1. Crossing the Userland

by John Pagonis March 2003

The purpose of this article is to explain and familiarise developers with some of the internal mechanisms of the operating system. We will demystify some lower-level OS mechanisms that applications so commonly use; the services offered by the user library. Following we publicise some OS internals in order to provide insight into the design and construction of a modern mobile OS like Symbian OS.

The approach that will be followed herein is that of driving through the system from the top down, thus crossing from the application space down to the kernel space. Developers who are interested can thus go deeper into the OS, as well as get the chance to appreciate architectural features under the hood of Symbian OS.

1.1. Intro

Symbian OS follows a lightweight micro-kernel based design, where all executables dynamically link to a shared library called euser.dll (also referred to as the user library). In there live some very ubiquitous interfaces to the OS services, as well as utility classes and privileged calls. Considering and appreciating what happens every time these calls are executed is the purpose of the present article.

1.2. What is euser.dll for ?

In Symbian OS components never link to the kernel directly (i.e. statically - as one would expect in more traditional embedded OSs) but have to interface to the kernel through a dynamically linked library, called euser.dll. This is always located at a known address so that calls can be resolved at link/load time for more efficiency.

In fact, in euser.dll there can be found 3 families of exported calls, namely:

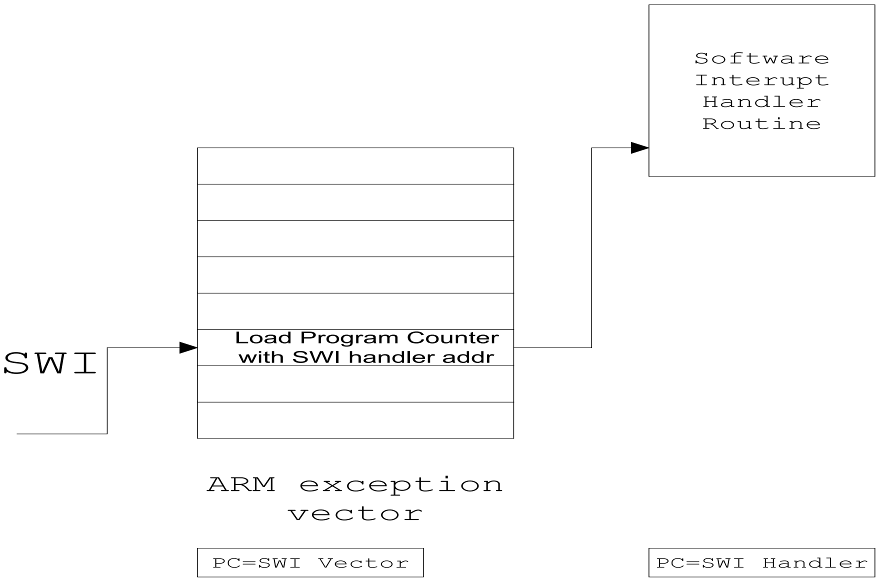
- Utility classes, methods and common constructs like descriptors, array classes, math functions, etc.
- Executive calls
- Kernel server requests

In this article we will not discuss further the first family of these exports, as they are effectively a set of library functionality as in any other shared dynamically linked library.

1.3. What are Executive Calls ?

Executive calls are the user library calls that allow a user thread to enter a processor privileged mode so that they can access, in a controlled and predefined way, hardware

resources or kernel space objects. What an executive call does (in euser.dll), is to switch control to the kernel executive.

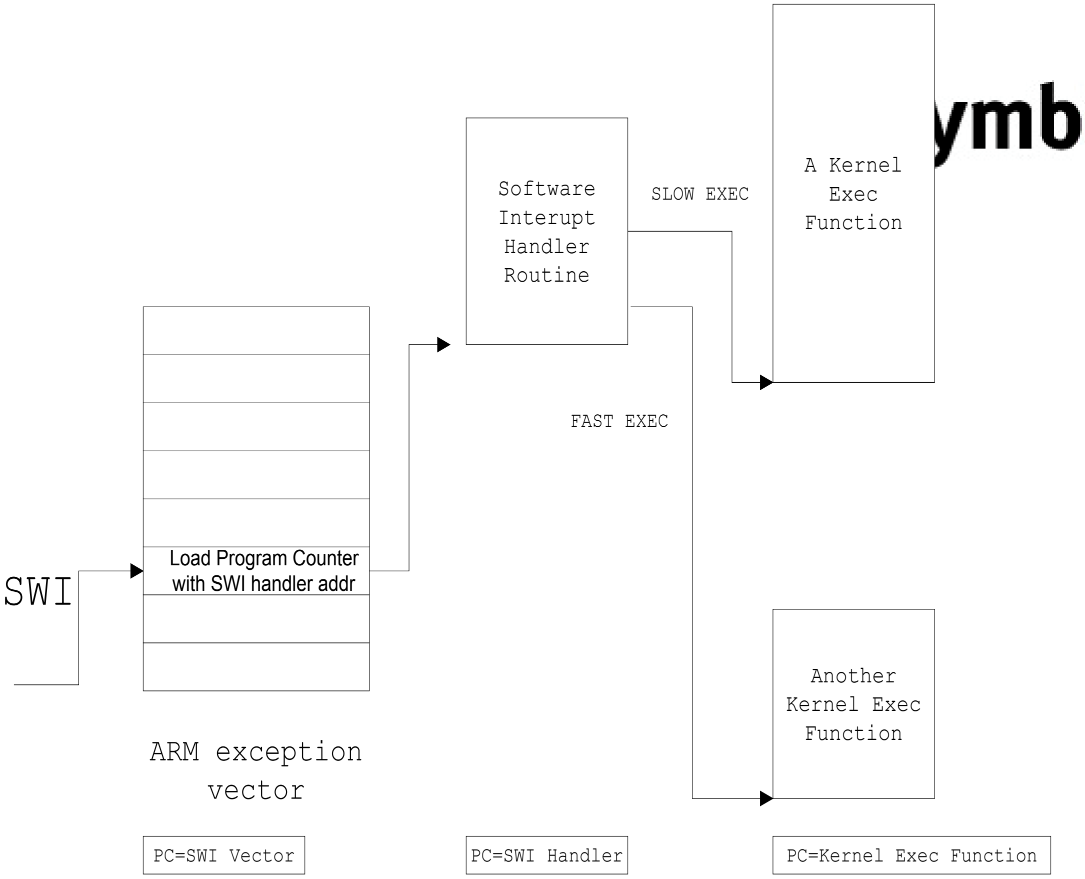

Software Interupt Handler Routine
Load Program Counter with SWI handler addr
SWI
ARM exception vector
PC=SWI Vector
PC=SWI Handler

In fact what happens is that, when programs call such user library functions, the user library has been pre-programmed to cause a software interrupt, thus causing the processor to branch to the software interrupt handler routine; as it has been setup at OS startup time (at the processor’s exception vector).

1.4. What happens on Executive calls ?

There, the software interrupt handler will check the type of the exec call and branch to the correct kernel function accordingly. On the ARM this identification is possible because on the exec call from the user library, the calling method passes an exec number as part of the software interrupt instruction. (the SWI instruction on the ARM allows for a 24bit-long exec number).

Actually there are 2 kinds of exec calls, as designed in Symbian OS, namely *slow* and *fast exec calls*.



Fast exec calls, operate with the interrupt requests (IRQs) disabled - but not the ARM fast interrupt requests (FIQs) - and thus are designed to be so short as not to impact interrupt latency, while they usually carry zero to one parameters. Such executive calls are mostly used to gain access to kernel-side objects or to hardware resources (like reading the system clock for example). Fast executive calls run in the context of the calling thread (although the processor is switched to supervisor mode); thus they use the heap of the calling thread.

Nevertheless in order to avoid faulting the system (remember user threads have entered privileged mode now) because of a lack of space on their stack, they make use of a predefined re-entrant stack.

It is also important to note that, following a fast executive call, the kernel does not try to reschedule any threads, so execution continues from the calling thread.

Slow exec calls operate with all interrupts enabled and thus can be interrupted by both FIQs and IRQs. Such executive calls are usually for operations that make use of more parameters (up to four), need to save more state and in general need more time for processing (for example when looking up a dll's entry point or ordinal). Slow exec calls run in the context of the calling thread and make use of either the kernel server or null thread stack.

Some slow executive calls may also call fast executive calls from the user library. After a slow executive call the scheduler will get the opportunity to switch if necessary to the highest, in priority, ready-to-run thread. Indeed before such a re-schedule takes place the kernel scheduler will attempt to sequentially execute any queued DFCs (deferred function calls, i.e. top half of interrupt handling routines).

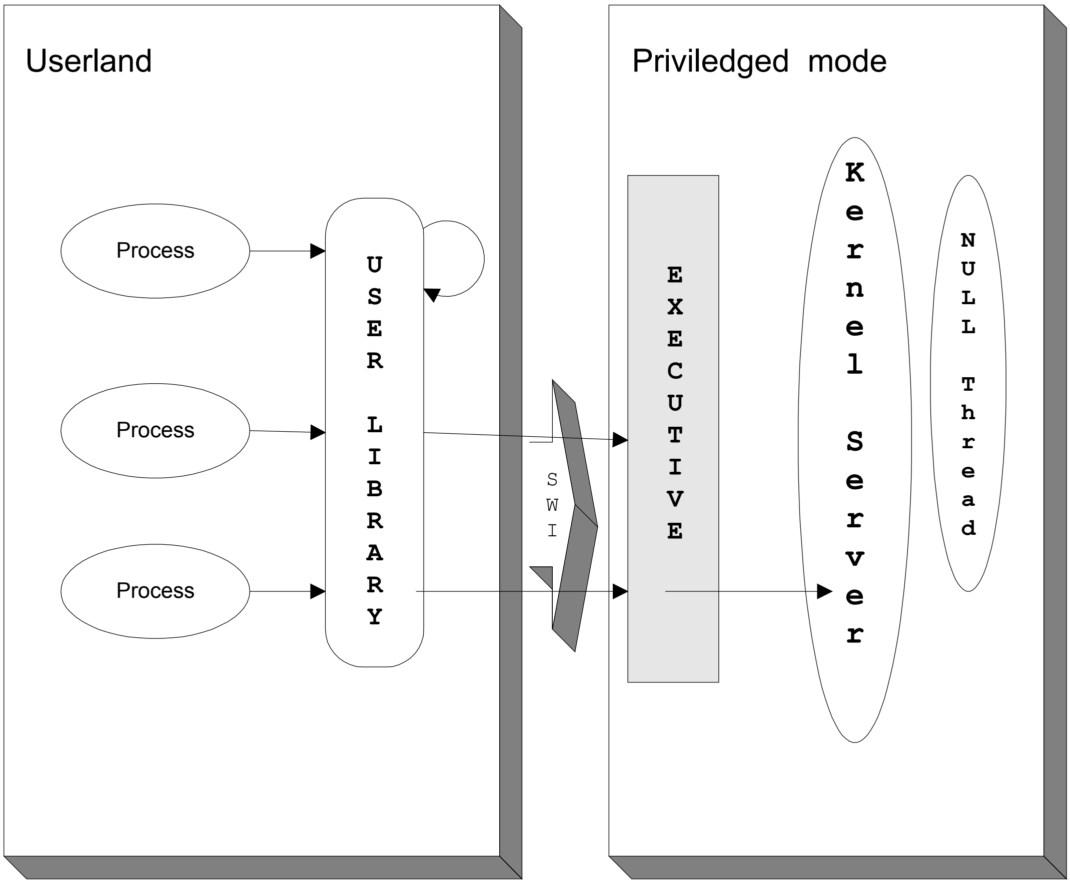
Thus we can see the justification as to why such executive calls are called slow; because a) they need to do more work, b) they can be interrupted and c) they may lead to context switching

1.5. What are Kernel Server requests ?

Executive calls, as we said, may access and even modify certain kernel-side objects, as well as offering privileged access to hardware. One thing though they are not allowed to do, is to create and/or destroy such kernel-side objects (and in general perform allocations or de-allocations on the kernel heap).

Kernel Server requests, effectively start life as slow exec calls which code kernel server requests and send them to the kernel server thread via the Symbian OS IPC mechanism. Calls in the user library that deal with process, thread and (memory) chunk creations or deletions are all kernel server requests.

These requests involve 2 context switches, from user thread to kernel server and back (to the most high priority ready to run thread). When such calls have been issued, since they start life as slow execs, the kernel scheduler gets the opportunity to re-schedule and thus switches execution to the kernel server which needs to process the message sent from the user thread. Upon completion of such requests, the kernel scheduler will again attempt to execute the highest priority ready thread, which may or may not be the calling thread at that point.



1.6. Summary

As a modern embedded mobile OS, Symbian OS offers certain services common to all user space threads, through a so called ‘user library’, named euser.dll. All components and applications link to that shared library dynamically, from which they may utilise many services by indirectly issuing slow or fast calls to the kernel executive, as well as requests to the kernel server.

1.7. Appendix

Bellow are most such executive calls and kernel server requests that feature in the euser.dll’s header files.

`TBool User::JustInTime()`	**1.7.1. F**
`void User::SetJustInTime(const TBool aBoolean)`	F
`TUint User::Fold(TUint aChar)`	F
`TUint User::Fold(TUint aChar,TInt aFlags)`	F
`TUint User::Collate(Tuint aChar)`	F
`TUint User::LowerCase(TUint aChar)`	F
`TUint User::UpperCase(TUint aChar)`	F

TUint User::TitleCase(TUint aChar)	S
TUint TChar::GetUpperCase() const	F
TUint TChar::GetLowerCase() const	F
TUint TChar::GetTitleCase() const	S
TInt TChar::GetNumericValue() const	S
TInt TFindChunk::Next(TFullName& aResult)	S
TUint8* RChunk::Base() const	F
TInt RChunk::Size() const	F
TInt RChunk::Bottom() const	F
TInt RChunk::Top() const	F
TInt RChunk::MaxSize() const	F
TInt TFindLogicalDevice::Next(TFullName& aResult)	S
TInt TFindPhysicalDevice::Next(TFullName& aResult)	S
void RDevice::GetCaps(TDes8& aDes) const	S
TBool RDevice::QueryVersionSupported(const TVersion &aVer) const	S
TBool RDevice::IsAvailable(TInt aUnit,const TDesC* aPhysicalDevice,const TDesC* anInfo) const	S
TInt TFindLogicalChannel::Next(TFullName& aResult)	S
MBusDev::CheckOpenMode(const TDesC& aDeviceName,TInt aUnit)	S
void RBusLogicalChannel::DoRequest(TInt aReqNo,TRequestStatus& aStatus)	S
void RBusLogicalChannel::DoRequest(TInt aReqNo,TRequestStatus &aStatus,TAny* a1)	S
void RBusLogicalChannel::DoRequest(TInt aReqNo,TRequestStatus& aStatus,TAny* a1,TAny* a2)	S
void RBusLogicalChannel::DoCancel(TUint aRequestMask)	S
TInt RBusLogicalChannel::DoControl(TInt aFunction)	S
TInt RBusLogicalChannel::DoControl(TInt aFunction,TAny* a1)	S
TInt RBusLogicalChannel::DoControl(TInt aFunction,TAny* a1,TAny* a2)	S
void User::WaitForAnyRequest()	F/S
void User::WaitForRequest(TrequestStatus &aStatus)	F/S
void User::WaitForRequest(TrequestStatus &aStatus1,TRequestStatus &aStatus2)	F/S
TInt TFindLibrary::Next(TFullName& aResult)	S
TLibraryFunction RLibrary::Lookup(TInt anOrdinal) const	S
TLibraryEntry RLibrary::EntryPoint() const	S
TUint* RLibrary::DllRefTable() const	S
TFileName RLibrary::FileName() const	S
TUidType RLibrary::Type() const	S
TInt RLibrary::GetRamSizes(TInt& aCodeSize, TInt& aConstDataSize)	S
TInt User::SetHomeTime(const TTime& aTime)	S
TUint User::TickCount()	F
TTimeIntervalSeconds User::InactivityTime()	S
void User::ResetInactivityTime()	S

TUint32 User::FastCounter()	F
TTimerLockSpec User::LockPeriod()	F
TName RHandleBase::Name() const	S
TFullName RHandleBase::FullName() const	S
void RHandleBase::HandleInfo(ThandleInfo* anInfo)	S
TUint RHandleBase::Attributes() const	S
TAny* User::Adjust(TAny* aCell,TInt anOffset,TInt aDelta)	F
TAny* User::AdjustL(TAny* aCell,Tint anOffset, TInt aDelta)	F
TInt User::AllocLen(const TAny* aCell)	F
TAny* User::Alloc(TInt aSize)	F
TAny* User::AllocL(TInt aSize)	F
TAny* User::AllocLC(TInt aSize)	F
TInt User::Available(TInt& aBiggestBlock)	F
void User::Check()	F
void User::Free(TAny* aCell)	F
void User::FreeZ(TAny*& aCell)	F
TAny* User::ReAlloc(TAny* aCell,Tint aSize)	F
TAny* User::ReAllocL(TAny* aCell,Tint aSize)	F
RHeap& User::Heap()	F
RHeap::ChunkHeapCreated() const	S
TInt User::AllocSize(TInt& aTotalAllocSize)	F
TInt User::CountAllocCells()	F
TInt User::CountAllocCells(TInt& aFreeCount)	F
RHeap* User::SwitchHeap(RHeap* aHeap)	S
void TDayName::Set(TDay aDay)	S
void TDayNameAbb::Set(TDay aDay)	S
void TMonthName::Set(TMonth aMonth)	S
void TMonthNameAbb::Set(TMonth aMonth)	S
void TDateSuffix::Set(TInt aSuffix)	S
void TAmPmName::Set(TAmPm aSelector)	S
void TCurrencySymbol::Set()	S
void TShortDateFormatSpec::Set()	S
void TLongDateFormatSpec::Set()	S
void TTimeFormatSpec::Set()	S
const TFatUtilityFunctions* UserExec::FatUtilityFunctions()	F
void User::SetCurrencySymbol(const TDesC& aSymbol)	S
TLanguage User::Language()	S
TLocale::TLocale()	S
void TLocale::Refresh()	S
void TLocale::Set() const	S
void TLocaleMessageText::Set(TlocaleMessage aMsgNo)	S
TInt TFindServer::Next(TFullName& aResult)	S
void RServer::Receive(TrequestStatus& aStatus)	S
void RServer::Cancel()	S

TInt TFindMutex::Next(TFullName& aResult)	S
TInt RMutex::Count()	F
void RMutex::Wait()	F
void RMutex::Signal()	F
TInt TFindProcess::Next(TFullName& saResult)	S
TUidType RProcess::Type() const	S
void RProcess::SetType(const TuidType& aType)	S
TProcessId RProcess::Id() const	F
TProcessId RProcess::Next() const	S
TProcessId RProcess::Resume()	S
TFileName RProcess::FileName() const	S
void RProcess::CommandLine(TDes& aCommand) const	S
TInt RProcess::CommandLineLength() const	S
TExitType RProcess::ExitType() const	F
TInt RProcess::ExitReason() const	F
TExitCategoryName RProcess::ExitCategory() const	F
TProcessPriority RProcess::Priority() const	F
void RProcess::SetPriority(TprocessPriority aPriority) const	S
TBool RProcess::System() const	F
void RProcess::SetSystem(TBool aState) const	S
TBool RProcess::Protected()	F
void RProcess::SetProtected(TBool aState) const	S
TBool RProcess::LoadedFromRam() const	F
void RProcess::SetOwner(const Rprocess &aProcess) const	S
TInt RProcess::GetMemoryInfo(TprocessMemoryInfo& aInfo) const	S
TInt TFindSemaphore::Next(TFullName& aResult)	S
TInt RSemaphore::Count()	F
void RSemaphore::Wait()	F
void RSemaphore::Signal()	S
void RSemaphore::Signal(TInt aCount)	S
void RCriticalSection::Wait()	F
void RCriticalSection::Signal()	S
TInt TFindThread::Next(TFullName &aResult)	S
TThreadId RThread::Id() const	F
void RThread::HandleCount(TInt& aProcessHandleCount, TInt& aThreadHandleCount) const	S
TExceptionHandler* RThread::ExceptionHandler() const	S
TInt RThread::SetExceptionHandler(TExceptionHandler* aHandler,TUint32 aMask)	S
RThread::ModifyExceptionMask(TUint32 aClearMask, TUint32 aSetMask)	S
TInt RThread::RaiseException(TexcType aType)	S
TBool RThread::IsExceptionHandled(TExcType aType)	S
void RThread::Context(TDes8 &aDes) const	S
void RThread::Resume() const	S

void RThread::Suspend() const	S
TThreadPriority RThread::Priority() const	F
void RThread::SetProcessPriority(TProcessPriority aPriority) const	S
TProcessPriority RThread::ProcessPriority() const	F
void RThread::SetPriority(TThreadPriority aPriority) const	S
TBool RThread::System() const	F
void RThread::SetSystem(TBool aState) const	S
TBool RThread::Protected() const	F
void RThread::SetProtected(TBool aState) const	S
TInt RThread::RequestCount() const	F
TExitType RThread::ExitType() const	F
TInt RThread::ExitReason() const	F
TExitCategoryName RThread::ExitCategory() const	S
TInt RThread::GetDesLength(const Tany* aPtr) const	S
TInt RThread::GetDesMaxLength(const TAny* aPtr) const	S
void RThread::ReadL(const TAny* aPtr,TDes8& aBuf,TInt anOffset) const	S
void RThread::ReadL(const TAny* aPtr,TDes16& aBuf,TInt anOffset) const	S
void RThread::WriteL(const TAny* aPtr,const TDesC8& aBuf,TInt anOffset) const	S
void RThread::WriteL(const TAny* aPtr,const TDesC16& aBuf,TInt anOffset) const	S
void RThread::RequestComplete(TrequestStatus*& saStatus,TInt aReason) const	S
TInt RThread::GetRamSizes(TInt& aHeapSize, TInt& aStackSize)	S
TInt RThread::GetCpuTime(TtimeIntervalMicroSeconds& aCpuTime) const	S
void User::After(TtimeIntervalMicroSeconds32 anInterval)	S
TInt User::At(const TTime& aTime)	S
void RTimer::Cancel()	S
void RTimer::After(TRequestStatus& aStatus,TTimeIntervalMicroSeconds32 anInterval)	S
void RTimer::At(TRequestStatus& aStatus,const TTime& aTime)	S
void RTimer::Lock(TRequestStatus& aStatus,TTimerLockSpec aLock)	S
void RTimer::Inactivity(TrequestStatus& aStatus, TTimeIntervalSeconds aSeconds)	S
TInt RChangeNotifier::Logon(TrequestStatus& aStatus) const	S
TInt RChangeNotifier::LogonCancel() const	S
void UserSvr::CaptureEventHook()	S
void UserSvr::ReleaseEventHook()	S
void UserSvr::RequestEvent(TrawEventBuf& anEvent,TRequestStatus& aStatus)	S
void UserSvr::RequestEventCancel()	S
TInt UserSvr::AddEvent(const TrawEvent& anEvent)	S

void UserSvr::ScreenInfo(TDes8& anInfo)	S
TBool UserSvr::DllGlobalAllocated(TInt aHandle)	S
TInt UserSvr::DllGlobalRead(TInt aHandle,TInt aPos,TInt aLength,TDes8& aDes)	S
TInt UserSvr::DllGlobalWrite(TInt aHandle,TInt aPos,const TDesC8& aDes)	S
TAny* UserSvr::DllTls(TInt aHandle)	F
void UserSvr::DllFileName(TInt aHandle, TDes& aFileName)	S
void UserSvr::WsRegisterSwitchOnScreenHandling(TBool aState)	S
void UserSvr::WsSwitchOnScreen()	S
TTrapHandler* User::TrapHandler()	F
TTrapHandler* User::SetTrapHandler(TTrapHandler* aHandler)	F
TInt User::Beep(TInt aFrequency, TTimeIntervalMicroSeconds32 aDuration)	S
TInt TBusLocalDrive::Read(TInt64 aPos,TInt aLength, const TAny* aTrg,TInt aThreadHandle, TInt anOffset)	S
TInt TBusLocalDrive::Read(TInt64 aPos,TInt aLength,TDes8& aTrg)	S
TInt TBusLocalDrive::Write(TInt64 aPos,TInt aLength,const TAny* aSrc,TInt aThreadHandle,TInt anOffset)	S
TInt TBusLocalDrive::Write(TInt64 aPos,const TDesC8& aSrc)	S
TInt TBusLocalDrive::Caps(TDes8& anInfo)	S
TInt TBusLocalDrive::Format(TformatInfo& anInfo)	S
TInt TBusLocalDrive::Format(TInt64 aPos,TInt aLength)	S
TInt TBusLocalDrive::ReadPasswordData(TDes8& aBuf)	S
TInt TBusLocalDrive::PasswordStoreLengthInBytes()	S
TInt TBusLocalDrive::SetMountInfo(const TDesC8* aMountInfo,TInt aMountInfoThreadHandle)	S
TInt TBusLocalDrive::ForceRemount(TUint aFlags)	S
TInt UserSvr::HalGet(TInt aFunction, TAny* aParam)	S
TInt UserSvr::HalSet(TInt aFunction, TAny* aParam)	S
TInt UserSvr::SetMemoryThresholds(TInt aLowThreshold, TInt aGoodThreshold)	S
void User::SetDebugMask(TUint32 aVal)	F
TInt UserHal::MachineInfo(TDes8& anInfo)	S
TInt UserHal::MemoryInfo(TDes8& anInfo)	S
TInt UserHal::RomInfo(TDes8& anInfo)	S
TInt UserHal::DriveInfo(TDes8& anInfo)	S
TInt UserHal::StartupReason(TmachineStartupType& aReason)	S
TInt UserHal::FaultReason(TInt& aReason)	S
TInt UserHal::ExceptionId(TInt& anId)	S
TInt UserHal::ExceptionInfo(TexcInfo& aInfo)	S
TInt UserHal::PageSizeInBytes(TInt& aSize)	S
TInt UserHal::SwitchOff()	S
TInt UserHal::SetXYInputCalibration(const TDigitizerCalibration& aCalibration)	S

TInt UserHal::CalibrationPoints(TdigitizerCalibration& aCalibration)	S
TInt UserHal::TickPeriod(TtimeIntervalMicroSeconds32& aTime)	S
TInt UserHal::SaveXYInputCalibration()	S
TInt UserHal::RestoreXYInputCalibration(TDigitizerCalibrationType aType)	S
TInt User::MachineConfiguration(Tdes8& aConfig,TInt& aSize)	S
TInt RDebug::RegisterInfo(SregisterInfo& aInfo)	
TInt RDebug::Print(TRefByValue<const TDesC> aFmt,...)	
TBool Password::IsEnabled()	S
TBool Password::IsValid(const Tpassword& aPassword)	S
TUint32 Math::Random()	S
void User::IMB_Range(TAny* aStart, TAny* aEnd)	S
TInt RTransferBuffer::GetBufferSize() const	F
void RTransferWindow::WaitForBuffer() const	S
void RTransferWindow::WaitForBuffer(TRequestStatus& aStatus) const	S
void RTransferWindow::CancelWaitForBuffer() const	S
void RTransferWindow::WaitForBufferFree() const	S
void RTransferWindow::WaitForBufferFree(TRequestStatus& aStatus) const	S
void RTransferWindow::CancelWaitForBufferFree() const	S
TUint8* RTransferWindow::GetBufferAddress() const	F
TInt RTransferWindow::GetBufferSize() const	F
TInt RTransferWindow::MapInBuffer(const RTransferBuffer& aBuffer) const	S
void RTransferWindow::MapOutBuffer() const	S
TInt RNotifier::StartNotifier(TUid aNotifierUid, const TDesC8& aBuffer,TDes8& aResponse)	KSR
TInt RNotifier::StartNotifier(TUid aNotifierDllUid,TUid aNotifierUid,const TDesC8& aBuffer,TDes8& aResponse)	KSR
TInt RNotifier::CancelNotifier(TUid aNotifierUid)	KSR
TInt RNotifier::UpdateNotifier(TUid aNotifierUid, const TDesC8& aBuffer,TDes8& aResponse)	KSR
void RNotifier::StartNotifierAndGetResponse(TRequestStatus& aRs,TUid aNotifierUid,const TDesC8& aBuffer,TDes8& aResponse)	KSR
void RNotifier::StartNotifierAndGetResponse(TRequestStatus& aRs,TUid aNotifierDllUid,TUid aNotifierUid,const TDesC8& aBuffer,TDes8& aResponse)	KSR
TInt RNotifier::Connect()	KSR
void RNotifier::Notify(const TDesC& aLine1,const TDesC& aLine2,const TDesC& aBut1,const TdesC& aBut2, TInt& aButtonVal, TRequestStatus& aStatus)	KSR
TInt RChunk::CreateLocal(TInt aSize,TInt aMaxSize,TOwnerType aType)	KSR

TInt RChunk::CreateLocalCode(TInt aSize,TInt aMaxSize,TOwnerType aType)	KSR
TInt RChunk::CreateGlobal(const TdesC& aName,TInt aSize,TInt aMaxSize,TOwnerType aType)	KSR
TInt RChunk::CreateDoubleEndedLocal(TInt aInitialBottom, TInt aInitialTop,TInt aMaxSize,TownerType aType)	KSR
TInt RChunk::CreateDoubleEndedGlobal(const TDesC& aName,TInt aInitialBottom,TInt aInitialTop,TInt aMaxSize,TOwnerType aType)	KSR
TInt RChunk::OpenGlobal(const TDesC& aName,TBool isReadOnly,TOwnerType aType)	KSR
TInt RChunk::Adjust(TInt aNewSize) const	KSR
TInt RChunk::AdjustDoubleEnded(TInt aBottom, TInt aTop) const	KSR
TInt RDevice::Open(const TDesC& aName,TOwnerType aType)	KSR
TInt RBusLogicalChannel::DoCreate(const TDesC& aLogicalDevice,const TVersion& aVer,const TDesC* aChan,TInt aUnit,const TDesC* aPhysicalDevice,const TDesC8* anInfo,TOwnerType aType)	KSR
TInt RBusLogicalChannel::DoSvControl(TInt aFunction)	KSR
TInt RBusLogicalChannel::DoSvControl(TInt aFunction,TAny* a1)	KSR
TInt RBusLogicalChannel::DoSvControl(TInt aFunction,TAny* a1,TAny* a2)	KSR
TInt RHandleBase::Duplicate(const RThreads& aSrc,TOwnerType aType)	KSR
TInt RHandleBase::Open(const TfindHandleBase& aFindHandle,TOwnerType aType)	KSR
void RHandleBase::Close()	KSR
void RMessagePtr::Complete(TInt aReason) const	KSR
void RMessage::Complete(TInt aReason) const	KSR
TInt RServer::CreateGlobal(const TdesC &aName)	KSR
TInt RSessionBase::CreateSession(const TDesC& aServer,const TVersion& aVersion,Tint aMessageSlots)	KSR
TInt RSessionBase::Share(TattachMode aAttachMode)	KSR
TInt RSessionBase::Attach() const	KSR
TInt RMutex::CreateLocal(TOwnerType aType)	KSR
TInt RMutex::CreateGlobal(const TdesC& aName,TOwnerType aType)	KSR
TInt RMutex::OpenGlobal(const TDesC &aName,TOwnerType aType)	KSR
TInt RSemaphore::CreateLocal(TInt aCount,TOwnerType aType)	KSR
TInt RSemaphore::CreateGlobal(const TDesC& aName,TInt aCount,TOwnerType aType)	KSR
TInt RSemaphore::OpenGlobal(const TDesC& aName,TOwnerType aType)	KSR
TInt RCriticalSection::CreateLocal(TOwnerType aType)	KSR
TInt RTimer::CreateLocal()	KSR
TInt RProcess::Open(const TDesC& aName,TOwnerType aType)	KSR
TInt RProcess::Open(TProcessId aId,TOwnerType aType)	KSR
TInt RProcess::Rename(const TDesC& aName)	KSR

void RProcess::Kill(TInt aReason)	KSR
void RProcess::Terminate(TInt aReason)	KSR
void RProcess::Panic(const TDesC& aCategory,TInt aReason)	KSR
TInt RProcess::Owner(RProcess& anOwner) const	KSR
void RProcess::Logon(TRequestStatus& aStatus) const	KSR
TInt RProcess::LogonCancel(TrequestStatus& aStatus) const	KSR
TInt RThread::Create(const TDesC& aName,TThreadFunction aFunction,TInt aStackSize,TAny* aPtr,RLibrary* aLibrary,RHeap* aHeap, TInt aHeapMinSize,TInt aHeapMaxSize,TOwnerType aType)	KSR
TInt RThread::Create(const TDesC& aName,TThreadFunction aFunction,TInt aStackSize,TInt aHeapMinSize,TInt aHeapMaxSize,TAny* aPtr,TOwnerType aType)	KSR
TInt RThread::Create(const TDesC& aName,TThreadFunction aFunction,TInt aStackSize,RHeap* aHeap,TAny* aPtr,TOwnerType aType)	KSR
TInt RThread::SetInitialParameter(TAny* aPtr)	KSR
TInt RThread::Open(const TDesC& aFullName,TOwnerType aType)	KSR
TInt RThread::Open(TThreadId aId,TOwnerType aType)	KSR
TInt RThread::Process(RProcess& aProcess) const	KSR
TInt RThread::Rename(const TDesC& aName) const	KSR
void RThread::Kill(TInt aReason)	KSR
void RThread::Terminate(TInt aReason)	KSR
void RThread::Panic(const TDesC& aCategory,TInt aReason)	KSR
void RThread::Logon(TRequestStatus& aStatus) const	KSR
TInt RThread::LogonCancel(TRequestStatus& aStatus) const	KSR
RHeap* RThread::Heap()	KSR
TInt TBusLocalDrive::Connect(TInt aDriveNumber,TBool& aChangedFlag)	KSR
void TBusLocalDrive::Disconnect()	KSR
TInt TBusLocalDrive::Enlarge(TInt aLength)	KSR
TBusLocalDrive::ReduceSize(TInt aPos,TInt aLength)	KSR
TInt TBusLocalDrive::Unlock(TMediaPassword& aPassword, TBool aStorePassword)	KSR
TInt TBusLocalDrive::Lock(TMediaPassword& aOldPassword, TMediaPassword& aNewPassword, TBool aStorePassword)	KSR
TInt TBusLocalDrive::Clear(TMediaPassword& aPassword)	KSR
TInt TBusLocalDrive::WritePasswordData(TDesC8& aBuf)	KSR
void User::__DbgMarkStart(RHeap::TDbgHeapType aHeapType)	Exec for User Heaps, KSR otherwise
void User::__DbgMarkCheck(RHeap::TDbgHeapType aHeapType,TBool aCountAll,TInt aCount,const TDesC8& aFileName,TInt aLineNum)	Exec for User Heaps, KSR otherwise
TUint32 User::__DbgMarkEnd(RHeap::TDbgHeapType aHeapType,TInt aCount)	Exec for User Heaps, KSR otherwise
void User::__DbgSetAllocFail(RHeap::TDbgHeapType	Exec for User

aHeapType,RHeap::TAllocFail aType,TInt aValue)	Heaps, KSR otherwise
TInt RProcess::Create(const TDesC& aFileName,const TDesC &aCommand,TOwnerType aType)	KSR
TInt RProcess::Create(const TDesC& aFileName,const TDesC& aCommand,const TUidType& aUidType, TOwnerType aType)	KSR
TInt User::LoadLogicalDevice(const TDesC& aFileName)	KSR
TInt User::FreeLogicalDevice(const TDesC& aDeviceName)	KSR
TInt User::LoadPhysicalDevice(const TDesC& aFileName)	KSR
TInt User::FreePhysicalDevice(const TDesC& aDeviceName)	KSR
TInt RLoader::Connect()	KSR
TInt RLoader::LoadLibrary(TInt& aHandle,const TDesC& aFileName,const TDesC& anExt, const TDesC& aPath, const TUidType& aType)	KSR
TInt UserSvr::ProcessCreate(TLoaderInfo& anInfo, HBufC* aCommand)	KSR
void UserSvr::ProcessLoaded(TLoaderInfo& anInfo)	KSR
TInt UserSvr::LibraryCreateExact(TLoaderInfo& anInfo)	KSR
TInt UserSvr::LibraryLoaded(TLoaderInfo& anInfo)	KSR
TInt UserSvr::DllSetTls(TInt aHandle,TAny* aPtr)	KSR
void UserSvr::DllFreeTls(TInt aHandle)	KSR
TInt UserSvr::DllInitialiseData(TInt aHandle)	KSR
void UserSvr::DllFreeData(TInt aHandle)	KSR
TInt RChangeNotifier::Create()	KSR
TInt RUndertaker::Create()	KSR
TInt RUndertaker::Logon(TRequestStatus& aStatus, TInt& aThreadHandle) const	KSR
TInt RUndertaker::LogonCancel() const	KSR
TInt UserSvr::DllGlobalAlloc(TInt aHandle,TInt aSize)	KSR
TInt User::SetMachineConfiguration(const TDesC8& aConfig)	KSR
TInt User::CompressAllHeaps()	KSR
TInt RDebug::Open(TInt aMaxBreak,TInt aMaxWatch,TInt aMaxPanic,TUint aDebugLimit)	KSR
TInt RDebug::Close(	KSR
TInt RDebug::KillThread(const TThreadId aId)	KSR
TInt Password::SetEnabled(const TPassword& aPassword,TBool aIsEnabled)	KSR
TInt Password::Set(const TPassword& anOldPassword,const TPassword& aNewPassword)	KSR
void UserSvr::ForceRemountMedia(TMediaDevice aDevice)	KSR
TInt UserSvr::MediaChangeNotify(TMediaDevice aDevice,TRequestStatus* aReqStat)	KSR
TInt UserSvr::ChangeLocale(RLibrary aLibrary)	KSR
TInt UserSvr::DllAddDependency(TAny* anImportingDll, TAny* anExportingDll)	KSR
TInt UserSvr::ExeAddDependency(TAny* anImportingExe, TAny* anExportingDll)	KSR
TInt UserSvr::ResetMachine(TMachineStartupType aType)	KSR

`TInt RTransferBuffer::Create(TInt aBufferSize)`	KSR
`TInt RTransferWindow::Create(TInt aMaxBufferSize)`	KSR
`TInt RTransferWindow::Open(RTransferBuffer& aBuffer)`	KSR

Want to be kept informed of new articles being made available on the Symbian Developer Network?
Subscribe to the Symbian Community Newsletter.
The Symbian Community Newsletter brings you, every month, the latest news and resources for Symbian OS.