NtAddAtom adds the character string to the global atom table and returns an atom value.

PARAMETERS

pString	Pointer to the wide character string to be added to the global atom table.
pStringLength	Length of the string pointed to by pString.
pAtom	Pointer to the variable that receives the atom value corresponding to the string.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

The memory for the atom table is allocated by WIN32K.SYS. The system service gets the pointer to this memory using a callout established by WIN32K.SYS. This callout is established by WIN32K.SYS using the PspEstablishWin32Callouts function from NTOSKRNL. The system service uses this callout to get the pointer to the global atom table and then manipulates the atom table using its internal functions, such as RtlAddAtomToAtomTable. These internal functions also reside in user-mode DLL NTDLL.DLL. These NTDLL routines are used to support local atom tables (per process). It seems that these internal functions in NTOSKRNL and NTDLL are shared at the source code level because their binary equivalents are identical. The user-mode API call GlobalAddAtom uses this system service. The AddAtom function does not call this system service; instead, it manipulates the local atom table using internal functions in NTDLL.

The second parameter (pStringLength) is introduced starting from Windows 2000. Previous versions of Windows NT take only two parameters pString and pAtom.

EQUIVALENT WIN32 API

GlobalAddAtom

```
NtQuerylnformationAtom
NTSTATUS
     NtQuerylnformationAtonH
          IN ATOM Atom,
          IN ATOM_INFO_CLASS AtomlnfoClass,
          OUT  PVOID  AtomlnfoBuffer,
          IN ULONG AtomlnfoBufferLength,
          OUT  PULONG  BytesCopied
);
```

NtQuerylnformationAtom returns information about specific/all atom objects in the global atom table.

PARAMETERS

Atom	The atom ID returned by NtAddAtom, NtFindAtom, and so on.
AtomlnfoClass	The type of information requested. This value can be Oor 1.
AtomlnfoBuffer	Pointer to the buffer that receives the information about the atoms.
AtomlnfoBufferLength	Size of the buffer in bytes pointed to by AtomlnfoBuffer.
BytesCopied	Pointer to the variable that receives the number of bytes copied into AtomlnfoBuffer. However, this variable is not set by the system service.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

If the AtomlnfoClass is 1, then the Atom field is ignored and the system service returns information about all the objects in the global atom table. If the AtomlnfoClass is 0, then the system service returns the information about the atom specified by the Atom parameter.

For AtomlnfoClass 0, the data in AtomlnfoBuffer is laid out as follows:

```
typedef struct AtomInfoSingle {
WORD ReferenceCount;
WORD Unknown;
WORD AtomStringLength;
WCHAR AtomStnng[l];
) ATOMINFOSINGLE, *P ATOMINFOSINGLE;
```

The size of data returned varies based on the size of the atom string.

For AtomlnfoClass 1, the data in AtomlnfoBuffer is laid out as follows:

```
typedef struct AtomlnfoAl1 (
DWORD TotalNumberOfEntriesInGlobalAtomTable;
ATOM  AtomValues[l];
} ATOMINFOALL, *PATOMINFOALL;
```

The size of data returned varies based on the number of entries in the global atom table.

The user-mode API call can get the atom string corresponding to the atom ID. Other information is available only through this system service.

EQUIVALENT WIN32 API
GlobalGetAtomName

NtFindAtom

```
NTSTATUS
     NtFindAtom(
          IN PWCHAR pString,
          IN ULONG pStringLength,
          OUT PATOM pAtom
);
```

NtFindAtom retrieves the atom corresponding to the specified string in the global atom table.

PARAMETERS

pString	Pointer to the wide character string to be searched in the global atom table.
pStringLength	Length of the string pointed to by pString.
pAtom	Pointer to the variable that receives the atom value corresponding to the string.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
The second parameter (pStringLength) is introduced starting from Windows 2000. Previous versions of Windows NT take only two parameters, pString and pAtom.

EQUIVALENT WIN32 API
GlobalFindAtom

NtDeleteAtom

```
NTSTATUS
     NtDeleteAtom(
          IN ATOM Atomld.
);
```

NtDeleteAtom decrements the reference count for the specified atom. If the reference count reaches zero, it deletes the atom from the global atom table.

PARAMETERS

Atomld Atom to be deleted.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
GlobalDeleteAtom

NtCreateKey

```
NTSTATUS
    NtCreateKey(
        OUT PHANDLE phKey,
        IN  ACCESS_MASK  DesiredAccess ,
        IN  POBJECT_ATTRIBUTES ObjectAttributes,
        IN  ULONG TitleIndex,
        IN  PUNICODE_STRING Class,
        IN   ULONG  CreateOptions,
        OUT PULONG pDisposition
);
```

NtCreateKey creates a new Registry key or opens the Registry key if it is already present.

PARAMETERS

phKey	Pointer to the variable that receives handle to the Registry key object.
AccessMask	Type of access requested to the Registry key object. This could be KEY_QUERY_VALUE, KEY_SET_VALUE, KEY_CREATE_SUBKEYS, KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY, or KEY_CREATE_LINK, or set of standard rights such as KEY_READ, KEY_WRiTE, KEY_EXECUTE, or KEY_ALL_ACCESS.

ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the key to be created, such as name, parent directory, objectflags, and so on.
TitleIndex	This parameter should be set to 0.
Class	Points to the object class of the key. This parameter is ignored if the key already exists.
CreateOptions	Specifies the options to be applied while creating the key. This could be REG_OPTIONAL_VOLATILE, REG_OPTION_NON_VOLATILE, REG_OPTION_CREATE_LINK, or REG_OPTION_BACKUP_RESTORE.
pDisposition	Pointer to the variable that receives whether the new key is created (REG_CREATED_NEW_KEY) or the existing key is opened (REG_OPENED_EXISTING_KEY).

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

None.

EQUIVALENT WIN32 API

RegCreateKey, RegCreateKeyEx

```
NtOpenKey
NTSTATUS
     NtOpenKey(
          OUT PHANDLE phKey,
          IN ACCESS_MASK DesiredAccess,
          IN POBJECT_ATTRIBUTES ObjectAttnbutes
);                                                    j
```

NtOpenKey opens an existing Registry key.

PARAMETERS

phKey	Pointer to the variable that receives handle to the Registry key object.

AccessMask	Type of access requested to the Registry key object. This could be KEY_QUERY_VALUE, KEY_SET_VALUE, KEY_CREATE_SUBKEYS, KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY, or KEY_CREATE_LINK, or set of standard rights such as KEY_READ, KEY_WRITE, KEY_EXECUTE, or KEY_ALL_ACCESS.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the key to be opened, such as name, parent directory, objectflags, and so on.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
RegOpenKey, RegOpenKeyEx

```
NtDeleteKey
NTSTATUS
     NtDeleteKey(
          IN HANDLE hKey
);
```

NtDeleteKey deletes the Registry key specified by hKey.

PARAMETERS

hKey Handle to the open Registry key.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
RegDeleteKey

```
NtDeleteValueKey
NTSTATUS
```

```
NtDeleteValueKey(
    IN HANDLE hKey,
    IN PUNICODE_STRING pValueName
);
```

NtDeleteValueKey deletes the value specified by pValueName under the Registry key specified by hKey.

PARAMETERS

hKey	Handle to the open Registry key.
PValueName	Pointer to the Unicode string containing the name of the value to be deleted. If this parameter is an empty string, the system service deletes the default unnamed value under the Registry key.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

None.

EQUIVALENT WIN32 API

RegDeleteValue

NtEnumerateKey

```
NTSTATUS
NtEnumerateKey(
    IN HANDLE hKey,
    IN ULONG Index,
    IN KEY_INFORMATION_CLASS KeyInfoClass,
    OUT PVOID KeyInfoBuffer,
    IN ULONG KeyInfoBufferLength,
    OUT PULONG BytesCopied
);
```

NtEnumerateKey retrieves information about subkeys of an existing open key.

PARAMETERS

hKey	Handle to the open Registry key.
Index	Zero-based index of the subkey for which the information is to be retrieved.

KeylnfoClass	The information class requested for the subkey. This can be KeyBasicInformation, KeyNodelnformation, or KeyFullInformation.
KeylnfoBuffer	Pointer to the buffer that receives the information about the subkey.
KeylnfoBufferLength	Size of the buffer in bytes pointed to by KeylnfoBuffer.
BytesCopied	Pointer to the variable that receives the number of bytes copied into KeylnfoBuffer.

RETURN VALUE

Returns STATUS_SUCCESS on success, STATUS_NO_MORE_ENTRIES when all the entries are over, and an appropriate error code on failure.

COMMENTS

The layout of the buffer returned is based on the information class; that is, if the KeylnfoClass is KeyBasicInformation, the information in the KeylnfoBuffer is according to the structure definition KEY_BASIC_INFORMATION. If the KeylnfoClass is KeyNodelnformation, the KeylnfoBuffer is according to the structure definition of KEY_NODE_INFORMATION. And if the KeylnfoClass is KeyFullInformation, the KeylnfoBuffer is according to the structure definition of KEY_FULL_INFORMATION. If the passed KeylnfoBuffer is not enough to hold the requested information, the system service returns the number of bytes required to hold the information in the BytesCopied variable.

EQUIVALENT WIN32 API

RegEnumKey, RegEnumKeyEx

NtEnumerateValueKey

```
NTSTATUS
     NtEnumerateValueKey(
          IN HANDLE hKey,
          IN ULONG Index,
          IN KEY_VALUE_INFORMATION_CLASS KeyValueInfoClass,
          OUT PVOID KeyValueInfoBuffer,
          IN ULONG KeyValueInfoBufferLength,
          OUT PULONG BytesCopied
);
```

NtEnumerateValueKey retrieves information about the value entries of an existing open key.

PARAMETERS

hKey	Handle to the open Registry key.
Index	Zero-based index of the valuename for which the information is to be retrieved.
KeyValuelnfoClass	The information class requested for the valuename. This can be KeyValueBasicInformation, KeyValueFullInformation, or KeyValuePartiallnformation.
KeyValuelnfoBuffer	Pointer to the buffer that receives the information about the valuename.
KeyValuelnfoBufferLength	Size of the buffer in bytes pointed to by KeyValuelnfoBuffer.
BytesCopied	Pointer to the variable that receives the number of bytes copied into KeyValuelnfoBuffer.

RETURN VALUE

Returns STATUS_SUCCESS on success, STATUS_NO_MORE_ENTRIES when all the entries are over, and an appropriate error code on failure.

COMMENTS

The layout of the buffer returned is based on the information class; that is, if the KeyValuelnfoClass is KeyValueBasicInformation, the information in the KeyValuelnfoBuffer is according to the structure definition of KEY_VALUE_BASIC_INFORMATION. If the KeylnfoClass is KeyValueFullInformation, the KeyValuelnfoBuffer is according to the structure definition of KEY_VALUE_FULL_INFORMATION. And if the KeyValuelnfoClass is KeyPartiallnformation, the KeyValuelnfoBuffer is according to the structure definition of KEY_VALUE_PARTIAL_INFORMATION. If the passed KeyValuelnfoBuffer is not enough to hold the requested information, the system service returns the number of bytes required to hold the information in the BytesCopied variable.

EQUIVALENT WIN32 API

RegEnumValue

NtFlushKey

```
NTSTATUS
     NtFlushKey(
          IN HANDLE hKey
);
```

NtFlushKey syncs (commits) the data under the specified Registry key to disk. -

PARAMETERS

hKey Handle to the open Registry key.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
This function is very costly because it synchronously flushes the changes made to the Registry. This system service should be used only when it is absolutely necessary to do so. The system automatically flushes the Registry periodically.

EQUIVALENT WIN32 API
RegFlushKey

NtInitializeRegistry

```
NTSTATUS
      NtInitializeRegistry(
            IN  DWORD  UnknownParam
);
```

NtInitializeRegistry initializes the Registry.

PARAMETERS

UnknownParam This parameter does not seem to be used.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
This function is called by SMSS.EXE during the early boot sequence. The single parameter passed to it does not seem to be used. SMSS passes 0 for this parameter. This function does the work only the first time it is called. NTOSKRNL sets its internal variable CMFirstTime to 0 once the first call to this service is made. NTOSKRNL does the registry initialization only if CMFirstTime is set to 1.

EQUIVALENT WIN32 API
None.

NtNotifyChangeKey

```
NTSTATUS
     NtNotifyChangeKey(
          IN HANDLE hKey,
          IN HANDLE hEvent,
          IN PIO_APC_ROUTINE ApcRoutine,
          IN PVOID ApcRoutineContext,
          IN ULONG NotifyFilter,
          IN BOOLEAN bWatchSubtree,
          OUT PVOID RegChangesDataBuffer,
          IN ULONG RegChangesDataBufferLength,
          IN BOOL bAynchronous
);
```

NtNotifyChangeKey monitors the Registry changes under the specified Registry key and optionally calls an APC routine when the changes occur.

PARAMETERS

hKey	Handle to the Registry key to monitor. Changes under this Registry key are monitored.
hEvent	Handle to the event object. This parameter is ignored if bAsynchronous is FALSE. If bAsynchronous is TRUE, this event object is signaled when the changes occur under the specified Registry key.
ApcRoutine	Pointer to the function that gets called when the Registry changes occur.
ApcRoutineContext	Parameter to be passed to the ApcRoutine.
NotifyFilter	Set of flags specifying what changes to monitor. This could be all or any combination of the following: REG_N01TFY_CHANGE_NAME, REG_NOTIFY_CHANGE_ATrRIBUTES, REG_NOTIFY_CHANGE_LAST_SET, and REG_NOTIFY_CHANGE_SECURlTY.
bWatchSubtree	Flag specifying whether to monitor all subkeys under the specifed hKey also.
RegChangesDataBuffer	Pointer to the buffer that receives the changes made to the Registry.

RegChangesDataBufferLength	Size of the buffer pointed to by RegChangesDataBuffer.
bAsynchronous	Flag specifying whether the monitoring should be done synchronously or asynchronously.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

This function is similar to NtNotifyChangeDirectoryFile. It monitors the Registry tree instead of the file system tree. Presently, the RegChangesDataBuffer and RegChangesDataBufferLength parameters are ignored. Hence, the notification is given only when the Registry changes occur. The function does not return in RegChangeDataBuffer the actual changes made in the Registry. This is different from NtNotifyChangeDirectoryFile, which returns actual changes also. We feel that the functionality to provide the actual list of changes to the Registry will be added in future versions of Windows NT. For now, this system service does not provide any more information than its Equivalent Win32 API RegNotifyChangeKeyValue except for the fact that you can specify a callback function that gets called when the changes occur in the Registry.

EQUIVALENT WIN32 API

RegNotifyChangeKeyValue

NtQueryKey

```
NTSTATUS
    NtQueryKey(
        IN HANDLE hKey,
        IN KEY_INFORMATION_CLASS KeyInfoClass.
        OUT PVOID KeyInfoBuffer,
        IN ULONG KeyInfoBufferLength,
        OUT PULONG BytesCopied
);
```

NtQueryKey retrieves information about the specified key.

PARAMETERS

hKey	Handle to the key for which the information is requested.

KeylnfoClass	The information class requested for the key. This can be KeyBasicInformation, KeyNodelnformation, or KeyFullInformation.
KeylnfoBuffer	Pointer to the buffer that receives the information about the key.
KeylnfoBufferLength	Size of the buffer (in bytes) pointed to by KeylnfoBuffer.
BytesCopied	Pointer to the variable that receives the number of bytes copied into KeylnfoBuffer.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

Look at the Comments section for the NtEnumerateKey function.

EQUIVALENT WIN32 API

RegQueryKey

NtQueryMultipleValueKey

```
NTSTATUS
    NtQueryMultipleValueKey(
        IN HANDLE hKey,
        IN OUT PKEY_VALUE__ENTRY ValueNameArray,
        IN DWORD nElementsValueNameArray,
        OUT PVOID ValueDataBuffer,
        IN OUT PULONG ValueDataBufferSize.
        OUT PULONG SizeRequired
);
```

NtQueryMultipleValueKey retrieves the data and the type of information about the specified values.

PARAMETERS

hKey	Handle to the key to which the values belong.
ValueNameArray	Pointer to the array of KEY_VALUE_ENTRY structures. The first member of this should be filled in with a pointer to the Unicode string representation of the valuename whose information is to be retrieved. The other members of this structure are returned upon the successful execution of the service.

nElementsValueNameArray	The number of KEY_VALUE_ENTRY type entries in the array pointed to by ValueNameArray.
ValueDataBuffer	Pointer to the buffer that receives the data associated with the values specified in ValueNameArray. The DataOffset field in KEY_VALUE_ENTRY points to offsets in this buffer upon successful execution of the service.
ValueDataBufferSize	Pointer to the variable that contains the size of the buffer pointed to by ValueDataBuffer. This variable receives the number of bytes actually copied to the ValueDataBuffer upon successful execution of the service.
SizeRequired	Pointer to the variable that receives the size of the buffer required to fulfill the query request. This member can be used to allocate appropriate space if the function fails with STATUS BUFFER OVERFLOW.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
RegQueryMultipleValues

NtQueryValueKey

```
NTSTATUS
    NtQueryValueKey(
        IN HANDLE hKey,
        IN PUNICODE_STRING uValueName,
        IN KEY_VALUE_INFORMATION_CLASS KeyValueInfoClass,
        OUT PVOID KeyValueInfoBuffer,
        IN ULONG KeyValueInfoBufferLength,
        OUT PULONG BytesCopied
);
```

NtQueryValueKey retrieves information about the specified value.

PARAMETERS

hKey	Handle to the key to which the valuename belongs.
uValueName	Pointer to the Unicode string representation of the valuename.
KeyValuelnfoClass	The information class requested for the Registry value. This can be KeyValueBasicInformation, KeyValueFullInformation, or KeyValuePartiallnformation.
KeyValuelnfoBuffer	Pointer to the buffer that receives the information about the Registry value.
KeyValuelnfoBufferLength	Size of the buffer (in bytes) pointed to by KeyValuelnfoBuffer.
BytesCopied	Pointer to the variable that receives the number of bytes copied into KeyValuelnfoBuffer.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
Look at the Comments section for the NtEnumerateValueKey function.

EQUIVALENT WIN32 API /
RegQueryValueEx

NtReplaceKey

```
NTSTATUS
    NtReplaceKey(
        IN POBJECT_ATTRIBUTES NewHiveFile,
        IN HANDLE hKey,
        IN POBJECT_ATTRIBUTES BackupHiveFile
):
```

NtReplaceKey replaces the hive file backing up the existing key with a new hive file.

PARAMETERS

NewHiveFile	Pointer to the object attribute structure describing the new hive file.

hKey	Handle to root key of the hive.
BackupHiveFile	Pointer to the object attribute structure describing the backup hive file that receives the copy of the existing Registry hive.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
RegReplaceKey

```
NtSaveKey
NTSTATUS
     NtSaveKey(
          IN HANDLE hKey,
          IN HANDLE hFile
);
```

NtSaveKey backs up the Registry contents under the specified Registry key into a file.

PARAMETERS

hKey	Handle to the key to backup.
hFile	Handle to the file in which the contents of the Registry are to be saved.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
RegSaveKey

```
NtRestoreKey
NTSTATUS
     NtRestoreKey(
```

```
        IN HANDLE hKey,
        IN HANDLE hFile,
        IN ULONG Flags
);
```

NtRestoreKey restores the Registry contents under the specified Registry key from the specified file.

PARAMETERS

hKey	Handle to the key to restore.
hFile	Handle to the file containing the Registry data.
Flags	This parameter can be REG_WHOLE_HIVE_VOLATILE.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

None.

EQUIVALENT WIN32 API

RegRestoreKey

NtSetInformationKey

```
NTSTATUS
    NtSetInformationKey(
        IN HANDLE hKey,
        IN KEY_SET_INFORMATION_CLASS KeySetInfoClass,
        IN PKEY_WRITE_TIME_INFORMATION pInfoBuffer,
        IN ULONG pInfoBufferLength
);
```

NtSetInformationKey sets the last write time of the specified key.

PARAMETERS

hKey	Handle to the Registry key.
KeySetInfoClass	This parameter should be KeyWriteTimeInformation.
pInfoBuffer	Pointer to the structure of type KEY_WRITE_TIME_INFORMATION. The structure contains only one large integer field called LastWriteTime.
pInfoBufferLength	Size of the buffer pointed to by pInfoBuffer.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
None.

```
NtSetValueKey
NTSTATUS
      NtSetValueKey(
            IN HANDLE hKey,
            IN PUNICODE_STRING uValueName,
            IN ULONG TitleIndex,
            IN ULONG ValueType,
            IN  PVOID  pValueData,
            IN   ULONG   pValueDataLength
);
```

NtSetValueKey sets/creates a value entry under the specified Registry key.

PARAMETERS

hKey	Handle to the Registry key to which the value is associated with.
uValueName	Pointer to the Unicode string containing the valuename.
Titlelndex	This parameter should be 0.
ValueType	Data type for the value. This could be REG_BINARY, REG_DWORD, and so on. For all types, refer to the documentation of the RegSetValueEx call.
pValueData	Pointer to the buffer containing the data to be associated with the valuename.
pValueDataLength	Size of the data buffer pointed to by pValueData.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
RegSetValueEx

```
NtLoadKey
NTSTATUS
     NtLoadKey(
          IN POBJECT_ATTRIBUTES KeyNameAttnbutes,
          IN POBJECT_ATTRIBUTES HiveFi1eNameAttnbutes
);
```

NtLoadKey loads the specified Registry hive on top of the existing Registry key.

PARAMETERS

KeyNameAttributes	Pointer to the object attributes structure describing the key on which the hive is to be loaded.
HiveFileNameAttributes	Pointer to the object attributes structure describing the hive filename.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
Refer to the documentation of the NtLoadKey2 system service. NtLoadKey internally calls NtLoadKey2 by specifying the third parameter flags as 0.

EQUIVALENT WIN32 API
RegLoadKey

```
NtLoadKey2
NTSTATUS
     NtLoadKey2(
          IN POBJECT_ATTRIBUTES KeyNameAttributes,
          IN POBJECT_ATTRIBUTES HiveFi1eNameAttributes,
          IN ULONG Flags
);
```

NtLoadKey2 loads the specified Registry hive on top of the existing Registry key and applies the specified flags for the loaded hive.

PARAMETERS

KeyNameAttributes	Pointer to the object attributes structure describing the key on which the hive is to be loaded.
HiveFileNameAttributes	Pointer to the object attributes structure describing the hive filename.
Flags	The only flag value allowed is 0x00000004.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
The hive loaded with NtLoadKey is periodically synched by the system worker thread. With NtLoadKey2, you can specify a flag 0x00000004 that turns off this periodic synching. Hence, if this flag is specified, the hive is not synched periodically by the system worker thread. Internally, the system maintains a linked list of loaded hives data structures. One member of this data structure contains the information about the flags passed while loading the hive. The system worker thread periodically walks the list of loaded hives and, based on this member, decides whether to synch the hive or not.

EQUIVALENT WIN32 API
None.

```
NtUnloadKey
NTSTATUS
      NtUnloadKey(
            IN  POBJECT_ATTRIBUTES  KeyNameAttributes
);                          .    -
```

NtUnloadKey unloads the hive loaded on top of the existing key (using NtLoadKey2/NtLoadKey).

PARAMETERS

KeyNameAttributes	Pointer to the object attributes structure describing the key from which the hive is to be unloaded.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
RegUnloadKey

NtAlertResumeThread ' «-_-

```
NTSTATUS
    NtAlertResumeThread(
        IN HANDLE hThread,
        OUT PBOOLEAN pbResumed
);
```

NtAlertResumeThread resumes the thread or alerts the thread that is in alertable wait state.

PARAMETERS

hThread	Handle to the thread.
pbResumed	Pointer to the variable that receives whether the thread was actually resumed or it was already running (TRUE if the thread was in the suspended state at the time of function call, otherwise FALSE).

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
If the thread is in alertable wait state, then this function takes the thread out of the wait state with the status of STATUS_ALERTED. However, the user mode alertable wait calls such as WaitForSingleObject and SleepEx put the thread again back in alertable wait state if they return with the status of STATUS_ALERTED.

EQU1VALEMT WIN32 API
None.

NtAlertThread

```
NTSTATUS
    NtAlertThread(
        IN HANDLE hThread
);
```

NtAlertThread alerts the thread that is in alertable wait state.

PARAMETERS

hThread Handle to the thread.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
See the Comments section for NtAlertResumeThread.

EQUIVALENT WIN32 API
None.

NtTestAlert

```
NTSTATUS
     NtTestAlert (
);
```

NtTestAlert checks whether the current thread has any pending alerts.

PARAMETERS
None.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
None.

NtCreateProcess

```
NTSTATUS
     NtCreateProcess(
          OUT PHANDLE phProcess,
          IN ACCESS_MASK AccessMask,
          IN POBJECT_ATTRIBUTES ObjectAttributes,
          IN HANDLE hParentProcess,
          IN BOOLEAN blnheritHandles,
          IN HANDLE hSection.
          IN HANDLE hDebugPort.
          IN HANDLE hExceptionPort
);
```

NtCreateProcess creates a new process object.

PARAMETERS

phProcess	Pointer to the variable that receives handle to the process object.
AccessMask	Type of access requested to the process object.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the process object to be created, such as name, parent directory, objectflags, and so on.
hParentProcess	Handle to the process object that this process will be a child of.
blnheritHandles	The flag specifying whether the handles from the parent process described by hParentProcess are to be inherited by this process.
hSection	Handle to the section object created for the executable file.
hDebugPort	Handle to the debug port for the process.
hExceptionPort	Handle to the exception port for the process.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

The system service enables you to specify a different parent process for the process to be created, whereas in case of the user-mode CreateProcess call, the caller of the function becomes the parent of the process being created. The reason for this is that CreateProcess, by default, passes OxFFFFFFFF as the parent process handle. We feel that the provision to specify a different process handle than that of the caller is required in the POSIX subsystem where the fork call is actually implemented by the subsystem. Also note that unlike all other synchronization objects, such as mutex and semaphore, the CreateProcess call does not enable you to name the process object, whereas the system service enables you to specify the name of the process object in the ObjectAttributes structure. This way, you can open the process using its name instead of the process ID that is required for the OpenProcess call. Further, the system service enables you to specify specific port handles for DebugPort and ExceptionPort, whereas the CreateProcess call does not allow you to do so. The CreateProcess call, by default, passes NULL handles to these two parameters. When NULL handles are passed, the NtCreateProcess call uses the CSRSS's port objects for debug port and exception port.

EQUIVALENT WIN32 API

CreateProcess

```
NtCreateThread
NTSTATUS
     NtCreateThread(
          OUT PHANDLE phThread,                « ,.
          IN ACCESS_MASK AccessMask,
          IN POBJECT_ATTRIBUTES ObjectAttributes,
          IN HANDLE hProcess,
          OUT PCLIENT_ID pClientId,
          IN PCONTEXT pContext,
          OUT PSTACKINFO pStacklnfo,
          IN BOOLEAN bSuspended
);
```

NtCreateThread creates a new thread object.

PARAMETERS

phThread	Pointer to the variable that receives handle to the thread object.
AccessMask	Type of access requested to the thread object.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the thread object to be created, such as name, parent directory, objectflags, and so on.
hProcess	Handle to the process object that this thread will belong to.
pClientld	Pointer to the structure that will receive the thread ID and the process ID for this thread.
pContext	Pointer to the CONTEXT structure containing the state of various processor registers when the thread begins executing.
PStacklnfo	Pointer to the variable that receives the information about the stack of the thread.
bSuspended	Flag indicating whether the thread should be in suspended mode.

RETURM VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

Note that unlike all other synchronization objects, such as mutex and semaphore, the Equivalent Win32 API CreateThread call does not enable you to name the

thread object, whereas the system service enables you to specify the name of the thread object in the ObjectAttributes structure. This way, you can open the thread using its name instead of the thread ID.

The structure PSTACKINFO is defined as follows:

```
typedef struct StackInfo_t {
     LONG  Unknownl;
     LONG Unknown2;
     LONG  TopOfStack;
     LONG   OnePageBelowTopOfStack;
     LONG   BottomOfStack;
} STACKINFO, *PSTACKINFO;
```

EQUIVALENT WIN32 API

CreateThread

NtDelayExecution

```
NTSTATUS
     NtDelayExecution(
          IN BOOLEAN bAlertable,
          IN  PLARGE_INTEGER  pDuration
);
```

NtDelayExecution puts the calling thread in alertable/nonalertable sleep state for the specified duration.

PARAMETERS

bAlertable	Flag specifying whether the sleep is alertable/nonalertable.
pDuration	Pointer to the LARGEJNTEGER structure containing the duration for sleep. Negative values represent relative time, whereas positive values represent absolute time.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

None.

EQUIVALENT WIN32 API

SleepEx

NtGetContextThread

```
NTSTATUS
     NtGetContextThread(
          IN HANDLE hThread.
          IN OUT PCONTEXT pContext
);
```

NtGetContextThread returns the context structure for the specified thread.

PARAMETERS

hThread	Handle to the thread object.
pContext	Pointer to the structure that contains new context structure for the thread. The ContextFlags field in this structure must be filled with any combination of CONTEXT_CONTROL, CONTEXTJNTEGER, CONTEXT_SEGMENTS, CONTEXT_FLOATING_POINT, CONTEXT_DEBUG_REGISTERS, CONTEXT_EXTENDED_ REGISTERS, and CONTEXT_FULL based on the type of context information filled before invoking this system service.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

None.

EQUIVALENT WIN32 API

GetThreadContext

NtSetContextThread

```
NTSTATUS
     NtSetContextThread(
          IN HANDLE hThread,
          IN PCONTEXT pContext
);
```

NtSetContextThread sets the context structure for the specified thread.

PARAMETERS

hThread	Handle to the thread object.

pContext	Pointer to the structure that receives the context structure for the thread. The ContextFlags field in this structure must be filled with any combination of CONTEXT_CONTROL, CONTEXTJNTEGER, CONTEXT_SEGMENTS, CONTEXT_FLOATING_POINT, CONTEXT_DEBUG_REGISTERS, CONTEXT_EXTENDED_ REGISTERS, and CONTEXT_FULL before invoking this system service.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

None.

EQUIVALENT WIN32 API

SetThreadContext

NtOpenProcess

```
NTSTATUS
    NtOpenProcess(
        OUT PHANDLE phProcess,
        IN ACCESS_MASK AccessMask,
        IN POBJECT_ATTRIBUTES ObjectAttnbutes,
        IN PCLIENT_ID pClientId
);
```

NtOpenProcess opens a handle to the existing process object.

PARAMETERS

phProcess	Pointer to the variable that receives handle to the process object.
AccessMask	Type of access requested to the process object.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the process object to be opened, such as name, parent directory, objectflags, and so on.
pClientld	Pointer to the CLIENTJD structure. The process ID member of this structure must be filled by the caller before invoking this system service.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
Only one of the parameters ObjectAttributes and pClientld may be specified. If the process object is named, then the ObjectAttribute structure must be filled with the details; otherwise, the process ID field must be filled in the pClientld structure.

EQUIVALENT WIN32 API
OpenProcess

NtOpenThread

```
NTSTATUS
    NtOpenThread(
        OUT PHANDLE phThread,
        IN ACCESS_MASK AccessMask,
        IN POBJECT_ATTRIBUTES ObjectAttributes,
        IN PCLIENT_ID pClientId
);
```

NtOpenThread opens handle to the existing thread object.

PARAMETERS

phThread	Pointer to the variable that receives handle to the thread object.
AccessMask	Type of access requested to the thread object.
ObjectAttributes	Points to the OBJECT_ATTRIBUTES structure containing the information about the thread object to be opened, such as name, parent directory, objectflags, and so on.
pClientld	Pointer to the CLIENTJD structure. The thread ID member of this structure must be filled by the caller before invoking this system service.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
Only one of the parameters ObjectAttributes and pClientld may be specified. If the thread object has a name, then the ObjectAttribute structure must be filled with the details; otherwise, the thread ID field must be filled in the pClientld structure.

EQUIVALENT WIN32 API
None.

NtQueryInformationProcess

```
NTSTATUS
     NtQueryInformationProcess(
          IN HANDLE hProcess,
          IN PROCESSINFOCLASS ProcessInfoClass,
          OUT PVOID ProcessInfoBuffer,
          IN ULONG ProcessInfoBufferLength,
          OPTIONAL OUT PULONG BytesCopied.
);
```

NtQuerylnformationProcess returns information about the specified process object.

PARAMETERS

hProcess	Handle to the process object.
ProcessInfoClass	Type of information requested.
ProcessInfoBuffer	Pointer to the buffer that receives the information about the process object.
ProcessInfoBufferLength	Size of the buffer (in bytes) pointed to by ProcessInfoBuffer.
BytesCopied	Pointer to the variable that receives the number of bytes copied into ProcessInfoBuffer.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

Different information is returned based upon the ProcessInfoClass parameter. Here is the layout of ProcessInfoBuffer based on the ProcessInfoClass:

ProcessBasicInformation(O)	PROCESS_BASIC_INFORMATION.
ProcessQuotaLimits(1)	QUOTA_LIMITS.
ProcessIoCounters(2)	IO_COUNTERS.
ProcessVmCounters(3)	VIVLCOUNTERS.
ProcessTimes(4)	KERNEL_USER_TIMES.
ProcessDebugPort(7)	ULONG containing the pointer to debug port object.
ProcessLdtInformation(10)	PROCESS LOT INFORMATION.

ProcessDefaultHardErrorMode(12)	ULONG containing the default error mode for the process. This could be any combination of SEM_FAlLCRLTICALERRORS, SEM_NOALIGNMENTFAULTEXCEPT, SEM_NOGPFAULTERRORBOX, and SEM_NOOPENFILEERRORBOX.
ProcessPooledUsageAndLimits(1 4)	POOLED_USAGE_AND_LIMITS.
ProcessWorkingSetWatch(15)	PROCESS_WS_WATCH_INFORMATION.
ProcessWx86Information(19)	ULONG. Always returns 0.
ProcessHandleCount(20)	ULONG containing the number of open handles.
ProcessPriorityBoost{22)	BOOLEAN containing the priority boost control state. TRUE means that dynamic boosting is disabled.

Most of the structures defined previously are documented in the NTDDK.H file from the Windows NT DDK. Other structures can be found in UNDOCNT.H on the accompanying CD-ROM.

Some of this information is available from Equivalent Win32 API calls. However, not all of them are available from Win32 API calls.

EQUIVALENT W1M32 API

GetProcessTimes, GetProcessPriorityBoost, GetProcessAffinityMask, GetProcessShutdownParameters, GetPriorityClass, GetProcessWorkingSetSize, GetProcessVersion

NtQueryInformati onThread

```
NTSTATUS
     NtQueryInformationThread(
          IN HANDLE hThread,
          IN THREADINFOCLASS ThreadInfoClass,
          OUT PVOID ThreadInfoBuffer,
          IN ULONG ThreadInfoBufferLength,
          OPTIONAL OUT PULONG BytesCopied,
);
```

NtQuerylnformationThread returns information about the specified thread object.

PARAMETERS

hThread	Handle to the thread object.

ThreadlnfoClass	Type of information requested.
ThreadlnfoBuffer	Pointer to the buffer that receives the information about the thread object.
ThreadlnfoBufferLength	Size of the buffer (in bytes) pointed to by ThreadlnfoBuffer.
BytesCopied	Pointer to the variable that receives the number of bytes copied into ThreadlnfoBuffer.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

Different information is returned based upon the ThreadlnfoClass parameter. Here is the layout of ThreadlnfoBuffer based on the ThreadlnfoClass:

ThreadBasicInformation(O)	THREAD_BASIC_INFORMATION.
ThreadTimes(l)	KERNEL_USER_TIMES.
ThreadDescriptorTableEntry (6)	DESCRIPTOR_TABLE_ENTRY.
ThreadQuerySetWin32StartAddress (9)	Pointer to ULONG containing the start address of the thread.
ThreadPerformanceCount (11)	Pointer to an array of two ULONGs.
ThreadAmILastThread(12)	Pointer to ULONG that receives whether the calling thread is the last thread of the process (hThread parameter is ignored in this case).
ThreadPriorityBoost (14)	Pointer to ULONG that receives whether dynamic priority boosting is enabled or disabled for the thread.

Some of the structures defined previously are documented in the NTDDK.H file from the Windows NT DDK. Other structures can be found in UNDOCNT.H on the accompanying CD-ROM.

Here are the definitions for the structures that are not documented in NTDDK.H:

```
typedef struct _THREAD_BASIC_INFORMATION (
NTSTATUS ExitStatus;
PVOID  TebBaseAddress;
ULONG  UniqueProcessId;
ULONG UniqueThreadId;
```

```
KAFFINITY AffinityMask;
KPRIORITY BasePnonty;
ULONG DiffProcessPriority;
} THREAD_BASIC_INFORMATION, *PTHREAD_BASIC_INFORMATION;
```

Some of this information is available from Equivalent Win32 API calls. However, not all of them [[[AU: "not all of them" OK?]]]are available from Win32 API calls.

EQUIVALENT WIN32 API

GetThreadPriorityBoost, GetThreadTimes, GetThreadPriority

```
NtQueueApcThread
NTSTATUS
     NtQueueApcThread(
          IN HANDLE hThread,
          IN PKNORMAL_ROUTINE ApcRoutine,
          IN  PVOID  NormalContext.
          IN  PVOID  SystemArgumentl,
          IN  PVOID  SystemArgument2,
);
```

NtQueueApcThread queues an entry to the thread's APC Queue.

PARAMETERS

hThread	Handle to the thread object. - ,
ApcRoutine	The function that gets called when the APC is scheduled for execution.
NormalContext	Points to the context associated with the APC.
SystemArgument1	Points to the first argument to be passed to the APC function.
SystemArgument2	Points to the second argument to be passed to the APC function.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

The Win32 API call corresponding to this system service is prototyped as

```
WINBASEAPI
DWORD
WINAPI
QueueUserAPC(
    PAPCFUNC  pfnAPC,
    HANDLE hThread,
    DWORD dwData
    );
```

The QueueUserAPC function passes pfnAPC for the NormalContext parameter and passes dwData for the SystemArgumentl parameter. It also passes the address of the internal KERNEL32 routine called BaseDipatchAPC as the ApcRoutine parameter. TheBaseDispatchRoutine extracts the pfnAPC and dwData parameters from the parameters passed to it and calls pfnAPC.

EQUIVALENT WIN32 API

QueueUserAPC

```
NtResumeThread
NTSTATUS
     NtResumeThread(
          IN  HANDLE  hThread,
          OUT  PULONG  pSuspendCount
);
```

NtResumeThread decrements the suspend count for the thread and resumes the thread if the suspend count reaches 0.

PARAMETERS

hThread	Handle to the thread object.
pSuspendCount	Pointer to the variable that receives the suspend count of the thread at the time this system service is invoked.

RETURN VALUE

Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS

The Equivalent Win32 API call does not return the suspend count of the thread, whereas the system service does return this information.

EQUIVALENT WIN32 API

ResumeThread

```
NtSetLowWaitHighThread
NTSTATUS
     NtSetLowWaitHighThread(
);
```

NtSetLowWaitHighThread sets the low event of the event pair associated with the calling thread and waits on the high event of the event pair to be signaled.

PARAMETERS
None.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
Chapter 8 provides more information.

EQUIVALENT WIN 32 API
None.

```
NtSetHighWaitLowThread
NTSTATUS
     NtSetHighWaitLowThread(
);
```

NtSetHighWaitLowThread sets the high event of the event pair associated with the calling thread and waits on the low event of the event pair to be signaled.

PARAMETERS
None.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
Chapter 8 provides more information.

EQUIVALENT WIN32 API
None.

```
NtSuspendThread
NTSTATUS
     NtSuspendThread(
          IN HANDLE hThread,
          OUT PULONG pSuspendCount
);
```

NtSuspendThread suspends the thread and increments the suspend count for the thread.

PARAMETERS

hThread	Handle to the thread object.
pSuspendCount	Pointer to the variable that receives the suspend count of the thread at the time this system service is invoked.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
The Equivalent Win32 API call does not return the suspend count of the thread, whereas the system service does return this information.

EQUIVALENT WIN32 API
SuspendThread

```
NtTerminateProcess
NTSTATUS
     NtTerminateProcess(
          IN HANDLE hProcess,
          IN ULONG ExitCode
);
```

NtTerminateProcess terminates the specified process.

PARAMETERS

hProcess	Handle to the process object.
ExitCode	Exit code for the process.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
ExitProcess, TerminateProcess

```
NtTerminateThread
NTSTATUS
     NtTerminateThread(
          IN HANDLE hThread,
          IN ULONG ExitCode
);
```

NtTerminateThread terminates the specified thread.

PARAMETERS

hThread Handle to the thread object.

ExitCode Exit code for the thread.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
ExitThread, TerminateThread

```
NtYieldExecution
NTSTATUS
     NtYieldExecution(
);
```

NtYieldExecution relinquishes the processor from the calling thread.

PARAMETERS
None.

RETURN VALUE
Returns STATUS_SUCCESS on success and an appropriate error code on failure.

COMMENTS
None.

EQUIVALENT WIN32 API
SwitchToThread.

Appendix B

What's on the CD-ROM

The CD-ROM that accompanies this book contains the source code and binaries for all the sample applications that we've discussed in this book. Each sample is kept in a separate directory.

To try out each sample, open the accompanying README.TXT file. Each contains step-by-step instructions for installing and trying out the sample.

In each README.TXT file, we have also identified which system (or systems) we used to test the sample - Windows NT 3.51, Windows NT 4, and/or Windows 2000 beta 1 and beta 2.

The samples are compiled using MSVC 4.2 compiler and Windows NT 4.0 DDK.