes of Current Status Report.

Report cancellation is achieved by calling CANCEL NOTICEBOARD REPORT with (for a Normal Event Report) a parameter returned by Director when the report was initiated.

8.4 Telephony Noticeboard

Special interfaces are provided for Telephone Handler to write to those parts of the Noticeboard reserved for telephony information. These interfaces are outside the scope of this document.

8.5 Interfaces

8.5.1 DISPLAY NOTICEBOARD FLAG

Trap Name: T.DIRECTOR
Action Value (D0.B): D.NBFLAG

Additional Call Parameters:

D1.W : character position for flag display in bottom row (5 to 14)
D2.B : internal code of character to be displayed
D3.W : paper colour for display
D4.W : ink colour for display
(0=black, 2=red, 4=green, 7=white)

Error Returns:

BP : bad parameter

This call causes the display of a flag in the Noticeboard. Display is cancelled by displaying a space or black character on black paper.

8.5.2 DISPLAY NOTICEBOARD REPORT

Trap Name: T.DIRECTOR
Action Value (DO.B): D.NBREPORT

Additional Call Parameters:

A1 : address of buffer containing report. The buffer may contain 1 to 20 graphic characters, interspersed with 0 to 4 PAPER or INK control code sequences (see [1]). At the start of the display, the attributes are black INK on green PAPER for an urgent event report, and green INK on black PAPER for a normal event report or current status report. Colours should be confined to 0 = black, 2 = red , 4 = green , 7 = white
D1.W : length of report (1 to 28)
D2.W : report type:
0 current status report
1 normal event report
2 urgent event report
D3.W : associated sound:
n(>0) standard sound identifier (see section 9.2)
-1 no sound

Return Parameter:

D1.W : event report identifier, for possible use in a call on CANCEL NOTICEBOARD REPORT. Defined only for report type 1 (Value undefined in other cases.)

Error Returns:

BP : bad parameter
OM : event report queue full

This call queues a report for display in the Output Area of the Noticeboard. See section 8.3 for details.

8.5.3 CANCEL NOTICEBOARD REPORT

Trap Name: T.DIRECTOR
Action Value (DO.B): D.NBCANCEL

Additional Call Parameter:

D1.W : event report identifier (normal event report only)
D2.W : report type: 0 = current status report

1 = normal event report
2 = urgent event report

Error Returns:

BP : bad parameter

This call cancels a previously requested event report (deletes it from the screen or removes it from the queue, together with any associated sound).

8.5.4 CONVERT DATE AND TIME

Trap Name: T.DIRECTOR
Action Value (D0.B): D.DATETIME

Additional Call Parameters:

D1.L : time in seconds-format to be converted to readable format
A2 : buffer to receive converted date and time in readable format
D2.W : buffer size (1 to 27). No error is reported if the buffer is less than 27 bytes; the date and time is truncated on the right to the buffer length. If a buffer size greater than 27 is specified, only the first 27 bytes of the buffer are changed.

Error Returns:

BP : bad parameter

This call converts a time in seconds-format (as held by the Real Time Clock device) to the date and time in readable format.

In seconds-format, the time is expressed as a 32-bit unsigned number, being the number of elapsed seconds since the midnight that prefaced 1st January 1970.

In readable format, the date and time are expressed as follows:

bytes 0 to 2	:	day of the week in 3-character form, e.g. WED
byte 3	:	space
bytes 4 to 5	:	day of the month; 2 digits
byte 6	:	space
bytes 7 to 9	:	month in 3-character form, e.g. JAN
byte 10	:	space
bytes 11 to 14	:	year including century; 4 digits
bytes 15 to 18	:	spaces (application may insert "AT")
byte 19 to 20	:	hours on 24-hour clock; 2 digits
byte 21	:	colon
byte 22 to 23	:	minutes; 2 digits
byte 24	:	colon
bytes 25 to 26	:	seconds; 2 digits

Zero suppression is not applied to numeric fields.

Example: TUE 03 JAN 1984 17:30:01

9. SOUND GENERATION

9.1 Description

OPD includes a Tone Generator which can generate a tone of constant frequency 2400/n Hz (n = 1 to 255). There is contention for access to the Tone Generator, in particular between the foreground application, Event Report tones, the keyboard, and the telephony system. For this reason, and to achieve consistency of sounds across applications, Director provides interfaces to allow sounds to be generated in a controlled way.

The sounds that can be generated by a normal application fall into two classes, standard sounds and special sounds, which are described below. (There is a third class of sounds, available only to Telephone Handler, which is outside the scope of this document.)

Applications are allowed to make sounds only by use of the Director procedure MAKE SOUND. They must not directly call the corresponding Kernel procedure.

9.2 Standard sounds

Standard sounds are provided to cater for situations which are common to a wide range of applications. Standard sounds should be used whenever possible, to provide consistency across the range of applications.

A standard sound is identified by the value passed in D1.W to MAKE SOUND. The values have names of the form TG.name, and are defined in the Application Handler INCLUDE file TGVALUES.

The following standard sounds are provided:

TG.BADKEY Invalid key depression. (In some cases it may be appropriate to ignore an invalid key rather than cause a sound.)

TG.TYPE Invitation to type (following a period when keyboard input would have been inappropriate).

TG.EVENT An expected event has occurred (usually associated with a Noticeboard Event Report).

TG.ERROR An unexpected event has occurred (often associated with a Noticeboard Event Report).

TG.CART Request for attention to file storage medium.

Director queues standard (and special) sounds, and outputs them in order of receipt, allowing a suitable gap between them to allow them to be distinguished. If the ergonomics of the application require other actions to be coordinated with the actual output of the sound, the caller can request that a local event be notified at

the start and/or at the end of the end of output of the sound.

If the response from MAKE SOUND indicates that the sound queue is full, the application should normally abandon the attempt to make the sound, since the user will already have more sounds than he can cope with.

Telephone ringing tones are suspended while standard (and special) sounds are output.

9.3 Special sounds

Special sounds can be output when there is no suitable standard sound.

Output of a special sound is caused by passing the value TG.SPECIAL in D1.W to MAKE SOUND.

A special sound is defined by a word-aligned buffer containing a sequence of contiguous two-byte entries defined as follows:

low-address byte:	n = 1 to 255	: sound frequency is 2400/n Hz
	n = 0	: no sound
high-address byte:	n = 1 to 254	: sound is to last for n fiftieths of a second
	n = 255	: sound is to last indefinitely
	n = 0	: sound sequence is to be repeated from start of buffer ad infinitum

The sound generated is the contiguous sequence of tones (or silences) defined by the two-byte entries taken in ascending address order.

The total duration of a special sound should not, without good reason, exceed 2 seconds. After 2 seconds of a special sound have been output, the sound will be aborted (and the end-of-sound event caused) as soon as there is another standard, special or telephone sound awaiting output.

An infinite or repeating special sound can be terminated (following its allocated 2 seconds) by queuing another sound (e.g. a short silence).

The sound buffer must not be modified until output of the sound is complete.

9.4 Interfaces

9.4.1 MAKE SOUND

Trap Name: T.DIRECTOR
Action Value (D0.B): D.SOUND

Additional Call Parameters:

D1.W : required sound (see sections 9.2 and 9.3)
D2.W : length of sound buffer (special sound only)
A1 : address of sound buffer (special sound only)
D3.W : number of a local event to be caused at start of output of sound (-1 if no event required)
D4.W : number of a local event to be caused at end of output of sound (-1 if no event required)

Error Returns:

BP : bad parameter
DT : sound queue full

This procedure requests the output of a sound.

10. NAME TABLE

10.1 Description

Director provides a Name Table, together with procedures for accessing it. The Name Table enables cooperating applications or activities to assign fixed names to objects that have, as far as other system interfaces are concerned, unpredictable values or identifiers.

Each entry in the Name Table describes a notional 'object', and comprises:

- an object name, consisting of 12 characters (recommended to be graphic)
- an object type
- an object subtype
- an object value

An object type may be one applicable to many applications (e.g. a Kernel resource such as a semaphore), or it may be some object private to one or a few applications. The only object types that can be used are those registered in section 10.2 of this document. (An up to date register is maintained by ICL, and new entries will be made on request on behalf of independent software writers.)

There are logically separate name tables for each object type. The names of the objects of a given type must be distinct; the usual convention is to derive the names from those of the applications that 'own' the objects.

The values of the objects of a given type should also be distinct, although Director does not necessarily check this when an object is entered into the name table. The semantics of the object value are up to those who access objects of that type. Usually the semantics will be obvious, for example, the resource identifier for an object type corresponding to a Kernel resource.

The semantics of the object subtype are entirely up to those who access objects of a given type.

Parallel facilities for the naming of permanent segments are described in section 7, and there should be no general requirement to register objects of type 'segment' in the visible Name Table, although particular applications may choose to register private objects related to segments.

10.2 Register of object types

The following object types may legally be used:

(0 reserved for internal use by Director)

1 semaphore

10.3 Interfaces

10.3.1 REGISTER NEW NAME

Trap Name: T.DIRECTOR
Action Value (DO.B): D.NEWNAME

Additional Call Parameters:

A0 : address of buffer containing 12-character name of object
D1.L: object value
D2.L: bits 0 to 7 : object type (see section 10.2)
bits 8 to 15: object sub-type (defined by caller)

Error Returns:

OM : name table full
EX : object of this name and type already exists

This call inserts the specified name and associated values into the Name Table.

10.3.2 FIND NAME

Trap Name: T.DIRECTOR
Action Value(DO.B): D.FINDNAME

Additional Call Parameters:

A0 : address of buffer containing 12-character name of object
D2.B: object type (see section 10.2)

Return Parameters:

D1.L: object value
D2.L: bits 0 to 7 : object type
bits 8 to 15: object subtype
bits 16 to 31: undefined

Error Returns:

NF : no such name of this object type

This call searches the Name Table for the specified name and object type, and returns the associated values.

10.3.3 DESTROY NAME

Trap Name: T.DIRECTOR
Action Value(Do.B): D.XNAME

Additional Call Parameters:

D1.L: object value
D2.B: object type (see section 10.2)

Error Returns:

NF : no name with this type and value

This call deletes from the Name Table the entry with the specified object value and type.

APPENDIX 1. PROGRAM FORMAT IN ROM

A1.1 Introduction

Programs for use on OPD can be stored in Read Only Memory (ROM). The chips containing ROM may be built into the OPD itself, or be part of the Rompack, or be contained in capsules which can be plugged into the Rompack, or be connected via some other physical arrangement.

In all these cases, the data in the ROM has a standard organisational layout, which enables Director to locate the individual programs. (Certain variations in the format apply to ROM containing intimate system software: these are outside the scope of this document.)

A1.2 ROM-unit format

The OPD address space (0 to $FFFFF) contains a number of regions which may be used to store programs. Different regions are not necessarily contiguous. In a hardware environment that supports paging, there may be several regions which occupy the same addresses, each in association with a different page. The effects of paging are discussed in section 4.6.

The physical ROM (in capsules etc.) is mapped into these regions of the address space. The details of this mapping are outside the scope of this document. Indeed the addresses of the regions and the assignment of page numbers may vary on different generations of hardware, and assumptions should not be built into applications (with certain exceptions in the case of system software).

The hardware mapping may cause some items of ROM to appear more than once in the address space. Director therefore ignores duplicate programs, with the consequence that it is not possible to have distinct programs with the same name; only one of the programs will be accessible. (In unpaged environments, early releases of Director may allow distinct programs with the same name. This is not to be exploited or relied upon.)

In certain cases, the hardware mapping is such that read and write accesses to the addresses can be distinguished, enabling the installation of programs that contain their own RAM or that are externally connected to some peripheral device. This requirement is indicated by the setting of a bit in the data format described below.

The data which is to appear in the stored program regions of the address space is organised into data structures called ROM-units, whose format is defined below. A ROM-unit occupies a continuous set of addresses within one of the regions, starting on an even byte boundary. The following types of ROM-unit are "preferred",

in that they will be compatibly supported in appropriate hardware environments:

1. A ROM-unit of length 32K bytes. This may be mapped into either paged or unpaged address space, and may require write access (as well as read/execute access) to some or all of its addresses.

2. A ROM-unit of length 128K bytes. This may be mapped only into paged address space (and hence can be used only in paged environments). It cannot require write access. The ROM-unit must not contain the word value $A54F at displacements 32K, 64K or 96K bytes from the start of the unit.

Other ROM-unit configurations are possible in specific environments, in particular for use by system software. Such configurations are outwith the scope of this document.

Each ROM unit has a standard layout. The unit starts with a unit header in its lowest addressed four bytes as follows:

bytes 0,1 : $A54F. This value confirms that a ROM unit is present

byte 2 : address space requirement of unit: number of bytes divided by 8K. For a 32K unit the value is 4; for a 128K unit the value is 16.

byte 3 : bit 0 = 0 : unit is not checksummed

bit 0 = 1 : unit is checksummed

bit 1 = 0 : only read (execute) access to the unit is required.

bit 1 = 1 : both read and write access to the unit are required. (INVALID CAPSULE will be reported if read access and write access cannot be distinguished at this address).

bit 2 to 7 : reserved (zero)

There then follow any number of programs. Each program is a multiple of 2 bytes in length, and is contiguous with the preceding program or the unit header. The length of each program is defined in its program header (see section A1.3).

The last program in the unit is immediately followed by a 16-byte unit trailer as follows:

bytes 0 to 11	spaces
bytes 12 to 15	zero

If the unit header indicates the presence of a checksum, the checksum value is held in the last four bytes of the unit. The checksum value is the long word (32-bit) sum of the zero-extended unsigned 16-bit words in the ROM unit preceding the checksum, taken in ascending address order and ignoring overflow whenever it occurs.

Space between the end of the unit trailer and the checksum bytes (or the end of the unit) is undefined as far as Director is concerned. (It is theoretically possible for this region to contain defined data, but it cannot contain programs in the standard sense, and the address of the data cannot be deduced by Director facilities.)

In a conventional ROM unit, the whole of the address space is ROM: the standard layout described above fills the whole address space, and the checksum (if any) applies to the whole unit. However, for special purposes, the following variations are permitted:

1. Only the lower addressed part of the unit (some multiple of 8K bytes in length) may be ROM, with the remainder used as RAM or for some other purpose. (The distinction between ROM and other usage is as defined for the Kernel procedure CHECK ADDRESS; see the Kernel specification [1].) In this case, the standard unit layout described above is contained within the ROM part of the unit, and the checksum (if any) applies only to that part. The value in byte 2 of the unit header defines the address space used by the whole unit. The circumstances in which this option might be exercised are beyond the scope of this document.

2. The whole of the unit may be RAM. This case is treated exactly as if the whole unit were ROM, and is intended for use in development systems in which ROM slots may be represented by RAM into which programs under test may be loaded.

A1.3 Program header format

Each program starts with 26 (or more) bytes of program header as follows:

bytes 0 to 11	program name (see section A1.5)
bytes 12 to 15	program length in bytes including header (multiple of 2)
bytes 16 to 19	program entry point (positive even byte displacement from start of header) for a machine code program; other types of program may define these bytes differently. The implied entry point must lie within the unit that contains the program, and must be distinct from any other machine code program entry point in that unit.

bytes 20 to 23	amount of stack space (in bytes) required by machine code program; other types of program may define these bytes differently. In the case of a program that may be entered as an activity, this value does not include the 72-byte register dump area
bytes 24 to 25	program properties (see section A1.4)
byte 26	(only required if bit 7 of byte 25 is 1) length of name that is to be displayed in system menus for this program. The maximum length is 22 bytes
bytes 27 onwards	menu name (if any) whose length is defined in byte 26
subsequent bytes	additional program header fields. These fields are present only if bit 6 of byte 25 is 1. They start immediately following the menu name (or in byte 26 if there is no menu name). Their format is defined in section A1.6.
subsequent bytes	program itself. A machine code program consists of the required machine code instructions and constant data; instructions must fall on an even byte boundary. The formats of programs of other types are outside the scope of this document

A1.4 Program properties

Byte 24 Bit 0 : For a non-machine code program, the definition of this bit is language-dependent.

For a machine code program, the bit has the values:

0 program is untrusted
1 program is trusted

(See section 4.4 for details. A machine code program must be trusted if it represents an application (bit 2 is zero). An untrusted machine code application causes the error message BAD PROGRAM or the error response ERR.NP.)

Bit 1 : 0 program is machine code
1 program is not machine code. (Details of the handling of non-machine code programs are beyond the scope of this document.)

Bit 2 : 0 program represents entry point to an application (and should therefore be offered on application menu subject to bit 3)
1 program is only intended to be loaded/invoked by other programs
This bit should always be zero for a non-machine code program

Bit 3 : 0 application represented by this program is to be visible to the end user
1 to be invisible

Bit 4 : 0

Bit 5 : 0 application cannot be tele-started
1 application can be tele-started (see section 4.4 for details)

Bit 6 : 0 application is not 'always reviewable'
1 application is 'always reviewable' (see section 6 for details)

Bit 7 : 0

Byte 25

Bits 0 to 3 : 0

Bit 4 : 0 application is not to be entered automatically (unless it is configured as the First Application)
1 application is to be entered automatically on power-up/reset. Bit 2 of byte 24 must be 0 (see section 4.4.1 for details)
This bit is ignored unless the program is in ROM

Bit 5 : 0 forced page switch is required: when this program is called by another program, the activity's page will be set to that containing this program (or to 'no page required' if this program is in unpaged store)
1 forced page switch is not required: when this program is called by another program, the activity's page will be changed only on a call from unpaged to paged store; a call between different pages will fail.

The significance of this bit is described in section 4.6.

Bit 6 : 0 there are no additional program header fields
1 there are additional program header fields (see section A1.6)

Bit 7 : 0 the name to be displayed in system menus is the program name specified in bytes 0 to 11
1 the name to be displayed in system menus is the menu name specified in bytes 26 onwards. (Additional rules apply to system software names: these rules are outside the scope of this document.)

This bit should only be set for a program held in ROM

A1.5 Name format

In all cases where a 'name' is supplied to an interface defined in this document, the name consists of 12 characters chosen from ASCII $20 to $7A inclusive. Director does not necessarily check adherence to this standard. When comparing names, Director treats all 12 characters as significant, but treats upper and lower case versions of the same letter as equivalent.

Names visible to the end user should avoid obscure punctuation, leading spaces and embedded non-single spaces.

Names to be loaded from filestore must conform to filestore naming conventions.

(System Software may use names containing the additional character $7E (tilde). Such usage is not normally visible to the end user.)

A1.6 Additional program header fields

If bit 6 of byte 25 of the program header is set to 1, the program header contains additional fields, which can be accessed using the Director procedure GIVE PROGRAM HEADER FIELD (see section 4.9.8). Each such field consists of a number of bytes, as defined below. The fields are contiguous, and the last field is followed by a zero byte. The fields can occur in any order.

The format of each field is as follows:

byte 0 : field identifier

byte 1 : length of field data in bytes (1 to 255)

byte 2 onwards : field data

Field identifiers 1 to 63 are reserved for system-wide purposes. Field identifiers 64 to 127 can be used for application-specific purposes. Field identifiers 128 to 255 are reserved.

The following system-wide field identifiers are defined:

Identifier 1 : program version number. It is an OPD standard that each separately released application or program should have a version number. An application should by default display its version number to the user, and should where possible provide the means of displaying the version number of any independent program that it uses. (Modified rules for display may apply to applications released as part of

the system software.) Version numbers may be accessed by other applications or system utilities. The version number is by convention an 8-character field as follows:

bytes 0 to 2 : territory identifier (right justified), for example, AUS, SAF, NAD
byte 3 : space
byte 4 to 5 : enhancement level (two digits, optionally zero suppressed)
byte 6 : full stop
byte 7 : revision number (one digit)

For example, AUS 04.1

APPENDIX 2. PROGRAM FORMAT IN FILESTORE

Each program is held in a file of the same name as the program.

The File Type is 1 for a machine code program, and 2 for a 'published' non-machine code program. Other program types are outside the scope of this document.

The File Type Qualifier is defined in the same way as byte 24 of the program header (see section A1.4).

The content of a machine code program file is as for such a program in ROM: a program header (positioned at byte 0 of the file) followed by the program itself. Space between the end of the program (as defined in the program header) and the end of the file should be regarded as undefined. When the program is loaded, the whole file is loaded into RAM, irrespective of the program size specified in the program header. The file size should therefore be the minimum necessary to accommodate the declared program size. There is no unit header or unit trailer in a program file.

The definition of the content of a non-machine code program file is outside the scope of this document. The file does not necessarily contain a program header.