Director Facilities for Application Writers

PSD 76.97.3.2

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 communications 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 ejected 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 partfally 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 PF (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

0. DOCUMENT CONTROL

0.1 Contents

0. DOCUMENT CONTROL
0.1 Contents List
0.2 Changes Since Previous Issue
0.3 Document Predecessors
0.4 Changes Forecast
0.5 Document Cross References

1. GENERAL
1.1 Scope
1.2 Introduction
1.3 Terminology

2. SUMMARY

3. DIRECTOR INTERFACE
3.1 General

4. APPLICATIONS AND ACTIVITIES
4.1 Terminology
4.2 Application structure
4.3 Application types
4.4 Application invocation
4.4.1 Auto Entry to Applications
4.4.2 State of Activity on entry
4.4.2.1 Priority
4.4.2.2 Registers
4.4.2.3 Stack and dump area
4.4.2.4 Rank
4.4.2.5 Events
4.5 Application termination
4.6 Loading and calling programs
4.6.1 Loading programs
4.6.2 Calling programs
4.7 Secondary activity invocation
4.8 Loading from microdrive cartridge
4.9 Interfaces
4.9.1 Start Application
4.9.2 Start Activity
4.9.3 Load Program
4.9.4 Release Program
4.9.5 Destroy Application
4.9.6 Inhibit Application
4.9.7 Give System Version Numbers
4.9.8 Give Program Header Field

5. SCREEN/KEYBOARD HANDLING
5.1 Introduction
5.2 Architectural Overview
5.3 Mechanisms of foreground control
5.4 Foreground state response

it 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
Actor Value (DO.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: f.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 (D0.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(D0.B): D. INHAPP

Additional Call Parameter:

D1.W : bit 0 : 1 put application into 'Abandoned' state
0 do not
bit 1 : (actionec 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:

D0.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 (D0.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 (D0.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 or 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

5.5 Keyboard handling
5.6 Interfaces
5.6.1 Request Foreground
5.6.2 Release Foreground
5.6.3 Suspend Foreground
5.6.4 Read Menu Key

6. REVIEW
6.1 Purpose
6.2 User Interface
6.3 Handling of Review by application
6.4 Use of Keyboard during Review
6.5 Ephemeral System Applications
6.6 Interfaces
6.6.1 Read Review Key
6.6.2 Exit Review
6.6.3 Declare Final Review Screen

7. SEGMENTS
7.1 Segment names
7.2 Reviewable segments
7.3 Segment usage
7.4 Store Usage Report
7.5 Interfaces
7.5.1 Change Segment Properties
7.5.2 Destroy Segment
7.5.3 Get Segment Identifier
7.5.4 Request Read Access to Segment
7.5.5 Request Write Access to Segment
7.5.6 Release Write Access to Segment
7.5.7 Release Access to Segment

8. NOTICEBOARD
8.1 Description
8.2 Flags
8.3 Output Area
8.4 Telephony Noticeboard
8.5 Interfaces
8.5.1 Display Noticeboard Flag
8.5.2 Display Noticeboard Report
8.5.3 Cancel Noticeboard Report
8.5.4 Convert Date and Time

9. SOUND GENERATION
9.1 Description
9.2 Standard sounds
9.3 Special sounds
9.4 Interfaces
9.4.1 Make Sound

10. NAME TABLE
10.1 Description
10.2 Register of object types
10.3 Interfaces

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 DPD 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 DPD, 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.

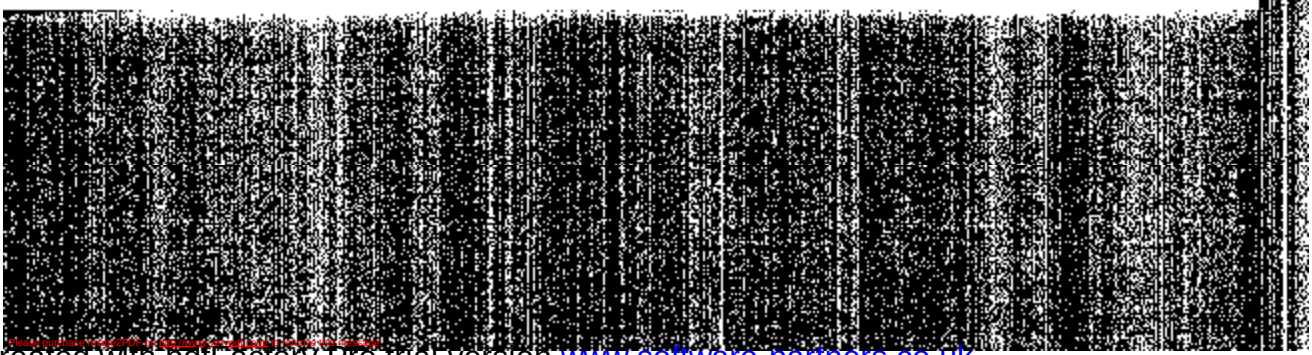
2. Extended applications. Such applications are typified by lengthy user interaction via screen and keyboard. The user is likely to want to break off to perform other urgent tasks while retaining the ability to return to the extended application. When the user selects a new foreground application by 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/RECIAL. 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 DPD 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 performed 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