Mixed Mode Manager Reference

Carbon > Runtime Architecture

2003-04-01

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Tables

Mixed Mode Manager Reference

Framework: CoreServices/CoreServices.h

Declared in MixedMode.h

Overview

Mac OS X does not require the Mixed Mode Manager, and does not support its functions. These unsupported functions are listed in the Appendix. The functions have been removed from the Mixed Mode Manager and redefined as macros for the purpose of source compatibility with code ported to CFM. See the header file MixedMode.h for details on these macros and their usage.

You do not need to remove Mixed Mode Manager calls from your application for compatibility with Mac OS X, and may want to retain them for source code compatibility with previous versions of the Mac OS.

The Mixed Mode Manager managed the mixed-mode architecture of PowerPC processor-based computers running 680x0-based code (including system software, applications, and stand-alone code modules). The Mixed Mode Manager cooperated with the 68LC040 Emulator to provide a fast, efficient, and virtually transparent method for code in one instruction set architecture to call code in another architecture. The Mixed Mode Manager handled all the details of switching between architectures.

The Mixed Mode Manager was intended to operate transparently to most applications and other software.

Although Mac OS X does not run 68K code, Carbon supports universal procedure pointers (UPPs) transparently, so you do not have to change them or remove them from your code. You may want to keep Mixed Mode Manager calls in your application to maintain source code compatibility with the previous versions of the Mac OS. Mixed Mode Manager calls from Carbon applications running on Mac OS 8 or 9 will function normally.

The Mixed Mode Manager was used by developers who

- wanted to recompile their applications into PowerPC code and their applications passed the address of some routines to the Mac OS using a reference of type `ProcPtr`
- created applications–written in either PowerPC or 680x0 code–that support installable code modules that might be written in a different architecture
- wrote stand-alone code (for example, a VBL task or a component) that could be called from either the PowerPC native environment or the 680x0 emulated environment
- wrote debuggers or other software that needed to know about the structure of the stack at any time (for example, during a mode switch)

Mac OS X will not run 68K code. Although Carbon supports universal procedure pointers (UPPs), applications should use ProcPtrs for their own code and plug-ins and use the new system-supplied UPP creation functions for Toolbox callback UPPs. You still need to dispose of those UPPs (using the corresponding disposal function), so that any allocated memory can be cleaned up when your application is running on Mac OS 8 or 9.

Data Types

MixedModeStateRecord

Contains mixed mode state information.

```
struct MixedModeStateRecord {
    UInt32 state1;
    UInt32 state2;
    UInt32 state3;
    UInt32 state4;
};
typedef struct MixedModeStateRecord MixedModeStateRecord;
```

Fields

`state1`

`state2`

`state3`

`state4`

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MixedMode.h`

ProcInfoType

Defines a data type used to encode a routine's procedure information.

```
typedef unsigned long ProcInfoType;
```

Discussion

The Mixed Mode Manager uses a long word of type `ProcInfoType` to encode a routine's procedure information, which contains essential information about the calling conventions and other features of a routine. These values specify

- the routine's calling conventions
- the sizes and locations of the routine's parameters, if any
- the size and location of the routine's result, if any

The Mixed Mode Manager provides a number of constants that you can use to specify the procedure information. See "Procedure Information Size Constants" (page 17) , "ProcInfo Field Offset And Width Constants" (page 18) , "Calling Convention Constants" (page 12) , "Special Case Calling Convention Constants" (page 26) , and "Register Constants" (page 21).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MixedMode.h`

RDFlagsType

Defines a data type for routine descriptor flags.

```
typedef UInt8 RDFlagsType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MixedMode.h`

RoutineDescriptor

Contains information used by the Mixed Mode Manager to execute a routine.

```
struct RoutineDescriptor {
    UInt16 goMixedModeTrap;
    SInt8 version;
    RDFlagsType routineDescriptorFlags;
    UInt32 reserved1;
    UInt8 reserved2;
    UInt8 selectorInfo;
    UInt16 routineCount;
    RoutineRecord routineRecords[1];
};
typedef struct RoutineDescriptor RoutineDescriptor;
typedef RoutineDescriptor * RoutineDescriptorPtr;
typedef RoutineDescriptorPtr RoutineDescriptorHandle;
```

Fields

`goMixedModeTrap`
: An A-line instruction that is used privately by the Mixed Mode Manager. When the emulator encounters this instruction, it transfers control to the Mixed Mode Manager. This field contains the value $AAFE.

`version`
: The version number of the `RoutineDescriptor` data type.

`routineDescriptorFlags`
: A set of routine descriptor flags. Currently, all the bits in this field should be set to 0, unless you are specifying a routine descriptor for a dispatched routine.

`reserved1`
: Reserved. This field must initially be 0.

`reserved2`
: Reserved. This field must be 0.

`selectorInfo`
: Reserved. This field must be 0.

`routineCount`
: The index of the final routine record in the following array, of routineRecords. Because the `routineRecords` array is zero-based, this field does not contain an actual count of the routine records contained in that array. Often, you will use a routine descriptor to describe a single procedure, in which case this field should contain the value 0. You can, however, construct a routine descriptor that contains pointers to both 680x0 and PowerPC code (known as a “fat” routine descriptor). In that case, this field should contain the value 1.

`routineRecords`

An array of routine records for the routines described by this routine descriptor. See "`RoutineRecord`" (page 10) for the structure of a routine record. This array is zero-based.

Discussion
A routine descriptor is a data structure used by the Mixed Mode Manager to execute a routine. The external interface to a routine descriptor is through a universal procedure pointer, of type `UniversalProcPtr`, which is defined as a procedure pointer (if the code is 680x0 code) or as a pointer to a routine descriptor (if the code is PowerPC code). A routine descriptor is defined by the `RoutineDescriptor` data type.

Your application (or other software) should never attempt to guide its execution by inspecting the value in the `ISA` field of a routine record and jumping to the address in the `procDescriptor` field.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MixedMode.h`

RoutineFlagsType

Defines a data type for routine flags.

```
typedef unsigned short RoutineFlagsType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MixedMode.h`

RoutineRecord

Cntains information about a particular routine.

```
struct RoutineRecord {
    ProcInfoType procInfo;
    SInt8 reserved1;
    ISAType ISA;
    RoutineFlagsType routineFlags;
    ProcPtr procDescriptor;
    UInt32 reserved2;
    UInt32 selector;
};
typedef struct RoutineRecord RoutineRecord;
typedef RoutineRecord * RoutineRecordPtr;
typedef RoutineRecordPtr RoutineRecordHandle;
```

Fields

`procInfo`

A value of type `ProcInfoType` that encodes essential information about the routine's calling conventions and parameters. See "Procedure Information Size Constants" (page 17), "ProcInfo Field Offset And Width Constants" (page 18), "Calling Convention Constants" (page 12), "Special Case Calling Convention Constants" (page 26), and "Register Constants" (page 21) for descriptions of the constants you can use to set this field.

`reserved1`

Reserved. This field must be 0.

`ISA`

The instruction set architecture of the routine. See "Instruction Set Architectures" (page 14) for a complete listing of the constants you can use to set this field.

`routineFlags`

A value of type `RoutineFlagsType` that contains a set of flags describing the routine. See "Routine Entry Point Flags" (page 25), "Fragment Flags" (page 14), "ISA Flags" (page 15), "Routine Selector Flags" (page 25), and "Default Routine Flags" (page 13) for descriptions of the constants you can use to set this field.

`procDescriptor`

A pointer to the routine's code. If the routine consists of 680x0 code and the `kProcDescriptorIsAbsolute` flag is set in the `routineFlags` field, then this field contains the address of the routine's entry point. If the routine consists of 680x0 code and the `kProcDescriptorIsRelative` flag is set, then this field contains the offset from the beginning of the routine descriptor to the routine's entry point. If the routine consists of PowerPC code, the `kFragmentIsPrepared` flag is set, and the `kProcDescriptorIsAbsolute` flag is set, then this field contains the address of the routine's transition vector. If the routine consists of PowerPC code, the `kFragmentNeedsPreparing` flag is set, and the `kProcDescriptorIsRelative` flag is set, then this field contains the offset from the beginning of the routine descriptor to the routine's entry point.

`reserved2`

Reserved. This field must be 0.

`selector`

Reserved. This field must be 0. For routines that are dispatched, this field contains the routine selector.

Discussion

A routine record is a data structure that contains information about a particular routine. The routine descriptor specifies, among other things, the instruction set architecture of the routine, the number and size of the routine's parameters, the routine's calling conventions, and the routine's location in memory. At least one routine record is contained in the `routineRecords` field of a routine descriptor. A routine record is defined by the `RoutineRecord` data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MixedMode.h`

Constants

Calling Convention Constants

Specify a routine's calling conventions.

```
typedef unsigned short CallingConventionType;
enum {
    kPascalStackBased = 0,
    kCStackBased = 1,
    kRegisterBased = 2,
    kD0DispatchedPascalStackBased = 8,
    kD1DispatchedPascalStackBased = 12,
    kD0DispatchedCStackBased = 9,
    kStackDispatchedPascalStackBased = 14,
    kThinkCStackBased = 5
};
```

Constants

`kPascalStackBased`
The routine follows normal Pascal calling conventions.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kCStackBased`
The routine follows the C calling conventions employed by the MPW development environment.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterBased`
The parameters are passed in registers.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kD0DispatchedPascalStackBased`
The parameters are passed on the stack according to Pascal conventions, and the routine selector is passed in register D0.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kD1DispatchedPascalStackBased`
The parameters are passed on the stack according to Pascal conventions, and the routine selector is passed in register D1.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kD0DispatchedCStackBased`
: The parameters are passed on the stack according to C conventions, and the routine selector is passed in register D0.

  Available in Mac OS X v10.0 and later.

  Declared in `MixedMode.h`.

`kStackDispatchedPascalStackBased`
: The routine selector and the parameters are passed on the stack.

  Available in Mac OS X v10.0 and later.

  Declared in `MixedMode.h`.

`kThinkCStackBased`
: The routine follows the C calling conventions employed by the THINK C software development environment. Arguments are passed on the stack from right to left, and a result is returned in register D0. All arguments occupy an even number of bytes on the stack. An argument having the size of a `char` is passed in the high-order byte. You should always provide function prototypes; failure to do so may cause THINK C to generate code that is incompatible with this parameter-passing convention.

  Available in Mac OS X v10.0 and later.

  Declared in `MixedMode.h`.

Discussion
These constants are used by the `ProcInfoType` (page 8) type to specify a routine's calling conventions.

Default Routine Flags

Specify defaults for a routine.

```
enum {
    kRoutineIsNotDispatchedDefaultRoutine = 0x00,
    kRoutineIsDispatchedDefaultRoutine = 0x10
};
```

Constants
`kRoutineIsNotDispatchedDefaultRoutine`
: This routine is not the default routine for a set of routines that is dispatched using a routine selector.

  Available in Mac OS X v10.0 and later.

  Declared in `MixedMode.h`.

`kRoutineIsDispatchedDefaultRoutine`
: This routine is the default routine for a set of routines that is dispatched using a routine selector. If a set of routines is dispatched using a routine selector and the routine corresponding to a specified selector cannot be found, this default routine is called. This routine must be able to accept the same procedure information for all routines. If possible, it is passed the procedure information passed in a call to `CallUniversalProc`.

  Available in Mac OS X v10.0 and later.

  Declared in `MixedMode.h`.

Discussion
The `routineFlags` field of a routine record contains a set of flags that specify information about a routine. You can use constants to specify the desired routine flags. Currently, only 5 of the 16 bits in a routine flags word are defined. You should set all the other bits to 0.

In general, you should use the constant `kRoutineIsNotDispatchedDefaultRoutine`. The constant and `kRoutineIsDispatchedDefaultRoutine` is reserved for use with selector-based system software routines.

Fragment Flags

Used in the `routineFlags` field of a routine record.

```
enum {
    kFragmentIsPrepared = 0x00,
    kFragmentNeedsPreparing = 0x02
};
```

Constants

`kFragmentIsPrepared`

The fragment containing the code to be executed is already loaded into memory and prepared by the Code Fragment Manager.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kFragmentNeedsPreparing`

The fragment containing the code to be executed needs to be loaded into memory and prepared by the Code Fragment Manager. If this flag is set, the `kPowerPCISA` and `kProcDescriptorIsRelative` flags should also be set.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion

The `routineFlags` field of a routine record contains a set of flags that specify information about a routine. You can use constants to specify the desired routine flags. Currently, only 5 of the 16 bits in a routine flags word are defined. You should set all the other bits to 0.

Instruction Set Architectures

Used in the `ISA` field of a routine record.

```
typedef SInt8 ISAType;
enum {
    kM68kISA = 0,
    kPowerPCISA = 1
};
```

Constants

`kM68kISA`

The routine consists of Motorola 680x0 code.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kPowerPCISA`

The routine consists of PowerPC code.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
The `ISA` field of a routine record contains a flag that specifies the instruction set architecture of a routine. You can use constants to specify the instruction set architecture.

ISA Flags

Used in the `routineFlags` field of a routine record.

```
enum {
    kUseCurrentISA = 0x00,
    kUseNativeISA = 0x04
};
```

Constants

`kUseCurrentISA`
If possible, use the current instruction set architecture when executing a routine.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kUseNativeISA`
Use the native instruction set architecture when executing a routine.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

Discussion
The `routineFlags` field of a routine record contains a set of flags that specify information about a routine. You can use constants to specify the desired routine flags. Currently, only 5 of the 16 bits in a routine flags word are defined. You should set all the other bits to 0.

Current Mixed Mode State

Specifies the current version of the mixed-mode state record.

```
enum {
    kCurrentMixedModeStateRecord = 1
};
```

RTA Types

```
typedef SInt8 RTAType;
enum {
    kOld68kRTA = 0 << 4,
    kPowerPCRTA = 0 << 4,
    kCFM68kRTA = 1 << 4
};
```

Constants

`kOld68kRTA`
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kPowerPCRTA`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCFM68kRTA`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Procedure Descriptors

```
enum {
    kProcDescriptorIsProcPtr = 0x00,
    kProcDescriptorIsIndex = 0x20
};
```

Constants

`kProcDescriptorIsProcPtr`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kProcDescriptorIsIndex`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Routine Descriptor Version

Specifies the version of routine descriptor.

```
enum {
    kRoutineDescriptorVersion = 7
};
```

Special Case Constant

Used to specify a special case.

```
enum {
    kSpecialCase = 0x000F
};
```

Constants

`kSpecialCase`
The routine is a special case. You can use the following constants to specify a special case.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

kX86ISA

```
enum {
    kX86ISA = 2
};
```

Constants

`kX86ISA`

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

kX86RTA

```
enum {
    kX86RTA = 2 << 4
};
```

Constants

`kX86RTA`

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

_MixedModeMagic

```
enum {
    _MixedModeMagic = 0xAAFE
};
```

Constants

`_MixedModeMagic`

Procedure Information Size Constants

Specify the size (in bytes) of a value encoded in the procedure information for a routine.

```
enum {
    kNoByteCode = 0,
    kOneByteCode = 1,
    kTwoByteCode = 2,
    kFourByteCode = 3
};
```

Constants

`kNoByteCode`

The value occupies no bytes.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kOneByteCode`
The value occupies 1 byte.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kTwoByteCode`
The value occupies 2 bytes.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kFourByteCode`
The value occupies 4 bytes.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
These constants are used by the `ProcInfoType` (page 8) to specify the size (in bytes) of a value encoded in a routine's procedure information.

ProcInfo Field Offset And Width Constants

Specify offsets to fields and the widths of the fields within a value.

```
enum {
    kCallingConventionWidth = 4,
    kCallingConventionPhase = 0,
    kCallingConventionMask = 0x0F,
    kResultSizeWidth = 2,
    kResultSizePhase = kCallingConventionWidth,
    kResultSizeMask = 0x30,
    kStackParameterWidth = 2,
    kStackParameterPhase = (kCallingConventionWidth + kResultSizeWidth),
    kStackParameterMask = 0xFFFFFFC0,
    kRegisterResultLocationWidth = 5,
    kRegisterResultLocationPhase = (kCallingConventionWidth + kResultSizeWidth),
    kRegisterParameterWidth = 5,
    kRegisterParameterPhase = (kCallingConventionWidth + kResultSizeWidth
+ kRegisterResultLocationWidth),
    kRegisterParameterMask = 0x7FFFF800,
    kRegisterParameterSizePhase = 0,
    kRegisterParameterSizeWidth = 2,
    kRegisterParameterWhichPhase = kRegisterParameterSizeWidth,
    kRegisterParameterWhichWidth = 3,
    kDispatchedSelectorSizeWidth = 2,
    kDispatchedSelectorSizePhase = (kCallingConventionWidth + kResultSizeWidth),
    kDispatchedParameterPhase = (kCallingConventionWidth + kResultSizeWidth
+ kDispatchedSelectorSizeWidth),
    kSpecialCaseSelectorWidth = 6,
    kSpecialCaseSelectorPhase = kCallingConventionWidth,
    kSpecialCaseSelectorMask = 0x03F0
};
```

Constants

`kCallingConventionWidth`

The number of bits in the procedure information that encode the calling convention information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCallingConventionPhase`

The offset from the least significant bit in the procedure information to the calling convention information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCallingConventionMask`

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kResultSizeWidth`

The number of bits in the procedure information that encode the function result size information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kResultSizePhase`

The offset from the least significant bit in the procedure information to the function result size information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kResultSizeMask`
A mask for the bits in the procedure information that encode the function result size information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kStackParameterWidth`
The number of bits in the procedure information that encode the size of a stack-based parameter.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kStackParameterPhase`
The offset from the least significant bit in the procedure information to the stack parameter information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kStackParameterMask`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterResultLocationWidth`
The number of bits in the procedure information that encode which register the result will be stored in.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterResultLocationPhase`
The offset from the least significant bit in the procedure information to the result register information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterWidth`
The number of bits in the procedure information that encode the information about a register-based parameter.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterPhase`
The offset from the least significant bit in the procedure information to the register parameter information.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterMask`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterSizePhase`
The offset from the beginning of a register parameter information field to the encoded size of the parameter.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterSizeWidth`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterWhichPhase`
The offset from the beginning of a register parameter information field to the encoded register.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kRegisterParameterWhichWidth`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kDispatchedSelectorSizeWidth`
The number of bits in the procedure information that encode the size of a routine-dispatching selector.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kDispatchedSelectorSizePhase`
The offset from the least significant bit in the procedure information to the selector size information of a routine that is dispatched though a selector.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kDispatchedParameterPhase`
The offset from the least significant bit in the procedure information to the parameter information of a routine that is dispatched though a selector.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseSelectorWidth`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseSelectorPhase`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseSelectorMask`
Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
The offsets to fields and the widths of the fields within a value of type `ProcInfoType` (page 8) are defined by constants.

Register Constants

Specify registers that are encoded in the procedure information for a routine.

```
enum {
    kRegisterD0 = 0,
    kRegisterD1 = 1,
    kRegisterD2 = 2,
    kRegisterD3 = 3,
    kRegisterD4 = 8,
    kRegisterD5 = 9,
    kRegisterD6 = 10,
    kRegisterD7 = 11,
    kRegisterA0 = 4,
    kRegisterA1 = 5,
    kRegisterA2 = 6,
    kRegisterA3 = 7,
    kRegisterA4 = 12,
    kRegisterA5 = 13,
    kRegisterA6 = 14,
    kCCRegisterCBit = 16,
    kCCRegisterVBit = 17,
    kCCRegisterZBit = 18,
    kCCRegisterNBit = 19,
    kCCRegisterXBit = 20
};
typedef unsigned short registerSelectorType;
```

Constants

`kRegisterD0`
Register D0.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD1`
Register D1.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD2`
Register D2.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD3`
Register D3.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD4`
Register D4.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD5`
Register D5.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD6`
Register D6.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterD7`
Register D7.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA0`
Register A0.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA1`
Register A1.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA2`
Register A2.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA3`
Register A3.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA4`
Register A4.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA5`
Register A5.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kRegisterA6`
Register A6.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kCCRegisterCBit`
The C (carry) flag of the Status Register.
Available in Mac OS X v10.0 and later.
Declared in `MixedMode.h`.

`kCCRegisterVBit`
The V (overflow) flag of the Status Register.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCCRegisterZBit`
The Z (zero) flag of the Status Register.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCCRegisterNBit`
The N (negative) flag of the Status Register.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kCCRegisterXBit`
The X (extend) flag of the Status Register.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
For register-based routines, the registers are encoded in the routine's procedure information using these constants.

Routine Descriptor Flags

Specify attributes of the described routine.

```
enum {
    kSelectorsAreNotIndexable = 0x00,
    kSelectorsAreIndexable = 0x01
};
```

Constants
`kSelectorsAreNotIndexable`
For dispatched routines, the recognized routine selectors are not contiguous.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSelectorsAreIndexable`
For dispatched routines, the recognized routine selectors are contiguous and therefore indexable.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
The `routineDescriptorFlags` field of a routine descriptor contains a set of routine descriptor flags that specify attributes of the described routine. You can use constants to specify the routine descriptor flags. In general, you should use the constant `kSelectorsAreNotIndexable` when constructing your own routine descriptors; the value `kSelectorsAreIndexable` is reserved for use by Apple.

Routine Entry Point Flags

Specify information about the entry point for a routine.

```
enum {
    kProcDescriptorIsAbsolute = 0x00,
    kProcDescriptorIsRelative = 0x01
};
```

Constants

`kProcDescriptorIsAbsolute`

The address of the routine's entry point specified in the `procDescriptor` field of a routine record is an absolute address.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kProcDescriptorIsRelative`

The address of the routine's entry point specified in the `procDescriptor` field of a routine record is relative to the beginning of the routine descriptor. If the code is contained in a resource and its absolute location is not known until run time, you should set this flag.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion

The `routineFlags` field of a routine record contains a set of flags that specify information about a routine. You can use constants to specify the desired routine flags. Currently, only 5 of the 16 bits in a routine flags word are defined. You should set all the other bits to 0.

Routine Selector Flags

Specify whether or not to pass a selector to a routine.

```
enum {
    kPassSelector = 0x00,
    kDontPassSelector = 0x08
};
```

Constants

`kPassSelector`

Pass the routine selector to the target routine as a parameter.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kDontPassSelector`

Do not pass the routine selector to the target routine as a parameter. You should not use this flag for 680x0 routines.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion

The `routineFlags` field of a routine record contains a set of flags that specify information about a routine. You can use constants to specify the desired routine flags. Currently, only 5 of the 16 bits in a routine flags word are defined. You should set all the other bits to 0.

In general, you should use the constant `kPassSelector`. The constant `kDontPassSelector` is reserved for use with selector-based system software routines.

Special Case Calling Convention Constants

Specify the calling conventions for a routine.

```
enum {
    kSpecialCaseHighHook = 0,
    kSpecialCaseCaretHook = 0,
    kSpecialCaseEOLHook = 1,
    kSpecialCaseWidthHook = 2,
    kSpecialCaseTextWidthHook = 2,
    kSpecialCaseNWidthHook = 3,
    kSpecialCaseDrawHook = 4,
    kSpecialCaseHitTestHook = 5,
    kSpecialCaseTEFindWord = 6,
    kSpecialCaseProtocolHandler = 7,
    kSpecialCaseSocketListener = 8,
    kSpecialCaseTERecalc = 9,
    kSpecialCaseTEDoText = 10,
    kSpecialCaseGNEFilterProc = 11,
    kSpecialCaseMBarHook = 12
};
```

Constants

`kSpecialCaseHighHook`

The routine follows the calling conventions documented in Inside Macintosh: Text; a rectangle is on the stack and a pointer is in register A3; no result is returned.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseCaretHook`

The routine follows the calling conventions documented in Inside Macintosh: Text; a rectangle is on the stack and a pointer is in register A3; no result is returned.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseEOLHook`

Parameters are passed to the routine in registers A3, A4, and D0, and output is returned in the Z flag of the Status Register. An `EOLHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseWidthHook`

Parameters are passed to the routine in registers A0, A3, A4, D0, and D1, and output is returned in register D1. A `WIDTHHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseTextWidthHook`
Parameters are passed to the routine in registers A0, A3, A4, D0, and D1, and output is returned in register D1. A `TextWidthHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseNWidthHook`
Parameters are passed to the routine in registers A0, A2, A3, A4, D0, and D1, and output is returned in register D1. An `nWIDTHHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseDrawHook`
Parameters are passed to the routine in registers A0, A3, A4, D0, and D1, and no result is returned. A `DRAWHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseHitTestHook`
Parameters are passed to the routine in registers A0, A3, A4, D0, D1, and D2, and output is returned in registers D0, D1, and D2. A `HITTESTHook` routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseTEFindWord`
Parameters are passed to the routine in registers A3, A4, D0, and D2, and output is returned in registers D0 and D1. A `TEFindWord` hook has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseProtocolHandler`
Parameters are passed to the routine in registers A0, A1, A2, A3, A4, and in the low-order word of register D1; output is returned in the Z flag of the Status Register. A protocol handler has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseSocketListener`
Parameters are passed to the routine in registers A0, A1, A2, A3, A4, in the low-order byte of register D0, and in the low-order word of register D1; output is returned in the Z flag of the Status Register. A socket listener has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseTERecalc`
Parameters are passed to the routine in registers A3 and D7, and output is returned in registers D2, D3, and D4. A TextEdit line-start recalculation routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseTEDoText`
Parameters are passed to the routine in registers A3, D3, D4, and D7, and output is returned in registers A0 and D0. A TextEdit text-display, hit-test, and caret-positioning routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseGNEFilterProc`
Parameters are passed to the routine in registers A1 and D0 and on the stack, and output is returned on the stack. A `GetNextEvent` filter procedure has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

`kSpecialCaseMBarHook`
Parameters are passed to the routine on the stack, and output is returned in register D0. A menu bar hook routine has these calling conventions.

Available in Mac OS X v10.0 and later.

Declared in `MixedMode.h`.

Discussion
These constants are used by the `ProcInfoType` (page 8) type to specify a routine's calling conventions.

Result Codes

The most common result codes returned by the Mixed Mode Manager are listed below.

Result Code	Value	Description
`mmInternalError`	-2526	An internal error has occurred. Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availabilty information by using the Mixed Mode Manager selectors defined in the Gestalt Manager. For more information see *Inside Mac OS X: Gestalt Manager Reference*.

Unsupported Functions

Table A-1 (page 29) lists functions that are unsupported and you should no longer use. These functions have been removed from the Mixed Mode Manager and redefined as macros for the purpose of source compatibility with code ported to CFM. See header file MixedMode.h for details.

Table A-1 Porting notes for unsupported Mixed Mode Manager functions

Unsupported functions	Porting notes
`CallOSTrapUniversalProc`	See the header file MixedMode.h for information.
`CallUniversalProc`	See the header file MixedMode.h for information.
`DisposeRoutineDescriptor`	Only useful for CFM-68K applications. Does nothing in PowerPC-native code.
`NewFatRoutineDescriptor`	Only useful for CFM-68K applications. Does nothing in PowerPC-native code.
`NewRoutineDescriptor`	Only useful for CFM-68K applications. Does nothing in PowerPC-native code.
`DisposeRoutineDescriptorTrap`	Only useful for CFM-68K applications. Does nothing in PowerPC-native code.
`NewFatRoutineDescriptorTrap`	Only useful for CFM-68K applications. Does nothing in PowerPC-native code.
`NewRoutineDescriptorTrap`	Only useful for CFM-68K applications. Does nothing in PowerPC-native code.
`RestoreMixedModeState`	Only useful for CFM-68K applications. Does nothing in PowerPC-native code.
`SaveMixedModeState`	Only useful for CFM-68K applications. Does nothing in PowerPC-native code.

Document Revision History

This table describes the changes to *Mixed Mode Manager Reference*.

Date	Notes
2003-04-01	Move unsupported functions to the Appendix.
	Added information to the introduction.
	Added abstracts to many data types and constants.
2003-02-01	Updated formatting, linking, and introduction.

Document Revision History

Index

Symbols

C

D

F

I

K

M

P

R

S