pExecD

COLLABORATORS			
	TITLE : pExecD		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 29, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

pExecD

1.1 pExecD.doc

pexec.library

```
ExecAllgemeines()                 pOS_AbortIO()
pOS_AddClass()                    pOS_AddDevice()
pOS_AddLibrary()                  pOS_AddLinkClass()
pOS_AddMemHandler()               pOS_AddMemList()
pOS_AddPort()                     pOS_AddResource()
pOS_AddSemaphore()                pOS_AddSemaphoreQR()
pOS_AddTask()                     pOS_AllocAbs()
pOS_Allocate()                    pOS_AllocEntry()
pOS_AllocMem()                    pOS_AllocPoolMem()
pOS_AllocPoolVec()                pOS_AllocSignal()
pOS_AllocTrap()                   pOS_AllocVec()
pOS_AttemptProcureSemaphore()     pOS_AttemptSemaphore()
pOS_AttemptSemaphoreShared()      pOS_AttemptTimeSemaphore()
pOS_AttemptTimeSemaphoreShared()  pOS_AvailMem()
pOS_BeginIO()                     pOS_CacheClearE()
pOS_CacheClearU()                 pOS_CacheControl()
pOS_CheckIO()                     pOS_CloseClass()
pOS_CloseDevice()                 pOS_CloseLibrary()
pOS_CloseResource()               pOS_ColdReboot()
pOS_ConstructMsgPort()            pOS_ConstructSigPort()
pOS_CopyMem()                     pOS_CreateClass()
pOS_CreateClassGrp()              pOS_CreateIORequest()
pOS_CreateObject()                pOS_CreatePort()
pOS_CreateTask()                  pOS_Deallocate()
pOS_DebugClassAbs()               pOS_DebugClassI()
pOS_DeleteClass()                 pOS_DeleteClassGrp()
pOS_DeleteIORequest()             pOS_DeleteObject()
pOS_DeletePort()                  pOS_DeleteTask()
pOS_DestructMsgPort()             pOS_DisplayAlert()
pOS_DoAbsMethodA()                pOS_DoIMethodA()
pOS_DoIO()                        pOS_DoMMethodA()
pOS_DoVirMethodA()                pOS_ExecCheckA()
pOS_FindPort()                    pOS_FindResident()
pOS_FindSemaphore()               pOS_FindTask()
pOS_ForbidIRQ()                   pOS_ForbidSchedul()
pOS_FreeEntry()                   pOS_FreeMem()
```

```
pOS_FreePoolAll()                pOS_FreePoolMem()
pOS_FreePoolVec()                pOS_FreeSignal()
pOS_FreeTrap()                   pOS_FreeVec()
pOS_GetIMemberAdr()              pOS_GetMemberAdr()
pOS_GetMsg()                     pOS_GetNClass()
pOS_GetObjectRootAdr()           pOS_GetUpObjectAdr()
pOS_InitMemPool()                pOS_InitResident()
pOS_InitSemaphore()              pOS_IsObtainSemaphore()
pOS_MakeFunctions()              pOS_MakeLibrary()
pOS_MoveDownClassPtr()           pOS_MoveUpClassPtr()
pOS_ObtainSemaphore()            pOS_ObtainSemaphoreList()
pOS_ObtainSemaphoreShared()      pOS_OpenClass()
pOS_OpenDevice()                 pOS_OpenLibrary()
pOS_OpenResource()               pOS_PermitIRQ()
pOS_PermitSchedul()              pOS_ProcureSemaphore()
pOS_ProcureSemaphoreShared()     pOS_PutMsg()
pOS_RawDoFmt()                   pOS_ReadAsciiFmt()
pOS_ReleaseSemaphore()           pOS_ReleaseSemaphoreList()
pOS_RemDevice()                  pOS_RemLibrary()
pOS_RemMemHandler()              pOS_RemPort()
pOS_RemResource()                pOS_RemSemaphore()
pOS_RemSemaphoreQR()             pOS_RemTask()
pOS_ReplyMsg()                   pOS_SendIO()
pOS_SendSignal()                 pOS_SetExcept()
pOS_SetLibFunction()             pOS_SetSignal()
pOS_SetTaskPriority()            pOS_SetTaskXPri()
pOS_StackSwap()                  pOS_SubClass()
pOS_SumLibrary()                 pOS_VacateSemaphore()
pOS_WaitIO()                     pOS_WaitPort()
pOS_WaitSignal()                 pOS_WaitTimeSignal()
pOS_WriteAsciiFmt()              pOS_WriteDebug()
```

1.2 pexec.library/ExecAllgemeines()

```
STRUKTUREN

        struct pOS_AsciiFmtData
        struct pOS_Callback
        struct pOS_Class
        struct pOS_ClassGrp
        struct pOS_Device
        struct pOS_ExecBase;
        struct pOS_ExList
        struct pOS_ExNode
        struct pOS_Interrupt
        struct pOS_IORequest
        struct pOS_IOStdReq
        struct pOS_LibDescribe;
        struct pOS_Library;
        struct pOS_LibraryFunction
        struct pOS_List
        struct pOS_Member
        struct pOS_MemChunk
        struct pOS_MemEntry
```

```
        struct pOS_MemHeader
        struct pOS_MemList
        struct pOS_MemPool
        struct pOS_Message
        struct pOS_Method;
        struct pOS_MsgPort
        struct pOS_NClass;
        struct pOS_Node
        struct pOS_Object
        struct pOS_RamLibMessage
        struct pOS_RawDoFmtData
        struct pOS_Resident
        struct pOS_ResidentLibInit
        struct pOS_Resource
        struct pOS_SemaphoreRequest
        struct pOS_Semaphore
        struct pOS_StackSwapData
        struct pOS_StdDeviceFunction
        struct pOS_StdLibraryFunction
        struct pOS_TagItem;
        struct pOS_Task
        struct pOS_Unit

ENUMERATION

        enum pOS_Alerts
        enum pOS_ExNodeType
        enum pOS_IOReqCommands
        enum pOS_IOReqErrors
        enum pOS_IOReqFlags
        enum pOS_LibraryFlags
        enum pOS_LibraryFuncID
        enum pOS_MemoryFlag
        enum pOS_MsgPortFlag
        enum pOS_RamLibMessageCom
        enum pOS_RawDoFmtFlags
        enum pOS_ResidentFlags
        enum pOS_SemaphoreFlags
        enum pOS_TaskSignal
        enum pOS_TaskStates
        enum pOS_UnitFlags

DEFINES

        RTC_MATCHWORD
        pOS_CLASSERR

INCLUDES

        pExec/CallBack.h
        pExec/Class.h
        pExec/Device.h
        pExec/Diagnos.h
        pExec/ExecBase.h
        pExec/Interupt.h
        pExec/Library.h
        pExec/List.cpp
```

```
      pExec/List.h
      pExec/Macros.h
      pExec/Memory.h
      pExec/MsgPort.h
      pExec/Node.h
      pExec/RamLib.h
      pExec/RawDoFmt.h
      pExec/Resident.h
      pExec/Resource.h
      pExec/Sema.h
      pExec/Task.h
      pExec/Types.h
      proto/pExec.h
```

1.3 pexec.library/pOS_ColdReboot()

```
PROTOTYP
      VOID pOS_ColdReboot
        (
          pOS_ExecBase *execbase
        );

FUNKTION
      Ausführen eines Kaltstarts (Rechnerneustart)

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library

AMIGA FUNKTION
          VOID ColdReboot(VOID);
```

1.4 pexec.library/pOS_WriteDebug()

```
      ### wird noch erweitert ###

PROTOTYP
      VOID pOS_WriteDebug
        (
          pOS_ExecBase *execbase,
          const VOID *data,
          size_t size,
          ULONG type
        );

FUNKTION
      Ausgeben von Debug-Informationen

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
```

```
      data (_R_A0)
      size (_R_D0)
      type (_R_D1)

AMIGA FUNKTION
      VOID kprintf(const char *,...);
```

1.5 pexec.library/pOS_AddTask()

```
PROTOTYP
      APTR res = pOS_AddTask
        (
          pOS_ExecBase *execbase,
          pOS_Task *task,
          APTR initialpc,
          APTR finalpc
        );

FUNKTION
      Hinzufügen eines Tasks in die Systemliste.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      task (_R_A0)
        Zeiger auf die installierte Task-Struktur. Die nicht
        vorbelegten Felder müssen auf 0 gesetzt werden.
      initialpc (_R_A1)
        Zeiger auf die Startadresse des Tasks
      finalpc (_R_A2)
        Zeiger auf die Rücksprungadresse des Tasks oder NULL

ERGEBNIS
      res (_R_D0)
        Übergebene Adresse der Task-Struktur,
        oder NULL im Fehlerfall.

HINWEIS
      Mittels pOS_CreateTask() kann normalerweise auf diese low-level
      Funktion verzichtet werden.

      Nach dem hinzufügen des Tasks wird ein Scheduling ausgelöst.
      Der Task mit der höchsten Priorität läuft danach. Je nach
      Priorität kann das der eigene oder ein anderer Task sein!

      Da es im Task-Umfeld keine Process-Struktur gibt, sind auch
      die meisten pDos-Funktionen nicht benutzbar.
      Es funktioniert pOS_CreateProc() und CreateProcessA() zum Erzeugen
      des Processes, sowie die Funktionen für ReadArgs.
      Der erzeugte Process bietet nur eine minimale Dos-Umgebung.
      CurrentDir zeigt auf SYS:, In/Out/Error-Kanäle zeigen auf NIL:

SIEHE AUCH
      pOS_RemTask(), pOS_FindTask(), pOS_CreateTask()
```

```
AMIGA FUNKTION
      struct Task *AddTask(struct Task *task, APTR initialpc, APTR finalpc);
```

1.6 pexec.library/pOS_RemTask()

```
PROTOTYP
      VOID pOS_RemTask
        (
          pOS_ExecBase *execbase,
          pOS_Task *task
        );

FUNKTION
      Entfernen eines Tasks aus der Systemliste und freigeben
      der vom System angelegten Resourcen

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      task (_R_A0)
        Zeiger auf den zu entfernenden Task,
        NULL wenn der eigene Task ausgehängt werden soll

HINWEIS
      Das Entfernen von fremden Task ist nicht ungefährlich,
      deshalb sollte sich jeder Task selber mittels pOS_RemTask(NULL)
      entfernen.
      Zum Entfernen von fremden Task sollte die Funktion pOS_DeleteTask()
      verwendet werden.
      Nach dieser Funktion läuft höchstens noch die finalpc-Routine
      (wenn bei pOS_AddTask() angegeben).

SIEHE AUCH
      pOS_AddTask(), pOS_FindTask(), pOS_DeleteTask()

AMIGA FUNKTION
      VOID RemTask(struct Task *task);
```

1.7 pexec.library/pOS_FindTask()

```
PROTOTYP
      pOS_Task *res = pOS_FindTask
        (
          pOS_ExecBase *execbase,
          const CHAR *name
        );

FUNKTION
      Suchen eines Tasks über seinen Namen. Dabei wird zwischen
```

```
      Groß- und Kleinschreibung unterschieden.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      name (_R_A0)
        Name des zu suchenden Tasks
        oder NULL für den eigenen Task

ERGEBNIS
      res (_R_D0)
        Zeiger auf den gefunden Task,
        oder NULL wenn kein entsprechender Task gefunden wurde

SIEHE AUCH
      pOS_AddTask(), pOS_RemTask(), pOS_CreateTask()

AMIGA FUNKTION
      struct Task *FindTask(STRPTR name);
```

1.8 pexec.library/pOS_SetTaskPriority()

```
PROTOTYP
      SBYTE res = pOS_SetTaskPriority
        (
          pOS_ExecBase *execbase,
          pOS_Task *task,
          SLONG priority
        );

FUNKTION
      Setzen der Priorität eines Tasks

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      task (_R_A0)
        Adresse des Tasks, dessen Priorität geändert werden soll
      priority (_R_D0)
        neue Priorität im Bereich von -128 (niedrigste Priorität)
        bis +127 (höchste Priorität)

ERGEBNIS
      res (_R_D0)
        bisheriger Prioritätswert

SIEHE AUCH
      pOS_FindTask()

AMIGA FUNKTION
      LONG SetTaskPri(struct Task *task, LONG priority);
```

1.9 pexec.library/pOS_SetSignal()

```
PROTOTYP
      ULONG res = pOS_SetSignal
        (
          pOS_ExecBase *execbase,
          ULONG newsignals,
          ULONG signalmask
        );

FUNKTION
      Auslesen und Verändern der Signal-Bits des aktuellen Tasks.

PARAMETER
          execbase (_R_LB)
            Zeiger auf pExec-Library
          newsignals (_R_D0)
            Bitmaske mit den neuen Signalwerten; vordefiniert sind
              DOSSIGF_CTRL_C : normalerweise Befehl abbrechen
      ~       DOSSIGF_CTRL_D : normalerweise Script abbrechen
              DOSSIGF_CTRL_E : Programmspezifisch
              DOSSIGF_CTRL_F : Programmspezifisch
          signalmask (_R_D1)
            Bitmaske die definiert, welche Bits von der Operaton betroffen
            sein sollen; vordefiniert sind
              DOSSIGF_CTRL_C : normalerweise Befehl abbrechen
      ~       DOSSIGF_CTRL_D : normalerweise Script abbrechen
              DOSSIGF_CTRL_E : Programmspezifisch
              DOSSIGF_CTRL_F : Programmspezifisch

ERGEBNIS
      res (_R_D0)
        Bitmaske mit den bisherigen Signalwerten

BEISPIEL
      Ermitteln der aktuellen Task-Signale:
        pOS_SetSignal(0,0);
      Löschen des CTRL-C-Signals:
        pOS_SetSignal(0,DOSSIGF_CTRL_C);
      Feststellen, ob CTRL-C gedrückt (empfangen) wurde
      danach wird das Signal automatisch gelöscht
        if(pOS_SetSignal(0,DOSSIGF_CTRL_C) & DOSSIGF_CTRL_C)
          printf("CTRL-C gedrückt!\n");

SIEHE AUCH
      pOS_WaitSignal()

AMIGA FUNKTION
          ULONG SetSignal(ULONG newsignals, ULONG signalmask);
```

1.10 pexec.library/pOS_SetExcept()

```
      ### wird noch erweitert ###
```

```
PROTOTYP
      ULONG res = pOS_SetExcept
        (
          pOS_ExecBase *execbase,
          ULONG newsignals,
          ULONG signalmask
        );

FUNKTION
      Signalwerte für das Auslösen einer Exception definieren

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      newsignals (_R_D0)
        Bitmaske mit den neuen Signalen
      signalmask (_R_D1)
        Bitmaske für die Definition der betroffenen Signale

ERGEBNIS
      res (_R_D0)
        Bitmaske der bisherigen Signale

SIEHE AUCH
      pOS_SendSignal(), pOS_SetSignal()

AMIGA FUNKTION
      ULONG SetExcept(ULONG newsignals, ULONG signalmask);
```

1.11 pexec.library/pOS_WaitSignal()

```
PROTOTYP
      ULONG res = pOS_WaitSignal
        (
          pOS_ExecBase *execbase,
          ULONG signalmask
        );

FUNKTION
      Warten auf Signale

PARAMETER
          execbase (_R_LB)
            Zeiger auf pExec-Library
          signalmask (_R_D0)
            Maske der Signale, auf die gewartet werden soll oder 0
            Trifft keines dieser Signale ein, so wartet die Funktion
            endlos! Vordefiniert sind
              DOSSIGF_CTRL_C : normalerweise Befehl abbrechen
      ~       DOSSIGF_CTRL_D : normalerweise Script abbrechen
              DOSSIGF_CTRL_E : Programmspezifisch
              DOSSIGF_CTRL_F : Programmspezifisch
            TIP: Verwechseln Sie keine Signalsbits (z.B. DOSSIGB_CTRL_C = 0xC)
```

```
                mit der Signalmaske (z.B. DOSSIGF_CTRL_C = 0x1000)

ERGEBNIS
      res (_R_D0)
        Aufgetretene Signale. Es können mehrere der in signalmask
        gesetzten Signale eingetroffen sein!

HINWEIS
      Geben Sie bei der signalmask 0 an, wartet der Task ewig.
      Damit ist aber ein definierter Stand geschaffen, damit der
      Task mit pOS_RemTask() gelöscht werden kann.

BEISPIEL
      Warten bis CTRL-C oder CTRL-D gedrückt wird
        ULONG sigs=pOS_WaitSignal(DOSSIGF_CTRL_C | DOSSIGF_CTRL_D)
        if(sigs & DOSSIGF_CTRL_C)
          printf("CTRL-C gedrückt!\n");
        if(sigs & DOSSIGF_CTRL_D)
          printf("CTRL-D gedrückt!\n");

SIEHE AUCH
      pOS_AllocSignal(), pOS_SendSignal(), pOS_WaitPort()

AMIGA FUNKTION
      ULONG Wait(ULONG signalmask);
```

1.12 pexec.library/pOS_SendSignal()

```
PROTOTYP
      VOID pOS_SendSignal
        (
          pOS_ExecBase *execbase,
          pOS_Task *task,
          ULONG signalmask
        );

FUNKTION
      Senden von Signalen an einen Task

PARAMETER
          execbase (_R_LB)
            Zeiger auf pExec-Library
          task (_R_A0)
            Task an den die Signale geschickt werden sollen
          signalmask (_R_D0)
            Bitmaske mit den Signalen; vordefiniert sind
              DOSSIGF_CTRL_C : normalerweise Befehl abbrechen
      ~       DOSSIGF_CTRL_D : normalerweise Script abbrechen
              DOSSIGF_CTRL_E : Programmspezifisch
              DOSSIGF_CTRL_F : Programmspezifisch

SIEHE AUCH
      pOS_AllocSignal(), pOS_PutMsg()
```

```
AMIGA FUNKTION
      VOID Signal(struct Task *task, ULONG signalmask);
```

1.13 pexec.library/pOS_AllocSignal()

```
PROTOTYP
      __ARID__ ULONG res = pOS_AllocSignal
        (
          pOS_ExecBase *execbase,
          ULONG signalnummer
        );

FUNKTION
      Reservieren eines Signalbits

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      signalnummer (_R_D0)
        gewünschte Signalnummer im Bereich von 0 bis 31
        oder SIGB_AllocAny für das nächste freie Signal-Bit
        (vom Betriebsystem werden z.Z. die Bits 4, 8, 12, 13,
         14, 15 benutzt und können daher nicht reserviert werden)

ERGEBNIS
      res (_R_D0)
        reserviertes Signal-Bit, das mittels pOS_FreeSignal()
        wieder freigegeben werden muß,
        oder SIGB_NoSignal wenn kein Signal mehr frei ist

SIEHE AUCH
      pOS_FreeSignal(), pOS_WaitSignal()

AMIGA FUNKTION
      ULONG AllocSignal(ULONG signalnummer);
```

1.14 pexec.library/pOS_FreeSignal()

```
PROTOTYP
      VOID pOS_FreeSignal
        (
          pOS_ExecBase *execbase,
          __ARID__ ULONG signalnummer
        );

FUNKTION
      Freigeben eines Signal-Bits

PARAMETER
      execbase (_R_LB)
```

```
        Zeiger auf pExec-Library
      signalnummer (_R_D0)
        Nummer des freizugebenden Signal-Bits, das mittels
        pOS_AllocSignal() reserviert wurde
        ACHTUNG: Nach der Funktion darf das Signal-Bit nicht
                 mehr verwendet werden!

SIEHE AUCH
      pOS_AllocSignal()

AMIGA FUNKTION
      VOID FreeSignal(ULONG signalnummer);
```

1.15 pexec.library/pOS_AllocTrap()

```
PROTOTYP
      __ARID__ ULONG res = pOS_AllocTrap
        (
          pOS_ExecBase *execbase,
          ULONG trapnummer
        );

FUNKTION
      Reservieren eines Prozessor-Trap-Vektors

PARAMETER
          execbase (_R_LB)
            Zeiger auf pExec-Library
          trapnummer (_R_D0)
            Nummer des gewünschten Traps im Bereich von 0 bis 15
      >>>       TRAPB_AllocAny für die nächste freie Trap-Nummer

ERGEBNIS
          res (_R_D0)
            Nummer des tatsächlich reservierten Trap-Vektors.
            Dieser muß mittels pOS_FreeTrap() wieder freigegeben werden.
      >>>       TRAPB_NoTrap falls kein Trap mehr zur Verfügung steht.

SIEHE AUCH
      pOS_FreeTrap()

AMIGA FUNKTION
      LONG AllocTrap(LONG trapnummer);
```

1.16 pexec.library/pOS_FreeTrap()

```
PROTOTYP
      VOID pOS_FreeTrap
        (
          pOS_ExecBase *execbase,
```

```
          __ARID__ ULONG trapnummer
        );

FUNKTION
      Freigeben eines Prozessor-Trap-Vektors

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      trapnummer (_R_D0)
        Nummer des Traps, der freigegeben werden soll und mittels
        pOS_AllocTrap() reserviert wurde.
        ACHTUNG: Nach der Funktion darf die Trap-Nummer nicht
                 mehr benutzt werden.

SIEHE AUCH
      pOS_AllocTrap()

AMIGA FUNKTION
      VOID FreeTrap(LONG trapnummer);
```

1.17 pexec.library/pOS_ForbidSchedul()

```
PROTOTYP
      VOID pOS_ForbidSchedul
        (
          pOS_ExecBase *execbase
        );

FUNKTION
      Unterbinden des Task-Switchings

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library

HINWEIS
      Diese Funktion sollte nur verwendet werden, wenn es unumgänglich
      ist (z.B. auslesen von globalen Systemlisten). Die Sperrung muß
      mittels pOS_PermitSchedul() aufgehoben werden.
      Interrupt müssen, bei Bedarf, mittels pOS_ForbidIRQ() unterbunden
      werden.

SIEHE AUCH
      pOS_PermitSchedul(), pOS_ForbidIRQ()

AMIGA FUNKTION
      VOID Forbid();
```

1.18 pexec.library/pOS_PermitSchedul()

```
PROTOTYP
      VOID pOS_PermitSchedul
        (
          pOS_ExecBase *execbase
        );

FUNKTION
      Erlauben des Task-Switchings, das mittels pOS_ForbidSchedul()
      gesperrt wurde.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library

SIEHE AUCH
      pOS_ForbidSchedul(), pOS_PermitIRQ()

AMIGA FUNKTION
      VOID Permit();
```

1.19 pexec.library/pOS_CreateTask()

```
PROTOTYP
      pOS_Task *res = pOS_CreateTask
        (
          pOS_ExecBase *execbase,
          const CHAR *name,
          SLONG priority,
          APTR startfunc,
          size_t stacksize,
          size_t tasksize,
          ULONG userdata0,
          pOS_TaskControl *taskcontrol
        );

FUNKTION
      Erzeugen eines neuen Tasks

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      name (_R_A0)
        Name des neuen Tasks; wird in task->tc_Node.ln_Name eingetragen
      priority (_R_D0)
        Priorität des neuen Tasks im Bereich von -128 bis +127;
        wird in task->tc_Node.ln_Pri eingetragen
      startfunc (_R_A1)
        Adresse der Startfunktion des neuen Tasks die angesprungen
        wird, sobald der neue Task (prioritätsmäßig) das erste mal läuft
      stacksize (_R_D1)
        Größe des Stackspeichers in Byte für den neuen Task
      tasksize (_R_D2)
```

```
        Größe der Task-Struktur, gewöhnlich sizeof(pOS_Task),
        kann jedoch durch eine größere Angabe um private Elemente
        erweitert werden
      unserdata0 (_R_D3)
        Beliebiger Beutzerwert, der in der Task-Struktur unter
        task->tc_UserData[0] abgelegt wird
      taskcontrol (_R_A2)
        Zeiger auf einen Task-Control-Block oder NULL (default)

ERGEBNIS
      res (_R_D0)
        Übergebene Adresse der Task-Struktur,
        oder NULL im Fehlerfall.

HINWEIS
      Es gelten die selben Hinweise wie bei pOS_AddTask()

SIEHE AUCH
      pOS_AddTask(), pOS_RemTask(), pOS_FindTask()

AMIGA FUNKTION
      struct Task *CreateTask(STRPTR name, LONG priority, APTR startfunc, ULONG  ↩
         stacksize);
```

1.20 pexec.library/pOS_CacheClearE()

```
PROTOTYP
      VOID pOS_CacheClearE
        (
          pOS_ExecBase *execbase,
          APTR adress,
          size_t size,
          ULONG flags
        );

FUNKTION
      Löschen des Cache-Speichers des Prozessors.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      adress (_R_A0)
        Startadresse des Speichers
      size (_R_D0)
        Größe des zu löschenden Speicherbereichs
        ~0 für den gesamten Cache
      flags (_R_D1)
        ???: Befehlscache löschen
        ???: Datencache löschen

HINWEIS
      Diese Funktion muß benutzt werden, wenn im Speicher eine
      Änderung ohne Prozessorbeteiligung stattgefunden hat
      (z.B. DMA-Controller). Dabei kann der Speicherbereich
```

```
      eingegrenzt werden.

SIEHE AUCH
      pOS_CacheControl(), pOS_CacheClearU()

AMIGA FUNKTION
      VOID CacheClearE(APTR adress, ULONG size, ULONG flags);
```

1.21 pexec.library/pOS_CacheClearU()

```
PROTOTYP
      VOID pOS_CacheClearU
        (
          pOS_ExecBase *execbase
        );

FUNKTION
      Löschen des gesamten Cache-Speichers des Prozessors.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library

SIEHE AUCH
      pOS_CacheControl(), pOS_CacheClearE()

AMIGA FUNKTION
      VOID CacheClearU()
```

1.22 pexec.library/pOS_CacheControl()

```
PROTOTYP
      ULONG res = pOS_CacheControl
        (
          pOS_ExecBase *execbase,
          ULONG newbits,
          ULONG bitmask
        );

FUNKTION
      Kontrollieren des Cache-Speichers des Prozessors.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      newbits (_R_D0)
        Neue Werte der Cache-Bits; definiert sind
      bitmask (_R_D1)
        Bitmaske, die definiert, welche Cache-Bits von der Operation
        betroffen sein sollen; definiert sind
```

```
ERGEBNIS
      res (_R_D0)
        Bisheriger Wert der Cache-Bits.

SIEHE AUCH
      pOS_CacheClearE(), pOS_CacheClearU()

AMIGA FUNKTION
      ULONG CacheControl(ULONG newbits, ULONG bitmask);
```

1.23 pexec.library/pOS_StackSwap()

```
PROTOTYP
      VOID pOS_StackSwap
        (
          pOS_ExecBase *execbase,
          pOS_StackSwapData *stackswapdata
        );

FUNKTION
      Setzen eines neuen Stacks für den aufrufenden Task

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      stackswapdata (_R_A0)
        Adresse und Größe des neuen Stacks

HINWEIS
      Nach der Funktion steht in stackswapdata die alten Stackwerte.
      Durch einen erneuten Aufruf von pOS_StackSwap() kann somit
      wieder der Orginalzustand hergestellt werden.

AMIGA FUNKTION
      StackSwap(struct StackSwapData *stackswapdata);
```

1.24 pexec.library/pOS_ForbidIRQ()

```
PROTOTYP
      VOID pOS_ForbidIRQ
        (
          pOS_ExecBase *execbase
        );

FUNKTION
      Unterbinden von Interrupts.

PARAMETER
      execbase (_R_LB)
```

```
        Zeiger auf pExec-Library

HINWEIS
      Diese Funktion sollte nur verwendet werden, wenn das Unterbinden
      von Interrupts unumgänglich ist. Die Sperrung muß mittels
      pOS_ForbidIRQ() aufgehoben werden.
      Durch die Interrupt-Sperrung findet auch kein Task-Switching stand.
      Dadurch ist ein Aufruf von pOS_ForbidSchedule() nicht notwendig.

SIEHE AUCH
      pOS_PermitIRQ(), pOS_ForbidSchedul()

AMIGA FUNKTION
      VOID Disable();
```

1.25 pexec.library/pOS_PermitIRQ()

```
PROTOTYP
      VOID pOS_PermitIRQ
        (
          pOS_ExecBase *execbase
        );

FUNKTION
      Erlauben von Interrupts, die mittels pOS_ForbidIRQ()
      gesperrt wurden.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library

SIEHE AUCH
      pOS_ForbidIRQ(), pOS_PermitSchedul()

AMIGA FUNKTION
          VOID Enable();


      pUtility.library/pOS_WaitTimeSignal (V_1)

PROTOTYP
      ULONG res = pOS_WaitTimeSignal
        (
          pOS_ExecBase *execbase,
          ULONG signalmask,
          ULONG micros
        );

FUNKTION
      Warten auf das Eintreffen von Signalen oder das Ablaufen des
      Timers.

PARAMETER
      execbase (_R_LB)
```

```
        Zeiger auf pExec-Library
      signalmask (_R_D0)
        Maske der Signale, auf die gewartet werden soll oder 0
      micros (_R_D1)
        Maximale Wartezeit in Micro-Sekunden (1000000 Mikros = 1 Sekunde).
        Trift während dieser Zeit kein definiertes Signal ein,
        wird die Funktion abgebrochen.

ERGEBNIS
      res (_R_D0)
        Aufgetretene Signale oder 0, wenn die Zeit abgelaufen ist.
        Es können mehrere der in signalmask gesetzten Signale ein-
        getroffen sein!

HINWEIS
      Wenn Sie für signalmask 0 angeben, können Sie mit dieser Funktion
      nur die definierte Zeit micros warten. Das ist ähnlich der
      AMIGA FUNKTION TimeDelay().

SIEHE AUCH
      pOS_WaitSignal()

AMIGA FUNKTION
```

1.26 pexec.library/pOS_DeleteTask()

```
PROTOTYP
      VOID pOS_DeleteTask
        (
          pOS_ExecBase *execbase,
          __ARID__ pOS_Task *task,
          pOS_Task *signaltask,
          ULONG signal
        );

FUNKTION
      Entfernen eines Tasks aus der Task-Liste des Systems.
      Nach dem Entfernen kann einem anderen Task ein Signal
      als Bestätigung geschickt werden.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      task (_R_A0)
        Zeiger auf den zu entfernenden Task
      signaltask (_R_A1)
        Zeiger auf den Task, der nach dem Entfernen das definierte
        Signal bekommt oder NULL
      signal (_R_D0)
        Signal, das an signaltask, nach dem Entfernen, geschickt werden soll.

HINWEIS
      Sie können signaltask und signal auf 0 setzen, dann ist
      pOS_DeleteTask(execbase,task,NULL,0) gleichbedeutend mit
```

```
      pOS_RemTask(execbase,task).

SIEHE AUCH
      pOS_RemTask(), pOS_CreateTask()

AMIGA FUNKTION
          ähnlich VOID DeleteTask(struct Task *task);
```

1.27 pexec.library/pOS_SetTaskXPri()

```
      ### wird noch erweitert ###
PROTOTYP
      SBYTE res = pOS_SetTaskXPri
        (
          pOS_ExecBase *execbase,
          pOS_Task *task,
          SLONG runpri,
          SLONG syspri,
          ULONG mode
        );

FUNKTION
      Ändern der Priorität für einen Task

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      task (_R_A0)
        Adresse des Tasks, dessen Priorität geändert werden soll
      runpri (_R_D0)
      syspri (_R_D1)
      mode (_R_D2) (enum pOS_SetTaskXPriMode)
        TKXPDMD_None
        TKXPDMD_IgnoreSPri
        TKXPDMD_UseSPri    : SysPri setzen
        TKXPDMD_NoSem      : Speziallfall nur intern

ERGEBNIS
      res (_R_D0)
        bisheriger Prioritätswert

SIEHE AUCH
      pOS_SetTaskPriority()

AMIGA FUNKTION
```

1.28 pexec.library/pOS_AddPort()

```
PROTOTYP
      VOID pOS_AddPort
        (
```

```
          pOS_ExecBase *execbase,
          pOS_MsgPort *msgport
        );

FUNKTION
      Einfügen eines Message-Ports in die Systemliste

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      msgport (_R_A0)
        Zeiger auf die vollständig installierte Message-Port-Struktur
        Entsprechend dem Eintrag msgport->mp_Node.ln_Pri wird der
        Port eingereiht.

HINWEIS
      Der Message-Port kann mittels pOS_RemPort() wieder aus der
      Systemliste entfernt werden.
      Die mp_MsgList wird beim Einfügen automatisch mittels pOS_ListInit()
      initialisiert.

SIEHE AUCH
      pOS_CreatePort(), pOS_RemPort(), pOS_FindPort()

AMIGA FUNKTION
      VOID AddPort(struct MsgPort *msgport);
```

1.29 pexec.library/pOS_RemPort()

```
PROTOTYP
      VOID pOS_RemPort
        (
          pOS_ExecBase *execbase,
          pOS_MsgPort *msgport
        );

FUNKTION
      Entfernen eines Message-Ports aus der Systemliste

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      msgport (_R_A0)
        Zeiger auf den zu entfernenden Message-Port, der mittels
        pOS_AddPort() oder pOS_CreatePort() in die Systemliste
        eingehängt wurde.

HINWEIS
      Nach dem Aushängen können sich noch Nachrichten in dem
      Port befinden. Diese müssen mittels pOS_GetMsg() ausgelesen
      und mittels pOS_ReplyMsg() zurückgeschickt werden.

SIEHE AUCH
      pOS_AddPort(), pOS_FindPort(), pOS_DeletePort()
```

```
AMIGA FUNKTION
      VOID RemPort(struct MsgPort *msgport);
```

1.30 pexec.library/pOS_PutMsg()

```
PROTOTYP
      VOID pOS_PutMsg
        (
          pOS_ExecBase *execbase,
          pOS_MsgPort *msgport,
          __ARID__ pOS_Message *message
        );

FUNKTION
      Senden einer Message an einen Message-Port

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      msgport (_R_A0)
        Zeiger auf den Message-Port, an den die Nachricht geschickt
        werden soll
      message (_R_A1)
        Zeiger auf die zu versendende Nachricht

HINWEIS
      In der Message wird der Eintrag message->mn_Node.ln_Typ auf
      NTYP_MESSAGE gesetzt. Entsprechend dem Eintrag msgport->mp_Flags
      werden zusätzliche Aktionen veranlaßt:
        MSGPORTF_Signal  : es wird ein Signal an den Empfänger geschickt
        MSGPORTF_SoftInt : es wird ein Software-Interrupt ausgelöst
        MSGPORTF_Ignore  : keine zusätzliche Aktion
      Wird zusätzlich das Flag MSGPORTF_TPEnqueue mit angegeben, wird
      die Nachricht entsprechend ihrer Priorität (TaskPri vom Nachrichtensender)
      in die Message-Port-Liste des Empfängers eingereit.

SIEHE AUCH
      pOS_SendSignal(), pOS_AddPort(), pOS_GetMsg(), pOS_ReplyMsg()

AMIGA FUNKTION
      VOID PutMsg(struct MsgPort *msgport, struct Message *message);
```

1.31 pexec.library/pOS_GetMsg()

```
PROTOTYP
      __ARID__ pOS_Message *res = pOS_GetMsg
        (
          pOS_ExecBase *execbase,
          pOS_MsgPort *msgport
```

```
        );

FUNKTION
      Abholen der nächsten Message von einem Message-Port

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      msgport (_R_A0)
        Zeiger auf den Message-Port, dessen nächste Nachricht
        ermittelt werden soll

ERGEBNIS
      res (_R_D0)
       Zeiger auf die erste Message im Port,
       oder NULL wenn keine Messages vorliegen

HINWEIS
      Auslesen aller Messages aus dem Port:
        struct pOS_MsgPort *msgport;
        struct pOS_Message *msg;
        while(msg=pOS_GetMsg(pOS_ExecBase,msgport))
        {
          /* Bearbeiten der Nachricht */
          pOS_ReplyMsg(pOS_ExecBase,msg);
          /* Auslesen von msg nicht mehr erlaubt !!! */
        }

SIEHE AUCH
      pOS_WaitPort(), pOS_ReplyMsg(), pOS_PutMsg()

AMIGA FUNKTION
      struct Message *GetMsg(struct MsgPort *msgport);
```

1.32 pexec.library/pOS_ReplyMsg()

```
PROTOTYP
      VOID pOS_ReplyMsg
        (
          pOS_ExecBase *execbase,
          __ARID__ pOS_Message *message
        );

FUNKTION
      Beantworten einer Message durch zurücksenden an den Absender

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      message (_R_A0)
        Zeiger auf die zurückzusendende Nachricht
        ACHTUNG: Nach der Funktion darf auf message nicht mehr
                 zugegriffen werden.
```

```
HINWEIS
      In der Message wird der Eintrag message->mn_Node.ln_Typ auf
      NTYP_REPLYMSG gesetzt.
      Anders als beim Amiga muß der Eintrag message->mn_ReplyPort
      immer gesetzt sein.

SIEHE AUCH
      pOS_GetMsg(), pOS_PutMsg()

AMIGA FUNKTION
      VOID ReplyMsg(struct Message *message);
```

1.33 pexec.library/pOS_WaitPort()

```
PROTOTYP
      __ARID__ pOS_Message *res = pOS_WaitPort
        (
          pOS_ExecBase *execbase,
          pOS_MsgPort *msgport
        );

FUNKTION
      Warten auf die nächste Message von einem Message-Port

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      msgport (_R_A0)
        Zeiger auf den Message-Port, auf den bis zum Eintreffen
        der nächsten Message gewartet werden soll

ERGEBNIS
      res (_R_D0)
        immer Zeiger auf die erste empfangene Message im Port
        Die Nachricht muß aber mittels pOS_GetMsg() vom Port
        abgeholt werden!

HINWEIS
      Liegt beim Funktionsaufruf bereits eine Nachricht im Port,
      kehrt die Funktion sofort zurück.
      Trifft nur ein Signal (pOS_SendSignal()) ohne einer Nachricht
      ein, kehrt die Funktion nicht zurück!

SIEHE AUCH
      pOS_AddPort(), pOS_PutMsg(), pOS_WaitSignal(), pOS_SendSignal()

AMIGA FUNKTION
      struct Message *WaitPort(struct MsgPort *msgport);
```

1.34 pexec.library/pOS_FindPort()

```
PROTOTYP
      pOS_MsgPort *res = pOS_FindPort
        (
          pOS_ExecBase *execbase,
          const CHAR *name
        );

FUNKTION
      Suchen eines Message-Ports über seinen Namen in der Systemliste,
      dabei wird zwischen Groß- und Kleinschreibung unterschieden.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      name (_R_A0)
        Namen des Message-Ports, der gesucht werden soll

ERGEBNIS
      res (_R_D0)
        Zeiger auf den gefundenen Message-Port,
        oder NULL

SIEHE AUCH
      pOS_CreatePort(), pOS_AddPort(), pOS_WaitPort()

AMIGA FUNKTION
      struct MsgPort *FindPort(STRPTR name);
```

1.35 pexec.library/pOS_CreatePort()

```
PROTOTYP
      __ARID__ pOS_MsgPort *res = pOS_CreatePort
        (
          pOS_ExecBase *execbase,
          const CHAR *name,
          SLONG priority
        );

FUNKTION
      Anlegen und installieren einer Message-Port-Struktur

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      name (_R_A0)
        Name für den öffentlichen Message-Port oder NULL für
        einen privaten Message-Port. Wird ein Name angegeben,
        wird der Message-Port selbständig mittels pOS_AddPort()
        in die Systemliste eingehängt
      priority (_R_D0)
        Priorität des Message-Ports im Bereich von -128 bis +127,
        normalerweise 0. Bestimmt die Position innerhalb der Systemliste.
```

```
ERGEBNIS
      res (_R_D0)
        Zeiger auf den reservierten Message-Port, der mittels
        pOS_DeletePort() wieder freigegeben werden muß,
        oder NULL im Fehlerfall (Speichermangel/Signalbit)

SIEHE AUCH
      pOS_DeletePort(), pOS_FindPort()

AMIGA FUNKTION
      struct MsgPort *CreatePort(STRPTR name, LONG priority);
```

1.36 pexec.library/pOS_DeletePort()

```
PROTOTYP
      VOID pOS_DeletePort
        (
          pOS_ExecBase *execbase,
          __ARID__ pOS_MsgPort *msgport
        );

FUNKTION
      Freigeben einer Message-Port-Struktur

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      msgport (_R_A0)
        Zeiger auf den Message-Port, der freigegeben werden soll.
        Der Message-Port wird, falls notwendig, aus der Systemliste
        ausgehängt (mittels pOS_RemPort()) und die Signalbits freigegeben
        ACHTUNG: Nach der Funktion darf auf den Message-Port nicht
                 mehr zurückgegriffen werden

HINWEIS
      Im Message-Port dürfen keine Nachrichten mehr liegen.
      Das können Sie folgendermaßen sicherstellen:
        struct MsgPort *msgport;
        msgport=pOS_CreateMsgPort(pOS_ExecBase,"msgport",0);
        ...
        pOS_RemPort(pOS_ExecBase,msgport);
        ...
        struct pOS_Message *msg;
        while(msg=pOS_GetMsg(pOS_ExecBase,msgport))
          pOS_ReplyMsg(pOS_ExecBase,msg);
        msgport->mp_Node.ln_Name=NULL; // damit pOS_DeletePort() kein
                                       pOS_RemPort() aufruft
        pOS_DeletePort(pOS_ExecBase,msgport);

SIEHE AUCH
      pOS_CreatePort(), pOS_FindPort()

AMIGA FUNKTION
```

```
      VOID DeletePort(struct MsgPort *msgport);
```

1.37 pexec.library/pOS_ConstructMsgPort()

```
PROTOTYP
      pOS_MsgPort *res = pOS_ConstructMsgPort
        (
          pOS_ExecBase *execbase,
          pOS_MsgPort *msgport
        );

FUNKTION
      Installieren einer Message-Port-Struktur

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      msgport (_R_A0)
        Zeiger auf die Message-Port-Struktur, die installiert werden soll
        Dabei wird automatisch ein Signal-Bit reserviert

ERGEBNIS
      res (_R_D0)
        Übergebener msgport wenn die Struktur installiert wurde,
        sonst NULL

SIEHE AUCH
      pOS_ConstructSigPort(), pOS_DestructMsgPort(), pOS_AddPort()

AMIGA FUNKTION
```

1.38 pexec.library/pOS_DestructMsgPort()

```
PROTOTYP
      VOID pOS_DestructMsgPort
        (
          pOS_ExecBase *execbase,
          pOS_MsgPort *msgport
        );

FUNKTION
      Deinstallieren einer Message-Port-Struktur

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      msgport (_R_A0)
        Zeiger auf die Message-Port-Struktur, die deinstalliert werden soll
        Dabei wird das Signal-Bit freigegeben
```

```
SIEHE AUCH
      pOS_ConstructMsgPort(), pOS_RemPort()

AMIGA FUNKTION
```

1.39 pexec.library/pOS_ConstructSigPort()

```
PROTOTYP
      pOS_MsgPort *res = pOS_ConstructSigPort
        (
          pOS_ExecBase *execbase,
          pOS_MsgPort *msgport,
          ULONG signalbit
        );

FUNKTION
      Installieren einer Message-Port-Struktur mit einem Signal-Bit

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      msgport (_R_A0)
        Zeiger auf die Message-Port-Struktur, die installiert werden soll
      signalbit (_R_D0)
        Signalbit, das in der Message-Port-Struktur eingetragen wird.
        Das Signal-Bit kann mittels pOS_AllocSignal() reserviert worden sein.

ERGEBNIS
      res (_R_D0)
        Übergebener msgport der installierten Struktur.

SIEHE AUCH
      pOS_ConstructMsgPort(), pOS_AddPort()

AMIGA FUNKTION
```

1.40 pexec.library/pOS_OpenLibrary()

```
PROTOTYP
      __ARID__ pOS_Library *res = pOS_OpenLibrary
        (
          pOS_ExecBase *execbase,
          const CHAR *name,
          ULONG version
        );

FUNKTION
      Öffnen einer Library zur Benutzung der Library-Funktionen

      Suchvorgang:
```

```
        - interne Liste der Libraries
        - CurrentDir/libs/...
        - CurrentDir/...
        - ProgDir:libs/...
        - ProgDir:...
        - LIBS:...

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      name (_R_A0)
        Name der Library, die geöffnet werden soll
        Es wird nicht zwischen Groß- und Kleinschreibung unterschieden
        Bei Bedarf wird die Library automatisch von Disk nachgeladen
      version (_R_D0)
        Minimale Versionsnummer der Library; 0 für eine beliebige Version
        Die tatsächlich geöffnete Library-Version kann in res->lib_Version
        ausgelesen werden

ERGEBNIS
      res (_R_D0)
        Zeiger auf die geöffnete Library, die mittels pOS_CloseLibrary()
        wieder geschlossen werden muß,
        NULL im Fehlerfall

HINWEIS
      Es kann auch aus einer Task-Umgebung eine Library nachgeladen werden.
      Als Name kann auch eine vollständige Pfadangabe übergeben werden,
      z.B. "mydir:mylib.library".
      Ist kein vollständiger Path angegeben, so wird zuerst
              in der Library-Liste/Resident-Liste,
              im aktuellen Verzeichnis unter Libs/,
              im Programmverzeichnis unter Libs/,
              im aktuellen Verzeichnis
              im Libs:
      gesucht.
      Intern wird der Zähler res->lib_OpenCnt um eins erhöht, bevor in
      die Open-Funktion der Library gesprungen wird.
      Daraus kann auch ermittelt werden, ob andere Tasks diese Library
      benutzen (Zähler ist größer als 1).

SIEHE AUCH
      pOS_CloseLibrary(), pOS_AddLibrary()

AMIGA FUNKTION
      struct Library *OpenLibrary(STRPTR name, ULONG version);
```

1.41 pexec.library/pOS_CloseLibrary()

```
PROTOTYP
      VOID pOS_CloseLibrary
        (
          pOS_ExecBase *execbase,
          __ARID__ pOS_Library *library
```

```
        );

FUNKTION
      Schließen einer Library

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      library (_R_A0)
        Zeiger auf die von pOS_OpenLibrary() geöffnete Library
        ACHTUNG: Nach dieser Funktion darf auf library bzw. auf die
                 Library-Funktionen nicht mehr zurückgegriffen werden.

HINWEIS
      Nach Aufruf der Close-Funktion der Library wird der Zähler
      library->lib_OpenCnt um eins vermindert.
      Ist dieser Zähler 0 wird beim nächsten Flush-Durchlauf im
      System die Expunge-Funktion der Library angesprungen, damit
      diese aus dem Speicher entfernt wird.

SIEHE AUCH
      pOS_OpenLibrary(), pOS_RemLibrary()

AMIGA FUNKTION
      VOID CloseLibrary(struct Library *library);
```

1.42 pexec.library/pOS_SetLibFunction()

```
PROTOTYP
      pOS_LibraryFunction *res = pOS_SetLibFunction
        (
          pOS_ExecBase *execbase,
          pOS_Library *library,
          SLONG offset,
          const pOS_LibraryFunction *newfunction,
          pOS_LibraryFunction *oldfunction
        );

FUNKTION
      Ändern eines Funktionsvektors innerhalb einer Library

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      library (_R_A0)
        Zeiger auf die Library, dessen Funktionsvektor verändert
        werden soll
      offset (_R_D0)
        Offset der Funktion in der Library, die durch eine neue
        ersetzt werden soll
      newfunction (_R_A1)
        Zeiger auf die neue Funktion
      oldfunction (_R_A2)
        Zeiger auf den Speicherbereich, in den die alte Funktion
```

```
        abgelegt wird

ERGEBNIS
      res (_R_D0)
        Übergebene Zeiger oldfunction, der die bisherige
        Library-Funktion enthält.

HINWEIS
      Die neue Prüfsumme der Library wird automatisch mittels
      pOS_SumLibrary() berechnet.

SIEHE AUCH
      pOS_OpenLibrary(), pOS_AddLibrary()

AMIGA FUNKTION
      APTR SetFunction(struct Library *library, LONG offset, APTR newfunction);
```

1.43 pexec.library/pOS_MakeLibrary()

```
PROTOTYP
      __ARID__ pOS_Library *res = pOS_MakeLibrary
        (
          pOS_ExecBase *execbase,
          const ULONG **functionary,
          APTR structure,
          BOOL (*initfunc)(_R_LB pOS_ExecBase *,_R_A0 pOS_Library *),
          size_t datasize,
          pOS_SegmentLst *segmentlist
        );

FUNKTION
      Erstellen einer Library oder eines Devices

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      functionarray (_R_A0)
        Tabelle mit den Adressen der Funktionen; die Tabelle muß
        mittels ~0 abgeschlossen werden.
      structure (_R_A1)
        sollte NULL sein; wird für interne Verwaltung benutzt
      initfunc (_R_A2)
        Zeiger auf die Initialisierungsroutine der Library,
        dabei wird pOS_ExecBase (_R_LB) und pOS_Library (_R_A0)
        übergeben. Bei Rückgabewert FALSE wird die Library sofort
        wieder aus dem Speicher entfernt und freigegeben.
        Der Prototyp für die Init-Funktion lautet:
        BOOL initfunc(pOS_ExecBase *execbase, pOS_Library *library);
        NULL wenn keine Init-Funktion vorhanden ist.
      datasize (_R_D0)
        Größe des Datenbereichs einschließlich der Library-Struktur
        (sizeof(struct Library)). Dieser Wert wird in res->lib_PosSize
        vermerkt.
      segmentlist (_R_A3)
```

```
        Segmentliste die an die Init-Funktion der Library übergeben wird.

ERGEBNIS
      res (_R_D0)
        Erstellte Library-Struktur, die mittels pOS_AddLibrary() in
        die Systemliste eingefügt werden kann,
        oder NULL im Fehlerfall (Speichermangel)

HINWEIS
      Diese Funktion wird normal nur vom Betriebssystem benutzt,
      bzw. ist bei der Verwendung privater Libraries notwendig.

SIEHE AUCH
      pOS_SumLibrary(), pOS_AddLibrary(), pOS_OpenLibrary()

AMIGA FUNKTION
      MakeLibrary(APTR functionarray, APTR structure, ULONG (*initfunction)(),  ↩
         ULONG datasize, BPTR segmentlist)
```

1.44 pexec.library/pOS_MakeFunctions()

```
PROTOTYP
      size_t res = pOS_MakeFunctions
        (
          pOS_ExecBase *execbase,
          pOS_LibraryFunction *libraryfunction,
          const ULONG **functionarray,
          ULONG *base,
          size_t size
        );

FUNKTION
      Aufbau einer Funktions-Sprungtabelle

PARAMETER
          execbase (_R_LB)
            Zeiger auf pExec-Library
          libraryfunction (_R_A0)
            Obere Startadresse der Sprungtabelle
          functionarray (_R_A1)
            Tabelle mit den Adressen der Funktionen, die mittels ~0
            abgeschlossen sein muß.
          base (_R_A2)
            Basisadresse, zu der alle Funktionen relativ berechnet
            werden; NULL wenn alle Adressen absolut vorliegen.
          size (_R_D0)
      >>> ???

ERGEBNIS
      res (_R_D0)
        Größer der erstellten Sprungtabelle in Bytes

HINWEIS
      Diese Funktion wird normal nur vom Betriebssystem benutzt,
```

```
      bzw. ist bei der Verwendung privater Libraries notwendig.

SIEHE AUCH
      pOS_SumLibrary()

AMIGA FUNKTION
      ULONG MakeFunctions(APTR, APTR, APTR);
```

1.45 pexec.library/pOS_AddLibrary()

```
PROTOTYP
      VOID pOS_AddLibrary
        (
          pOS_ExecBase *execbase,
          pOS_Library *library
        );

FUNKTION
      Einfügen einer Library in die Systemliste

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      library (_R_A0)
        Zeiger auf die installierte Library-Struktur

HINWEIS
      Diese Funktion wird normal nur vom Betriebssystem benutzt,
      bzw. ist bei der Verwendung privater Libraries notwendig.

SIEHE AUCH
      pOS_OpenLibrary(), pOS_RemLibrary()

AMIGA FUNKTION
      VOID AddLibrary(struct Library *library);
```

1.46 pexec.library/pOS_RemLibrary()

```
PROTOTYP
      VOID pOS_RemLibrary
        (
          pOS_ExecBase *execbase,
          pOS_Library *library
        );

FUNKTION
      Versuch eine Library aus der Systemliste zu entfernen

PARAMETER
      execbase (_R_LB)
```

```
        Zeiger auf pExec-Library
      library (_R_A0)
        Zeiger auf die Library, die aus der Systemliste entfernt
        werden soll

HINWEIS
      Diese Funktion wird normal nur vom Betriebssystem benutzt,
      bzw. ist bei der Verwendung privater Libraries notwendig.
      Es wird die Expunge-Funktion der Library aufgerufen, die
      die Library aus dem Speicher entfernen soll, wenn sie nicht mehr
      benutzt wird.

SIEHE AUCH
      pOS_AddLibrary(), pOS_CloseLibrary()

AMIGA FUNKTION
      VOID RemLibrary(struct Library *library);
```

1.47 pexec.library/pOS_SumLibrary()

```
PROTOTYP
      VOID pOS_SumLibrary
        (
          pOS_ExecBase *execbase,
          pOS_Library *library
        );

FUNKTION
      Berechnen der Checksumme einer Library

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      library (_R_A0)
        Zeiger auf die Library, dessen Prüfsumme neu berechnet und
        eingetragen werden soll (in library->lib_Sum)

SIEHE AUCH
      pOS_OpenLibrary(), pOS_AddLibrary(), pOS_MakeLibrary(),
      pOS_MakeFunctions()

AMIGA FUNKTION
          VOID SumLibrary(struct Library *library);

      /*****************************************************************************/ ←
```

1.48 pexec.library/pOS_AllocAbs()

```
PROTOTYP
```

```
      __ARID__ APTR res = pOS_AllocAbs
        (
          pOS_ExecBase *execbase,
          size_t size,
          APTR location
        );

FUNKTION
      Reservieren eines Speicherbereichs an einer absoluten Adresse

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      size (_R_D0)
        Größe des gewünschten Speicherbereichs; wird auf MEM_BLOCKSIZE gerundet
      locatiton (_R_A0)
        Adresse des absoluten Speicherbereichs

ERGEBNIS
      res (_R_D0)
        location wenn der Speicherbereich reserviert wurde,
        sonst NULL

SIEHE AUCH
      pOS_AllocMem(), pOS_AllocVec(), pOS_FreeMem()

AMIGA FUNKTION
      APTR AllocAbs(ULONG size, APTR location);
```

1.49 pexec.library/pOS_Allocate()

```
PROTOTYP
      __ARID__ APTR res = pOS_Allocate
        (
          pOS_ExecBase *execbase,
          pOS_MemHeader *memheader,
          size_t size
        );

FUNKTION
      Anlegen eines Speicherbereichs innerhalb einer privaten Speicherregion

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      memheader (_R_A0)
        Zeiger auf den MemHeader, der den privaten Speicher verwaltet
      size (_R_D0)
        Größe des gewünschten Speicherbereichs; wird auf MEM_BLOCKSIZE
        aufgerundet

ERGEBNIS
      res (_R_D0)
        Zeiger auf den reservierten Speicherbereich, der mittels
```

```
        pOS_Deallocate() wieder freigegeben werden muß;
        NULL im Fehlerfall

SIEHE AUCH
      pOS_Deallocate()

AMIGA FUNKTION
      APTR Allocate(struct MemHeader *memheader, ULONG size);
```

1.50 pexec.library/pOS_Deallocate()

```
PROTOTYP
      VOID pOS_Deallocate
        (
          pOS_ExecBase *execbase,
          pOS_MemHeader *memheader,
          __ARID__ APTR mem,
          size_t size
        );

FUNKTION
      Freigeben eines Speicherbereichs innerhalb einer privaten
      Speicherregion

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      memheader (_R_A0)
        Zeiger auf den MemHeader, der den privaten Speicher verwaltet
      mem (_R_A1)
        Zeiger auf den freizugebenden Speicherbereich, der mittels
        pOS_Allocate() reserviert wurde
      size (_R_D0)
        Größe des freizugebenden Speicherbereichs; wird auf MEM_BLOCKSIZE
        aufgerundet

SIEHE AUCH
      pOS_Allocate()

AMIGA FUNKTION
      VOID Deallocate(struct MemHeader *memheader, APTR mem, ULONG size);
```

1.51 pexec.library/pOS_AllocMem()

```
PROTOTYP
      __ARID__ APTR res = pOS_AllocMem
        (
          pOS_ExecBase *execbase,
          size_t size,
          ULONG flags
```

```
        );

FUNKTION
      Anlegen eines Speicherbereichs

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      size (_R_D0)
        Größe des gewünschten Speicherbereichs; wird auf MEM_BLOCKSIZE
        aufgerundet
      memflags (_R_D1) (enum pOS_MemoryFlag)
        MEMF_ANY        : beliebige Art von Speicher
        MEMF_PUBLIC     : öffentlicher Speicher, der auch von Interrups
                          und anderen Tasks angesprochen werden kann
        MEMF_VMEM       : der Speicherbereich kann vom System ausgelagert
                          werden
                          (Virtuelle Speicherverwaltung)
        MEMF_LOCAL      : für einen Speicherbereich, der bei einem
                          Warmstart nicht installiert wird
        MEMF_CLEAR      : der reservierte Speicherbereich wird mit
                          0 vorbelegt
        MEMF_NO_EXPUNGE : falls kein, mittels AddMemHandler() installierter,
                          Speicher-Handler aufgerufen werden soll, um
                          Speicherplatz freizugeben, wenn das Anlegen dieses
                          Bereiches wegen Speicherplatzmangels fehlschlägt.

ERGEBNIS
      res (_R_D0)
        Adresse des reservierten Speicherbereichs, der mittels
        pOS_FreeMem() wieder freigegeben werden muß;
        NULL im Fehlerfall

HINWEIS
      Nur wenn das Flag MEMF_VMEM gesetzt ist, kann der Speicherblock
      vom Betriebsystem, bei Bedarf, ausgelagert werden. Alle anderen
      Flags verwenden festen, nicht-virtuell-fähigen Speicher.

SIEHE AUCH
      pOS_FreeMem(), pOS_AllocVec(), pOS_AvailMem()

AMIGA FUNKTION
      APTR AllocMem(ULONG flags, ULONG size);
```

1.52 pexec.library/pOS_FreeMem()

```
PROTOTYP
      VOID pOS_FreeMem
        (
          pOS_ExecBase *execbase,
          __ARID__ APTR mem,
          size_t size
        );
```

```
FUNKTION
      Freigeben eines Speicherbereichs

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      mem (_R_A0)
        Adresse des freizugebenden Speicherbereichs, der mittels
        pOS_AllocMem() reserviert wurde
        ACHTUNG: auf den Speicher darf nach dieser Funktion nicht
                 mehr zugegriffen werden!
      size (_R_D0)
        Größe des reservierten Speicherbereichs;  wird auf MEM_BLOCKSIZE
        aufgerundet
        ACHTUNG: die Größe muß genau der bei pOS_AllocMem() angegebenen
                 entsprechen

HINWEIS
      Es darf nur Speicher freigegeben werden, welcher auch wirklich
      mit pOS_AllocMem() reserviert wurde. Der Speicher darf auch nur
      einmal wieder freigeben werden.

SIEHE AUCH
      pOS_AllocMem(), pOS_FreeVec()

AMIGA FUNKTION
      VOID FreeMem(APTR mem, ULONG size);
```

1.53 pexec.library/pOS_AllocVec()

```
PROTOTYP
      __ARID__ APTR res = pOS_AllocVec
        (
          pOS_ExecBase *execbase,
          size_t size,
          ULONG memflags
        );

FUNKTION
      Anlegen eines Speicherbereichs

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      size (_R_D0)
        Größe des gewünschten Speicherbereichs; wird auf MEM_BLOCKSIZE
        aufgerundet
      flags (_R_D1) (enum pOS_MemoryFlag)
        MEMF_ANY        : beliebige Art von Speicher
        MEMF_PUBLIC     : öffentlicher Speicher, der auch von Interrups
                          und anderen Tasks angesprochen werden kann
        MEMF_VMEM       : der Speicherbereich kann vom System ausgelagert
                          werden
                          (Virtuelle Speicherverwaltung)
```

```
        MEMF_LOCAL      : für einen Speicherbereich, der bei einem
                          Warmstart nicht installiert wird
        MEMF_CLEAR      : der reservierte Speicherbereich wird mit
                          0 vorbelegt
        MEMF_NO_EXPUNGE : falls kein mittels AddMemHandler() installierter
                          Speicher-Handler aufgerufen werden soll, um
                          Speicherplatz freizugeben, wenn das Anlegen dieses
                          Bereiches wegen Speicherplatzmangels fehlschlägt.

ERGEBNIS
      res (_R_D0)
        Adresse des reservierten Speicherbereichs, der mittels
        pOS_FreeVec() wieder freigegeben werden muß;
        NULL im Fehlerfall

HINWEIS
      Diese Funktion ist gleichbedeutend mit pOS_AllocMem().
      Zusätzlich wird aber die Größe des reservierten Speichers
      vermerkt und braucht daher bei pOS_FreeVec() nicht mehr
      angegeben zu werden. Daduch können Fehler beim Freigeben
      mittels einer falschen Größe ausgeschlossen werden.

SIEHE AUCH
      pOS_FreeVec(), pOS_AllocMem(), pOS_AvailMem()

AMIGA FUNKTION
      APTR AllocVec(ULONG size, ULONG flags);
```

1.54 pexec.library/pOS_FreeVec()

```
PROTOTYP
      VOID pOS_FreeVec
        (
          pOS_ExecBase *execbase,
          __ARID__ APTR mem
        );

FUNKTION
      Freigeben eines Speicherbereichs

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      mem (_R_A0)
        Adresse des freizugebenden Speicherbereichs, der mittels
        pOS_AllocVec() reserviert wurde
        ACHTUNG: auf den Speicher darf nach dieser Funktion nicht
                 mehr zugegriffen werden!

SIEHE AUCH
      pOS_AllocVec(), pOS_FreeMem()

AMIGA FUNKTION
```

```
      VOID FreeVec(APTR mem);
```

1.55 pexec.library/pOS_AvailMem()

```
PROTOTYP
      size_t res = pOS_AvailMem
        (
          pOS_ExecBase *execbase,
          ULONG flags
        );

FUNKTION
      Ermitteln des freien Speichers

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      flags (_R_D0) (enum pOS_MemoryFlag)
        MEMF_VMEM       : Virtuelle Speicherverwaltung
        MEMF_LARGEST    : für den größten zusammenhängenden Speicherblock
        MEMF_TOTAL      : für die gesamte Speichergröße (freier und
                          benutzter Speicher)

ERGEBNIS
      res (_R_D0)
        Größe des freien Speichers in Bytes, von flags abhängig.

SIEHE AUCH
      pOS_AddMemList(), pOS_AllocMem(), pOS_AllocVec()

AMIGA FUNKTION
      ULONG AvailMem(ULONG flags);
```

1.56 pexec.library/pOS_AddMemList()

```
PROTOTYP
      VOID pOS_AddMemList
        (
          pOS_ExecBase *execbase,
          size_t size,
          ULONG flags,
          SLONG priority,
          APTR mem,
          const CHAR *name
        );

FUNKTION
      Hinzufügen eines freien Speicherbereis zu der Systemspeicherliste

PARAMETER
```

```
      execbase (_R_LB)
        Zeiger auf pExec-Library
      size (_R_D0)
        Größe des Speicherbereichs in Byte
      flags (_R_D1)

      priority (_R_D2)
        Priorität des Speichers; wird pOS_AllocMem() ... mittels
        MEMF_ANY aufgerufen, wird der Speicher mit der höchsten
        Priorität zuerst reserviert (soweit die gewünschte Größe
        verfügbar ist)
      mem (_R_A0)
        Adresse des neuen Speicherbereichs
      name (_R_A1)
        Name für den neuen Speicherbereich oder NULL

SIEHE AUCH
      pOS_AllocMem(), pOS_AllocVec(), pOS_AvailMem()

AMIGA FUNKTION
      VOID AddMemList(ULONG size, ULONG flags, LONG priority, APTR mem, STRPTR  ↩
         name);
```

1.57 pexec.library/pOS_AllocEntry()

```
PROTOTYP
      __ARID__ pOS_MemList *res = pOS_AllocEntry
        (
          pOS_ExecBase *execbase,
          pOS_MemPool *mempool,
          pOS_MemList *memlist
        );

FUNKTION
      Anlegen mehrerer Speicherbereiche

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      mempool (_R_A0)
        Zeiger auf den MemPool, aus dem der Speicher besorgt werden
        soll oder NULL
      memlist (_R_A1)
        Zeiger auf die Speicherliste, in der die zu reservierenden
        Speicherblöcke und deren Attribute vermerkt sind

ERGEBNIS
      res (_R_D0)
        Neu reservierte MemList, in der die Adressen und Größen der
        einzelnen Speicherbereiche vermerkt ist;
        NULL im Fehlerfall.
        Der Speicher kann mittels pOS_FreeEntry() freigegeben werden.

SIEHE AUCH
```

```
      pOS_FreeEntry()

AMIGA FUNKTION
      struct MemList *AllocEntry(struct MemList *memlist);
```

1.58 pexec.library/pOS_FreeEntry()

```
PROTOTYP
      VOID pOS_FreeEntry
        (
          pOS_ExecBase *execbase,
          pOS_MemPool *mempool,
          __ARID__ pOS_MemList *memlist
        );

FUNKTION
      Freigeben mehrerer Speicherbereiche

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      mempool (_R_A0)
        Zeiger auf den MemPool, an den der Speicher zurückgegeben werden
        soll oder NULL
      memlist (_R_A1)
        Zeiger auf die Speicherliste, in der die freizugebenden
        Speicherblöcke und Größen vermerkt sind

SIEHE AUCH
      pOS_AllocEntry()

AMIGA FUNKTION
          VOID FreeEntry(struct MemList *memlist);

      /*****************************************************************************/ ←
```

1.59 pexec.library/pOS_InitMemPool()

```
PROTOTYP
      VOID pOS_InitMemPool
        (
          pOS_ExecBase *execbase,
          pOS_MemPool *mempool,
          size_t size,
          ULONG flags
        );

FUNKTION
      Installieren einer MemPool-Struktur
```

```
PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      mempool (_R_A0)
        Zeiger auf den privaten MemPool-Verwalter
      size (_R_D0)
        Größe des kleinsten Speicherblocks im Pool
      flags (_R_D1) (enum pOS_MemoryFlag)
        Art des privat verwalteten Speichers
        MEMF_ANY        : beliebige Art von Speicher
        MEMF_PUBLIC     : öffentlicher Speicher, der auch von Interrups
                          und anderen Tasks angesprochen werden kann
        MEMF_VMEM       : der Speicherbereich kann vom System ausgelagert
                          werden
                          (Virtuelle Speicherverwaltung)

HINWEIS
      Die beiden Funktions-Zeiger mpl_Alloc und mpl_Free zeigen
      nach dieser Funktion auf Systemroutinen. Soll eine eigene Speicher-
      verwaltung stattfinden, können hier andere Routinen eingehängt
      werden. Diese gelten in den Funktionen pOS_AllocPoolMem()/
      pOS_AllocPoolVec()/pOS_FreePoolMem()/pOS_FreePoolVec() wenn
      ein Speicher aus der globalen Systemspeicherliste reserviert
      werden muß.

SIEHE AUCH
      pOS_AllocPoolMem(), pOS_AllocPoolVec(), pOS_FreePoolAll()

AMIGA FUNKTION
      APTR CreatePool(ULONG flags, ULONG, ULONG size);
```

1.60 pexec.library/pOS_AllocPoolMem()

```
PROTOTYP
      __ARID__ APTR res = pOS_AllocPoolMem
        (
          pOS_ExecBase *execbase,
          pOS_MemPool *mempool,
          size_t size,
          ULONG flags
        );

FUNKTION
      Reservieren eines Speicherbereichs innerhalb eines privaten
      Speicher-Pools

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      mempool (_R_A0)
        Zeiger auf den privaten MemPool-Verwalter
      size (_R_D0)
        Größe des zu reservierenden Speicherbereichs
```

```
      flags (_R_D1) (enum pOS_MemoryFlag)
        MEMF_CLEAR      : der reservierte Speicherbereich wird mit
                          0 vorbelegt
        MEMF_NO_EXPUNGE : falls kein mittels AddMemHandler() installierter
                          Speicher-Handler aufgerufen werden soll, um
                          Speicherplatz freizugeben, wenn das Anlegen dieses
                          Bereiches wegen Speicherplatzmangels fehlschlägt.
        MEMF_ANY, MEMF_PUBLIC, MEMF_VMEM entsprechend der Angabe bei
        pOS_InitMemPool()

ERGEBNIS
      res (_R_D0)
        Adresse des reservierten Speicherbereichs, der mittels
        pOS_FreePoolMem() oder pOS_FreePoolAll() wieder freigegeben
        werden muß; NULL im Fehlerfall.

HINWEIS
      Kann der Speicherblock nicht aus dem Pool reserviert werden,
      wird er aus der Systemspeicherliste reserviert und in die Pool-
      Verwaltung aufgenommen.

SIEHE AUCH
      pOS_FreePoolMem(), pOS_AllocPoolVec(), pOS_InitMemPool()

AMIGA FUNKTION
      APTR AllocPooled(APTR mempool, ULONG size);
```

1.61 pexec.library/pOS_FreePoolMem()

```
PROTOTYP
      VOID pOS_FreePoolMem
        (
          pOS_ExecBase *execbase,
          pOS_MemPool *mempool,
          __ARID__ APTR mem,
          size_t size
        );

FUNKTION
      Freigeben eines Speicherbereichs innerhalb eines privaten
      Speicher-Pools

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      mempool (_R_A0)
        Zeiger auf den privaten MemPool-Verwalter
      mem (_R_A1)
        Adresse des freizugebenden Speicherbereichs, der mittels
        pOS_AllocPoolMem() reserviert wurde
      size (_R_D0)
        Größe des freizugebenden Speicherbereichs

SIEHE AUCH
```

```
      pOS_AllocPoolMem(), pOS_FreePoolAll()

AMIGA FUNKTION
      VOID FreePooled(APTR mempool, APTR mem, ULONG size);
```

1.62 pexec.library/pOS_FreePoolAll()

```
PROTOTYP
      VOID pOS_FreePoolAll
        (
          pOS_ExecBase *execbase,
          pOS_MemPool *mempool
        );

FUNKTION
      Freigeben aller mittels pOS_AllocPoolMem() oder pOS_AllocPoolVec()
      reservierten Speicherblöcke aus dem privaten Speicherpool.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      mempool (_R_A0)
        Zeiger auf den privaten MemPool-Verwalter, dessen kompletter
        reservierter Speicher freigegeben werden soll

HINWEIS
      Nach der Funktion ist der komplette Pool in dem Zustand,
      wie er nach pOS_InitMemPool() vorliegt.

SIEHE AUCH
      pOS_AllocPoolMem(), pOS_AllocPoolVec(), pOS_InitMemPool()

AMIGA FUNKTION
      VOID DeletePool(APTR mempool);
```

1.63 pexec.library/pOS_AllocPoolVec()

```
PROTOTYP
      __ARID__ APTR res = pOS_AllocPoolVec
        (
          pOS_ExecBase *execbase,
          pOS_MemPool *mempool,
          size_t size,
          ULONG flags
        );

FUNKTION
      Reservieren eines Speicherbereichs innerhalb eines privaten
      Speicher-Pools
```

```
PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      mempool (_R_A0)
        Zeiger auf den privaten MemPool-Verwalter
      size (_R_D0)
        Größe des zu reservierenden Speicherbereichs
      flags (_R_D0) (enum pOS_MemoryFlag)
        MEMF_CLEAR      : der reservierte Speicherbereich wird mit
                          0 vorbelegt
        MEMF_NO_EXPUNGE : falls kein mittels AddMemHandler() installierter
                          Speicher-Handler aufgerufen werden soll, um
                          Speicherplatz freizugeben, wenn das Anlegen dieses
                          Bereiches wegen Speicherplatzmangels fehlschlägt.
        MEMF_ANY, MEMF_PUBLIC, MEMF_VMEM entsprechend der Angabe bei
        pOS_InitMemPool()

HINWEIS
      Kann der Speicherblock nicht aus dem Pool reserviert werden,
      wird er aus der Systemspeicherliste reserviert und in die Pool-
      Verwaltung aufgenommen.

ERGEBNIS
      res (_R_D0)
        Adresse des reservierten Speicherbereichs, der mittels
        pOS_FreePoolVec() oder pOS_FreePoolAll() wieder freigegeben
        werden muß; NULL im Fehlerfall.

SIEHE AUCH
      pOS_FreePoolVec(), pOS_AllocPoolMem(), pOS_InitMemPool()

AMIGA FUNKTION
```

1.64 pexec.library/pOS_FreePoolVec()

```
PROTOTYP
      VOID pOS_FreePoolVec
        (
          pOS_ExecBase *execbase,
          pOS_MemPool *mempool,
          __ARID__ APTR mem
        );

FUNKTION
      Freigeben eines Speicherbereichs innerhalb eines privaten
      Speicher-Pools

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      mempool (_R_A0)
        Zeiger auf den privaten MemPool-Verwalter
      mem (_R_A1)
        Adresse des freizugebenden Speicherbereichs, der mittels
```

```
        pOS_AllocPoolVec() reserviert wurde

SIEHE AUCH
      pOS_AllocPoolVec(), pOS_FreePoolAll()

AMIGA FUNKTION
```

1.65 pexec.library/pOS_CopyMem()

```
PROTOTYP
      VOID pOS_CopyMem
        (
          pOS_ExecBase *execbase,
          const APTR source,
          APTR dest,
          size_t size
        );

FUNKTION
      Kopieren eines Speicherbereichs

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      source(_R_A0)
        Start-Adresse des zu kopierenden Bereichs
      dest (_R_A1)
        Ziel-Adresse für das Kopieren
      size (_R_D0)
        Größe des zu kopierenden Speicherbereichs

AMIGA FUNKTION
      VOID CopyMem(APTR source, APTR dest, ULONG size);
```

1.66 pexec.library/pOS_AddMemHandler()

```
PROTOTYP
      VOID pOS_AddMemHandler
        (
          pOS_ExecBase *execbase,
          pOS_Callback *callback
        );

FUNKTION
      Hinzufügen eines Speicher-Handlers zum System

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      callback (_R_A0)
```

```
        Zeiger auf den Callback, der den Speicher-Handler beschreibt.

SIEHE AUCH
      pOS_RemMemHandler()

AMIGA FUNKTION
      VOID AddMemHandler(struct Interrupt *);
```

1.67 pexec.library/pOS_RemMemHandler()

```
PROTOTYP
      VOID pOS_RemMemHandler
        (
          pOS_ExecBase *execbase,
          pOS_Callback *callback
        );

FUNKTION
      Entfernen eines (eigenen) Speicher-Handlers aus dem System

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      callback (_R_A0)
        Zeiger auf den Callback, der den Speicher-Handler beschreibt.

SIEHE AUCH
      pOS_AddMemHandler()

AMIGA FUNKTION
          VOID RemMemHandler(struct Interrupt *);

      /*****************************************************************************/ ←
```

1.68 pexec.library/pOS_InitSemaphore()

```
PROTOTYP
      VOID pOS_InitSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore
        );

FUNKTION
      Installieren einer Semaphore-Struktur

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
```

```
      semaphore (_R_A0)
        Zeiger auf die zu installierende Semaphore-Struktur.

HINWEIS
      Im Feld semaphore->ss_Link.ln_Name kann ein beliebiger Name
      eingetragen werden, wodurch die Semaphore über pOS_FindSemaphore()
      gefunden werden kann.

      Im Debugmodus wird bei Semaphorefehlern zusätzlich der Eintrag
      semaphore->ss_Link.ln_Name mit ausgegeben.

SIEHE AUCH
      pOS_ObtainSemaphore(), pOS_AttemptSemaphore(),
      pOS_ObtainSemaphoreShared(), pOS_AddSemaphore()

AMIGA FUNKTION
      VOID InitSemaphore(struct SignalSemaphore *semaphore);
```

1.69 pexec.library/pOS_ObtainSemaphore()

```
PROTOTYP
      VOID pOS_ObtainSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore
        );

FUNKTION
      Sperren einer Semaphore für exclusiven Zugriff

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore

HINWEIS
      Ist in semaphore->ss_Flags das Flag SEMF_TPEnqueue gesetzt und
      wollen mehrere Task die Semaphore locken, wird entsprechend
      der Taskpriorität, der Task mit der höchsten Priorität bevorzugt.

      Die Funktion wartet solange, bis die Semaphore exclusiv reserviert
      werden kann. Der gleiche Task kann diese Funktion mehrfach auf-
      rufen, wobei sofort zurückgekehrt wird.

SIEHE AUCH
      pOS_ReleaseSemaphore()

AMIGA FUNKTION
      VOID ObtainSemaphore(struct SignalSemaphore *semaphore);
```

1.70 pexec.library/pOS_ReleaseSemaphore()

```
PROTOTYP
      VOID pOS_ReleaseSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore
        );

FUNKTION
      Aufheben einer Semaphore-Sperre

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die gesperrte Semaphore

SIEHE AUCH
      pOS_ObtainSemaphore(), pOS_ObtainSemaphoreShared(),
      pOS_AttemptSemaphore()

AMIGA FUNKTION
      VOID ReleaseSemaphore(struct SignalSemaphore *semaphore);
```

1.71 pexec.library/pOS_AttemptSemaphore()

```
PROTOTYP
      BOOL pOS_AttemptSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore
        );

FUNKTION
      Versuch der Sperrung einer Semaphore für den exclusiven Zugriff.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore

ERGEBNIS
      res (_R_D0)
        TRUE wenn die Semaphore für den exclusiven Zugriff
        gesperrt wurde. Dann muß die Sperrung mittels pOS_ReleaseSemaphore()
        wieder aufgehoben werden.
        FALSE wenn die Semaphore schon von einem anderen Task gesperrt ist

SIEHE AUCH
      pOS_ReleaseSemaphore(), pOS_ObtainSemaphore(), pOS_AttemptTimeSemaphore()
```

```
AMIGA FUNKTION
      LONG AttemptSemaphore(struct SignalSemaphore *semaphore);
```

1.72 pexec.library/pOS_ObtainSemaphoreShared()

```
PROTOTYP
      VOID pOS_ObtainSemaphoreShared
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore
        );

FUNKTION
      Sperren einer Semaphore für nichtexclusiven Zugriff.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore

SIEHE AUCH
      pOS_ReleaseSemaphore(), pOS_ObtainSemaphore(),
      pOS_AttemptSemaphore()

AMIGA FUNKTION
      VOID ObtainSemaphoreShared(struct SignalSemaphore *semaphore);
```

1.73 pexec.library/pOS_ObtainSemaphoreList()

```
PROTOTYP
      VOID pOS_ObtainSemaphoreList
        (
          pOS_ExecBase *execbase,
          pOS_ExList *list
        );

FUNKTION
      Sperren mehrerer Semaphoren für exclusiven Zugriff

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      list (_R_A0)
        Zeiger auf die installierte Liste mit den zu sperrenden
        Semaphoren

HINWEIS
      Die Funktion kehrt erst dann zurück, wenn alle Semaphoren aus
```

```
      der Liste gesperrt werden konnten.

SIEHE AUCH
      pOS_ReleaseSemaphoreList()

AMIGA FUNKTION
      VOID ObtainSemaphoreList(struct List *list);
```

1.74 pexec.library/pOS_ReleaseSemaphoreList()

```
PROTOTYP
      VOID pOS_ReleaseSemaphoreList
        (
          pOS_ExecBase *execbase,
          pOS_ExList *list
        );

FUNKTION
      Aufheben mehrerer Semaphore-Sperren

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      list (_R_A0)
        Zeiger auf die Liste mit den gesperrten Semaphoren

SIEHE AUCH
      pOS_ObtainSemaphoreList()

AMIGA FUNKTION
      VOID ReleaseSemaphoreList(struct List *list);
```

1.75 pexec.library/pOS_FindSemaphore()

```
PROTOTYP
      pOS_Semaphore *res = pOS_FindSemaphore
        (
          pOS_ExecBase *execbase,
          const CHAR *name
        );

FUNKTION
      Suchen einer Semaphore in der Systemliste anhand des Namens

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      name (_R_A0)
        Name der Semaphore, nach der gesucht werden soll.
```

```
ERGEBNIS
      res (_R_D0)
        Zeiger auf die gefundene Semaphore,
        oder NULL wenn keine entsprechende Semaphore in der Systemliste
        vorhanden ist.

SIEHE AUCH
      pOS_AttemptSemaphore(), pOS_ObtainSemaphore(),
      pOS_ObtainSemaphoreShared()

AMIGA FUNKTION
      struct SignalSemaphore *FindSemaphore(STRPTR name);
```

1.76 pexec.library/pOS_AddSemaphore()

```
PROTOTYP
      VOID pOS_AddSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore
        );

FUNKTION
      Einfügen einer Semaphore in die Systemliste

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die einzufügende Semaphore
        Im Feld semaphore->ss_Link.ln_Name kann der Name der Semaphore
        angegeben werden.

SIEHE AUCH
      pOS_RemSemaphore(), pOS_FindSemaphore()

AMIGA FUNKTION
      VOID AddSemaphore(struct SignalSemaphore *semaphore);
```

1.77 pexec.library/pOS_RemSemaphore()

```
PROTOTYP
      VOID pOS_RemSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore
        );

FUNKTION
      Entfernen einer Semaphore aus der Systemliste
```

```
PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die zu entferndende Semaphore

SIEHE AUCH
      pOS_AddSemaphore(), pOS_FindSemaphore()

AMIGA FUNKTION
      VOID RemSemaphore(struct SignalSemaphore *semaphore);
```

1.78 pexec.library/pOS_ProcureSemaphore()

```
PROTOTYP
      VOID pOS_ProcureSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore,
          pOS_Message *message
        );

FUNKTION
      Asynchrones Sperren einer Semaphore für exclusiven
      Zugriff. Der Task wird dabei nicht angehalten.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore
      message (_R_A1)
        Message, die zurückgeschickt wird, wenn die Signal-
        Semaphore gesperrt wurde.

EXAMPLE
      {
        extern pOS_Semaphore *Semaphore;
        pOS_Message Message;
        pOS_MsgPort MsgPort;

        ww_InitNode(&Message.mn_Node);
        pOS_ConstructMsgPort(&MsgPort);
        Message.mn_ReplyPort=&MsgPort;
        ...
        /* Semaphore asynchron anfordern */
        pOS_ProcureSemaphore(Semaphore,&Message);
        ...
        ... do something
        ...

        /* Abbruch des asynchronen Semaphore-Locken */
        if(...) pOS_VacateSemaphore(Semaphore,&Message);
```

```
        else {
          /* Warten bis Semaphore gesperrt ist */
          pOS_WaitPort(&MsgPort);
          /* ReplyPort leeren: Semaphore-Nachricht */
          pOS_GetMsg(&MsgPort);
          ...
          ...
          /* Semaphore wieder freigeben */
          /* pOS_ReleaseSemaphore() ist ebenfalls möglich */
          pOS_VacateSemaphore(Semaphore,&Message);
        }
        ...
        pOS_DestructMsgPort(&MsgPort);
      }

SIEHE AUCH
      pOS_ObtainSemaphore(), pOS_ReleaseSemaphore(),
      pOS_VacateSemaphore()

AMIGA FUNKTION
```

1.79 pexec.library/pOS_VacateSemaphore()

```
PROTOTYP
      VOID pOS_VacateSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore,
          pOS_Message *message
        );

FUNKTION
      Abbrechen einer asynchronen Semaphore-Anforderung oder
      Aufheben der bestehenden Semaphore-Sperre.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore
      message (_R_A1)
        Message die an pOS_ProcureSemaphore() übergeben wurde

SIEHE AUCH
      pOS_ProcureSemaphore()

AMIGA FUNKTION
```

1.80 pexec.library/pOS_AttemptProcureSemaphore()

```
PROTOTYP
      BOOL res = pOS_AttemptProcureSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore,
          pOS_Message *message
        );

FUNKTION
      Versuch der sofortigen Sperrung einer Semaphore für
      den exclusiven Zugriff. Ist das nicht möglich, wird die
      Semaphore asynchron gesperrt.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore
      message (_R_A1)
        Message, die zurückgeschickt wird, wenn die Signal-
        Semaphore gesperrt wurde.

ERGEBNIS
      res (_R_D0)
        TRUE wenn die Semaphore sofort gesperrt werden konnte
        (entspricht pOS_ObtainSemaphore()-Verhalten),
        sonst FALSE, dann wird die Semaphore asynchron gesperrt
        (entspricht pOS_ProcureSemaphore()-Verhalten).
        Bei TRUE wird KEINE Nachricht zurückgesendet !!!

SIEHE AUCH
      pOS_ProcureSemaphore(), pOS_VacateSemaphore()

AMIGA FUNKTION
```

1.81 pexec.library/pOS_AttemptTimeSemaphore()

```
PROTOTYP
      BOOL res = pOS_AttemptTimeSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore,
          ULONG micros
        );

FUNKTION
      Versuch der Sperrung einer Semaphore für den exclusiven Zugriff.
      Falls die Semaphore noch von anderen Task benutzt wird, wird
      max. micros gewartet und dann die Funktion abgebrochen.

PARAMETER
      execbase (_R_LB)
```

```
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore
      micros (_R_D0)
        Maximale Wartezeit in Mikrosekunden, wenn die Semaphore im
        Moment noch von anderen Tasks benutzt wird.

ERGEBNIS
      res (_R_D0)
        TRUE wenn die Semaphore für den exclusiven Zugriff
        gesperrt wurde. Dann muß die Sperrung mittels pOS_ReleaseSemaphore()
        wieder aufgehoben werden.
        FALSE wenn andere Task die Semaphore noch benutzen.

SIEHE AUCH
      pOS_AttemptSemaphore(), pOS_ReleaseSemaphore()

AMIGA FUNKTION
```

1.82 pexec.library/pOS_AddSemaphoreQR()

```
PROTOTYP
      VOID pOS_AddSemaphoreQR
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore,
          pOS_Message *message
        );

FUNKTION
      Installieren einer Benachrichtigung, wenn ein anderer Task
      die angegebene Semaphore locken möchte.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore
      message (_R_A1)
        Message, die an den Aufrufer zurückgeschickt wird, wenn
        ein anderer Task die Semaphore locken möchte.

BEISPIEL
      /* Message und Port vorbereiten */

      pOS_MsgPort msgport;
      pOS_ConstructMsgPort(&msgport);
      struct pOS_Message msg={ {NULL,NULL,NTYP_MESSAGE,0,NULL},&msgport};

      /* Semaphore normal besorgen */
      pOS_Semaphore sema;
      pOS_InitSemaphore(&sema);

      /* ... */
```

```
      pOS_ObtainSemaphore(&sema);
      pOS_AddSemaphoreQR(&sema,&msg);

      for(i=0; i<100; i++)
      {
        /* ... */

        /* prüfen, ob jemand die Semaphore locken möchte */
        if(pOS_GetMsg(&msgport)
        {
          /* die Semaphore freigeben */
          pOS_RemSemaphoreQR(&sema,&msg);
          pOS_ReleaseSemaphore(&sema);
            /* anderen Task reinlassen */
          pOS_ObtainSemaphore(&sema);
          pOS_AddSemaphoreQR(&sema,&msg);
        }
      }

      /* Semaphore wieder freigeben */
      pOS_RemSemaphoreQR(&sema,&msg);
      pOS_ReleaseSemaphore(&sema);

      /* Message und Port freigeben */
      pOS_GetMsg(&msgport);
      pOS_DestructMsgPort(&msgport);

SIEHE AUCH
      pOS_RemSemaphoreQR()

AMIGA FUNKTION
```

1.83 pexec.library/pOS_RemSemaphoreQR()

```
PROTOTYP
      VOID pOS_RemSemaphoreQR
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore,
          pOS_Message *message
        );

FUNKTION
      Entfernen einer Benachrichtigung, die mittels pOS_AddSemaphoreQR()
      installiert wurde.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore
      message (_R_A1)
        Message-Zeiger, wie bei pOS_AddSemaphoreQR()
```

```
SIEHE AUCH
      pOS_AddSemaphoreQR()

AMIGA FUNKTION
```

1.84 pexec.library/pOS_IsObtainSemaphore()

```
PROTOTYP
      BOOL res = pOS_IsObtainSemaphore
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore
        );

FUNKTION
      Prüfen, ob eine Semaphore vom aktuelles Task bereits gesperrt ist.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore

ERGEBNIS
      res (_R_D0)
        TRUE wenn die angegebene Semaphore bereits gesperrt wurde, sonst FALSE.

SIEHE AUCH

AMIGA FUNKTION
```

1.85 pexec.library/pOS_AttemptSemaphoreShared()

```
PROTOTYP
      BOOL res = pOS_AttemptSemaphoreShared
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore
        );

FUNKTION
      Versuch der Sperrung einer Semaphore für den nichtexclusiven Zugriff.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore
```

```
ERGEBNIS
      res (_R_D0)
        TRUE, wenn die Semaphore gesperrt werden konnte;
        sonst FALSE.

SIEHE AUCH
      pOS_ProcureSemaphoreShared(), pOS_AttemptTimeSemaphoreShared()

AMIGA FUNKTION
```

1.86 pexec.library/pOS_ProcureSemaphoreShared()

```
PROTOTYP
      VOID pOS_ProcureSemaphoreShared
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore,
          pOS_Message *message
        );

FUNKTION
      Asynchrones Sperren einer Semaphore für nichtexclusiven
      Zugriff. Der Task wird dabei nicht angehalten.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore
      message (_R_A1)
        Message, die zurückgeschickt wird, wenn die Signal-
        Semaphore gesperrt wurde.

SIEHE AUCH
      pOS_AttemptSemaphoreShared()

AMIGA FUNKTION
```

1.87 pexec.library/pOS_AttemptTimeSemaphoreShared()

```
PROTOTYP
      BOOL res = pOS_AttemptTimeSemaphoreShared
        (
          pOS_ExecBase *execbase,
          pOS_Semaphore *semaphore,
          ULONG micros
        );

FUNKTION
      Versuch der Sperrung einer Semaphore für den nichtexclusiven Zugriff.
```

```
      Falls die Semaphore noch von anderen Task exclusiv benutzt wird, wird
      max. micros gewartet und dann die Funktion abgebrochen.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      semaphore (_R_A0)
        Zeiger auf die installierte Semaphore
      micros (_R_D0)
        Maximale Wartezeit in Mikrosekunden, wenn die Semaphore im
        Moment noch von anderen Tasks exclusiv gelockt ist.

ERGEBNIS
      res (_R_D0)
        TRUE wenn die Semaphore für den nichtexclusiven Zugriff
        gesperrt wurde. Dann muß die Sperrung mittels pOS_ReleaseSemaphore()
        wieder aufgehoben werden.
        FALSE wenn andere Task die Semaphore noch exclusiv benutzen.

SIEHE AUCH
      pOS_ProcureSemaphoreShared()

AMIGA FUNKTION
```

1.88 pexec.library/pOS_DisplayAlert()

```
PROTOTYP
      VOID pOS_DisplayAlert
        (
          pOS_ExecBase *execbase,
          ULONG alertnummer
        );

FUNKTION
      Darstellen einer Warnmeldung

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      alertnummer (_R_D0) (enum pOS_Alerts)
        Nummer der Warnmeldung, vordefinierte Fehlernummern
        sind in der Datei "pExec/Diagnos.h" nachzulesen.
        Nummern ab 0x35000000 stehen für Programmeigene
        Fehlernummern zur Verfügung.

SIEHE AUCH

AMIGA FUNKTION
      VOID Alert(ULONG alertnummer);
```

1.89 pexec.library/pOS_ExecCheckA()

```
PROTOTYP
      BOOL res = pOS_ExecCheckA
        (
          pOS_ExecBase *execbase,
          const pOS_TagItem *taglist
        );

FUNKTION
      Prüfen oder Erfragen der Systemumgebung.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      taglist (_R_A0)
        EXTSTTAG_CPU      : prüft, ob die gewünschte CPU verfügbar ist
        EXTSTTAG_FPU      : prüft, ob die gewünschte FPU verfügbar ist
        EXTSTTAG_MainOSID : prüft, ob die gewünschte pOS-Version läuft

        EXTSTTAG_GetCPU   : liefert die vorhandene CPU-Kennung
        EXTSTTAG_GetFPU   : liefert die vorhandene FPU-Kennung

ERGEBNIS
      res (_R_D0)
        TRUE wenn die Abfrage zutrifft, sonst FALSE.
        Es sollten keine Prüfungen mit Werteermittlungen kombiniert
        werden, da der Rückgabewert dann undefiniert ist.

BEISPIEL
      const struct pOS_TagItem taglist[]=
      {
        EXTSTTAG_MainOSID, pOS_MAINOSID,
        TAG_END
      };

      if(pOS_ExecCheckA(taglist))
      {
        /* ... Programmcode ... */
      }
      /* else falsche pOS-Version */

SIEHE AUCH

AMIGA FUNKTION
```

1.90 pexec.library/pOS_FindResident()

```
PROTOTYP
      pOS_Resident *res = pOS_FindResident
        (
          pOS_ExecBase *execbase,
          const CHAR *name
        );
```

```
FUNKTION
      Suchen eines residenten Moduls über seinen Namen in der Systemliste

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      name (_R_A0)
        Name des zu suchenden Moduls

ERGEBNIS
      res (_R_D0)
        Zeiger auf das gefundene Resident-Modul,
        oder NULL wenn kein entsprechendes Modul existiert.

SIEHE AUCH
      pOS_InitResident()

AMIGA FUNKTION
      struct Residnet *FindResident(STRPTR name);
```

1.91 pexec.library/pOS_InitResident()

```
PROTOTYP
      APTR res = pOS_InitResident
        (
          pOS_ExecBase *execbase,
          const pOS_Resident *resident,
          pOS_SegmentLst *segmentlist
        );

FUNKTION
      Installieren eines residenten Modules

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      resident (_R_A0)
        Zeiger auf die zu installierende Resident-Struktur
      segmentlist (_R_A1)
        Zeiger auf die Segmentliste des zu ladenden Objekts;

ERGEBNIS
      res (_R_D0)
        Rückgabewert der Installierungsfunktion.
        Diese hängt vom installierten Modul ab; normalerweise
        der Zeiger auf eine Library- oder Device-Struktur.
        NULL im Fehlerfall.

SIEHE AUCH
      pOS_FindResident()

AMIGA FUNKTION
          APTR InitResident(struct Resident *resident, BPTR segmentlist);
```

```
      /*****************************************************************************/ ←
```

1.92 pexec.library/pOS_OpenDevice()

```
PROTOTYP
      SBYTE res = pOS_OpenDevice
        (
          pOS_ExecBase *execbase,
          const CHAR *name,
          ULONG unit,
          pOS_IORequest *iorequest,
          ULONG flags,
          ULONG version
        );

FUNKTION
      Öffnen eines Devices zur Benutzung

      Suchvorgang:
        - interne Liste der Devices
        - CurrentDir/devs/...
        - CurrentDir/...
        - ProgDir:devs/...
        - ProgDir:...
        - DEVS:...

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      name (_R_A0)
        Name des zu öffnenden Devices
      unit (_R_D0)
        Unit-Nummer des zu öffnenden Devices
      iorequest (_R_A1)
        Zeiger auf die installierte IORequest-Struktur
      flags (_R_D1)
        Devicespezifische Flags
      version (_R_D2)
        minimale Versionsnummer des zu öffnenden Devices;
        0 für ein beliebiges Device

ERGEBNIS
      res (_R_D0)
        0 wenn das Device offen ist. Dann kann der Device-Basiszeiger
        aus iorequest->io_Device ausgelesen werden;
        sonst eine devicespezifische Fehlernummer, welche auch aus
        iorequest->io_Error ausgelesen werden kann.

HINWEIS
      Die genauen Daten (name, unit, iorequest, flags) können in
      den einzelnen Device-Beschreibungen nachgelesen werden.
```

```
SIEHE AUCH
      pOS_CloseDevice(), pOS_AddDevice(), pOS_BeginIO(), pOS_SendIO(),
      pOS_DoIO()

AMIGA FUNKTION
      LONG OpenDevice(STRPTR name, ULONG unit, struct IORequest *iorequest,  ↩
          ULONG flags);
```

1.93 pexec.library/pOS_CloseDevice()

```
PROTOTYP
      VOID pOS_CloseDevice
        (
          pOS_ExecBase *execbase,
          pOS_IORequest *iorequest
        );

FUNKTION
      Schließen eines Devices

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      iorequest (_R_A0)
        Zeiger auf die installierte IORequest-Struktur des zu
        schließenden Devices.

SIEHE AUCH
      pOS_OpenDevice(), pOS_RemDevice(), pOS_CheckIO(), pOS_WaitIO(),
      pOS_AbortIO()

AMIGA FUNKTION
      VOID CloseDevice(struct Device *device);
```

1.94 pexec.library/pOS_DoIO()

```
PROTOTYP
      SBYTE res = pOS_DoIO
        (
          pOS_ExecBase *execbase,
          pOS_IORequest *iorequest
        );

FUNKTION
      Synchrones Ausführen eines Device-Kommandos und Warten auf
      seine Abarbeitung

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
```

```
      iorequest (_R_A0)
        Zeiger auf die installierte IORequest-Struktur

ERGEBNIS
      res (_R_D0)
        0 bei Erfolg, sonst eine devicespezifische Fehlernummer
        (entsprechend dem Eintrag iorequester->io_Error).

HINWEIS
      Es wird zuerst iorequest->io_Flags auf IOREQF_Quick gesetzt.
      Die Funktion sendet die Message mittels pOS_BeginIO()
      und wartet mit pOS_WaitIO() auf dessen komplette Abarbeitung.
      Danach wird der Fehlercode aus iorequest->io_Error zurück-
      gegeben.

SIEHE AUCH
      pOS_OpenDevice(), pOS_CreateIORequest(), pOS_SendIO(),
      pOS_BeginIO()

AMIGA FUNKTION
      LONG DoIO(struct IORequest *iorequest);
```

1.95 pexec.library/pOS_SendIO()

```
PROTOTYP
      VOID pOS_SendIO
        (
          pOS_ExecBase *execbase,
          __ARID__ pOS_IORequest *iorequest
        );

FUNKTION
      Asynchrones Ausführen eines Device-Kommandos

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      iorequest (_R_A0)
        Zeiger auf die installierte IORequest-Struktur

HINWEIS
      Die Funktion ist gleichbedeutend mit pOS_BeginIO();
      zusätzlich wird IOREQF_Quick aus iorequest->io_Flags
      entfernt, wodurch eine asynchrone Abarbeitung erzwungen
      wird.

SIEHE AUCH
      pOS_OpenDevice(), pOS_CreateIORequest(), pOS_DoIO(),
      pOS_BeginIO(), pOS_CheckIO(), pOS_WaitIO(), pOS_AbortIO()

AMIGA FUNKTION
      LONG SendIO(struct IORequest *iorequest);
```

1.96 pexec.library/pOS_BeginIO()

```
PROTOTYP
      VOID pOS_BeginIO
        (
          pOS_ExecBase *execbase,
          __ARID__ pOS_IORequest *iorequest
        );

FUNKTION
      Asynchrones Ausführen eines Device-Kommandos

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      iorequest (_R_A0)
        Zeiger auf die installierte IORequest-Struktur

HINWEIS
      Die Funktion ist gleichbedeutend mit pOS_SendIO();
      jedoch wird aber der Eintrag iorequest->io_Flags nicht
      verändert.

SIEHE AUCH
      pOS_OpenDevice(), pOS_CreateIORequest(), pOS_SendIO(),
      pOS_CheckIO(), pOS_WaitIO(), pOS_AbortIO(), pOS_DoIO()

AMIGA FUNKTION
      VOID BeginIO(struct IORequest *iorequest);
```

1.97 pexec.library/pOS_CheckIO()

```
PROTOTYP
      pOS_IORequest *res = pOS_CheckIO
        (
          pOS_ExecBase *execbase,
          const pOS_IORequest *iorequest
        );

FUNKTION
      Prüfen ob ein asynchroner IORequest noch läuft (bearbeitet wird)

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      iorequest (_R_A0)
        Zeiger auf die installierte IORequest-Struktur,
        die mittels pOS_BeginIO() oder pOS_SendIO() an
        das Device geschickt wurde.

ERGEBNIS
      res (_R_D0)
```

```
        Wenn das Kommando nicht mehr läuft, wird der Zeiger auf den
        übergebenen iorequest zurückgegeben. Dann kann aus
        iorrequest->io_Error der devicespezifische Fehlercode des
        letzten Kommandos ermittelt werden;
        NULL wenn der Befehl noch bearbeitet wird.

SIEHE AUCH
      pOS_SendIO(), pOS_BeginIO(), pOS_WaitIO(), pOS_AbortIO()

AMIGA FUNKTION
      struct IORequest *CheckIO(struct IORequest *iorequest);
```

1.98 pexec.library/pOS_WaitIO()

```
PROTOTYP
      SBYTE res = pOS_WaitIO
        (
          pOS_ExecBase *execbase,
          pOS_IORequest *iorequest
        );

FUNKTION
      Warten auf das Ende eines asynchronen IORequest

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      iorequest (_R_D0)
        Zeiger auf die installierte IORequest-Struktur,
        die mittels pOS_BeginIO() oder pOS_SendIO() an
        das Device geschickt wurde.

ERGEBNIS
      res (_R_D0)
        0 wenn das Kommando fehlerfrei abgearbeitet wurde;
        sonst eine devicespezifische Fehlernummer
        (entsprechend dem Eintrag iorequester->io_Error).

HINWEIS
      Die Funktion kehrt sofort zurück, wenn das Kommando
      nicht mehr läuft. Zusätzlich wird die eigene Nachricht
      vom ReplyPort entfernt.

SIEHE AUCH
      pOS_SendIO(), pOS_BeginIO(), pOS_CheckIO(), pOS_AbortIO()

AMIGA FUNKTION
      LONG WaitIO(struct IORequest *iorequest);
```

1.99 pexec.library/pOS_AbortIO()

```
PROTOTYP
      VOID pOS_AbortIO
        (
          pOS_ExecBase *execbase,
          pOS_IORequest *iorequest
        );

FUNKTION
      Abbrechen eines asynchronen IORequesters

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      iorequest (_R_A0)
        Zeiger auf die installierte IORequest-Struktur,
        die mittels pOS_BeginIO() oder pOS_SendIO() an
        das Device geschickt wurde.

HINWEIS
      Nach dem Abbrechen des Kommandos, muß mittels pOS_WaitIO()
      auf das Abbrechen des Devices gewartet werden.
      Es ist dem Device überlassen, ob es das Abbrechen von
      Funktionen unterstützt. Daher kann zwischen pOS_AbortIO()
      und pOS_WaitIO() eine längere Zeitspanne liegen.

SIEHE AUCH
      pOS_SendIO(), pOS_BeginIO(), pOS_CheckIO(), pOS_WaitIO()

AMIGA FUNKTION
      VOID AbortIO(struct IORequest *iorequest);
```

1.100 pexec.library/pOS_CreateIORequest()

```
PROTOTYP
      __ARID__ pOS_IORequest *res = pOS_CreateIORequest
        (
          pOS_ExecBase *execbase,
          pOS_MsgPort *msgport,
          size_t size
        );

FUNKTION
   Installieren einer IORequest-Struktur

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      msgport (_R_A0)
        Zeiger auf den eigenen Message-Port, an den die
        Rück-Nachrichten der Devicekommandos geschickt werden
        sollen.
      size (_R_D0)
```

```
        Größe der zu installierenden (devicespezifischen)
        IORequest-Struktur

ERGEBNIS
      res (_R_D0)
        Zeiger auf die reservierte und initialisierte IORequest-
        Struktur, dessen Speicher mittels pOS_DeleteIORequest()
        wieder freigegeben werden muß.

SIEHE AUCH
      pOS_DeleteIORequest(), pOS_BeginIO(), pOS_DoIO(), pOS_SendIO()

AMIGA FUNKTION
```

1.101 pexec.library/pOS_DeleteIORequest()

```
PROTOTYP
      VOID pOS_DeleteIORequest
        (
          pOS_ExecBase *execbase,
          __ARID__ pOS_IORequest *iorequest
        );

FUNKTION
      Deinstallieren einer IORequest-Struktur

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      iorequest (_R_A0)
        Zeiger auf die IORequest-Struktur, die deinstalliert
        werden soll.
        ACHTNG: Nach der Funktion darf auf den Zeiger nicht
                mehr zugegriffen werden.


SIEHE AUCH
      pOS_CreateIORequest()

AMIGA FUNKTION
```

1.102 pexec.library/pOS_AddDevice()

```
PROTOTYP
      VOID pOS_AddDevice
        (
          pOS_ExecBase *execbase,
          pOS_Device *device
        );
```

```
FUNKTION
      Hinzufügen eines Devices in die Systemliste

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      device (_R_A0)
        Zeiger auf die installierte Device-Struktur

SIEHE AUCH
      pOS_RemDevice(), pOS_OpenDevice(), pOS_MakeLibrary()

AMIGA FUNKTION
      VOID AddDevice(struct Device *devcie);
```

1.103 pexec.library/pOS_RemDevice()

```
PROTOTYP
      VOID pOS_RemDevice
        (
          pOS_ExecBase *execbase,
          pOS_Device *device
        );

FUNKTION
      Versuch ein Device aus der Systemliste zu entfernen

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      device (_R_A0)
        Zeiger auf die Device-Stuktur, des zu entfernenden Devices

HINWEIS
      Es wird die Expunge-Funktion des Devices aufgerufen, die
      das Device aus dem Speicher entfernen soll, wenn es nicht mehr
      benutzt wird.

SIEHE AUCH
      pOS_AddDevice(), pOS_CloseDevice()

AMIGA FUNKTION
          VOID RemDevice(struct Device *device);

      /*****************************************************************************/ ←
```

1.104 pexec.library/pOS_OpenResource()

```
PROTOTYP
```

```
      __ARID__ pOS_Resource *res = pOS_OpenResource
        (
          pOS_ExecBase *execbase,
          const CHAR *name
        );

FUNKTION
      Öffnen einer Resource zur Benutzung

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      name (_R_A0)
        Name der zu öffnenden Reseource

ERGEBNIS
      res (_R_D0)
        Zeiger auf die geöffnete Resource, die mittels pOS_CloseResource()
        wieder geschlossen werden muß;
        NULL im Fehlerfall.

SIEHE AUCH
      pOS_CloseResource(), pOS_AddResource()

AMIGA FUNKTION
      APTR OpenResource(STRPTR name);
```

1.105 pexec.library/pOS_CloseResource()

```
PROTOTYP
      VOID pOS_CloseResource
        (
          pOS_ExecBase *execbase,
          __ARID__ pOS_Resource *resource
        );

FUNKTION
      Schließen einer Resource

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      resource (_R_A0)
        Zeiger auf die von pOS_OpenResource() zurückgelieferte
        Resource

HINWEIS
      Entgegen dem Amiga, muß bei pOS eine Resource wieder freigegeben
      werden, wenn sie nicht mehr benötigt wird.

SIEHE AUCH
      OpenResource(), pOS_RemResource()
```

```
AMIGA FUNKTION
```

1.106 pexec.library/pOS_AddResource()

```
PROTOTYP
      VOID pOS_AddResource
        (
          pOS_ExecBase *execbase,
          pOS_Resource *resource
        );

FUNKTION
      Hinzufügen einer Resource in die Systemliste

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      resource (_R_A0)
        Zeiger auf die installierte Resource-Struktur

SIEHE AUCH
      pOS_RemResource(), OpenResource()

AMIGA FUNKTION
      VOID AddResource(APTR resource);
```

1.107 pexec.library/pOS_RemResource()

```
PROTOTYP
      VOID pOS_RemResource
        (
          pOS_ExecBase *execbase,
          pOS_Resource *resource
        );

FUNKTION
      Entfernen einer Resource aus der Systemliste

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      resource(_R_A0)
        Zeiger auf die installierte Resource-Struktur, die
        entfernt werden soll

SIEHE AUCH
      pOS_AddResource()

AMIGA FUNKTION
          VOID RemResource(APTR resource);
```

```
      /*****************************************************************************/ ←
```

1.108 pexec.library/pOS_RawDoFmt()

```
PROTOTYP
      VOID pOS_RawDoFmt
        (
          pOS_ExecBase *execbase,
          pOS_RawDoFmtData *data);

FUNKTION
      Formatieren von Daten in einer Zeichenkette

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      data (_R_A0)
        data->rdft_Format  : enthält die Formatschablone
        data->rdft_Argv    : Zeiger auf Array mit Parameter
        data->rdft_DoFmt   : Funktion, die zur Ausgabe der einzelnen
                             Zeichen aufgerufen wird
        data->rdft_InterFmt: kann auf eine eigene Funktion zur
                             Interpretation der Platzhalter zeigen


        FormatString - wie "C"-Language

         %[flags][width.limit][length]type

        flags  - nur '-' definiert, linksbündig
        width  - Anzahl der auszugebenden Zeichen
          .    - Trenner
        limit  - Ausgabebeschränkung, maximal 'limit' Zeichen ausgeben (%s)
        length - Datenbreite: l=32 bit, h=16 bit
        type   - verwendbare Typen:
              d - dezimal
              i - Integer, wie %d
              u - vorzeichenlos dezimal
              x - hexadezimal
              s - c-string
              c - ein Zeichen
              p - Hex-Dump der Adresse

HINWEIS
      Die DoFmt-Funktion erhält als Parameter den Zeiger auf die
      pOS_RawDoFmtData-Struktur. Das auszugebende Zeichen ist aus
      data->rdft_Data auszulesen.

SIEHE AUCH
      pOS_ReadAsciiFmt(), pOS_WriteAsciiFmt()
      siehe Beispielprogramm "DDevList.c" für die Anwendung
```

```
AMIGA FUNKTION
          APTR RawDoFmt(STRPTR format, APTR data, VOID (*putfunc)(), APTR  ↩
              putdata);
```

1.109 pexec.library/pOS_ReadAsciiFmt()

```
      ### wird noch erweitert ###

PROTOTYP
      const CHAR *res = pOS_ReadAsciiFmt
        (
          pOS_ExecBase *execbase,
          pOS_AsciiFmtData *data,
          const CHAR *str
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      data (_R_A0)
      str (_R_A1)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH
      pOS_WriteAsciiFmt(), pOS_RawDoFmt()

AMIGA FUNKTION
```

1.110 pexec.library/pOS_WriteAsciiFmt()

```
      ### wird noch erweitert ###

PROTOTYP
      CHAR *res = pOS_WriteAsciiFmt
        (
          pOS_ExecBase *execbase,
          const pOS_AsciiFmtData *data,
          CHAR *str,
          size_t size
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      data (_R_A0)
        Zeiger auf die Formatschablone
```

```
      str (_R_A1)

      size (_R_D0)
        Größe des Buffers

ERGEBNIS
      res (_R_D0)

SIEHE AUCH
      pOS_ReadAsciiFmt(), pOS_RawDoFmt()

AMIGA FUNKTION
```

1.111 pexec.library/pOS_AddClass()

```
PROTOTYP
      VOID pOS_AddClass
        (
          pOS_ExecBase *execbase,
          pOS_NClass *class
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)

SIEHE AUCH

AMIGA FUNKTION
```

1.112 pexec.library/pOS_AddLinkClass()

```
PROTOTYP
      VOID pOS_AddLinkClass
        (
          pOS_ExecBase *execbase,
          const pOS_NClass *class,
          pOS_NClass *class2
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)
      class2 (_R_A1)
```

```
SIEHE AUCH

AMIGA FUNKTION
```

1.113 pexec.library/pOS_SubClass()

```
PROTOTYP
      VOID pOS_SubClass
        (
          pOS_ExecBase *execbase,
          pOS_NClass *class
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)

SIEHE AUCH

AMIGA FUNKTION
```

1.114 pexec.library/pOS_CreateObject()

```
PROTOTYP
      APTR res = pOS_CreateObject
        (
          pOS_ExecBase *execbase,
          pOS_NClass *class,
          const CHAR *name,
          ULONG,
          pOS_Method *method
        );

FUNKTION
      Objekt der Klasse erzeugen.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)
      name (_R_A1)
      (_R_D0)
      method (_R_A4)

ERGEBNIS
      res (_R_D0)
```

```
SIEHE AUCH
      pOS_DeleteObject()

AMIGA FUNKTION
```

1.115 pexec.library/pOS_DeleteObject()

```
PROTOTYP
      VOID pOS_DeleteObject
        (
          pOS_ExecBase *execbase,
          APTR object
        );

FUNKTION
      Objekt einer Klasse entfernen.

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      object (_R_A0)

SIEHE AUCH
      pOS_CreateObject()

AMIGA FUNKTION
```

1.116 pexec.library/pOS_DoIMethodA()

```
PROTOTYP
      ULONG res = pOS_DoIMethodA
        (
          pOS_ExecBase *execbase,
          APTR object,
          pOS_Method *method
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      object (_R_A0)
      method (_R_A4)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH
```

```
AMIGA FUNKTION
```

1.117 pexec.library/pOS_DoMMethodA()

```
PROTOTYP
      ULONG res = pOS_DoMMethodA
        (
          pOS_ExecBase *execbase,
          const pOS_Class *class,
          ULONG index,
          APTR object,
          pOS_Method *method
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)
      index (_R_D0)
      object (_R_A1)
      method (_R_A4)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH

AMIGA FUNKTION
```

1.118 pexec.library/pOS_DoVirMethodA()

```
PROTOTYP
      ULONG res = pOS_DoVirMethodA
        (
          pOS_ExecBase *execbase,
          const pOS_Class *class,
          APTR object,
          pOS_Method *method
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)
      object (_R_A1)
```

```
      method (R_A4)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH

AMIGA FUNKTION
```

1.119 pexec.library/pOS_DoAbsMethodA()

```
PROTOTYP
      ULONG res = pOS_DoAbsMethodA
        (
          pOS_ExecBase *execbase,
          const pOS_Class *class,
          APTR object,
          pOS_Method *method
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)
      object (_R_A1)
      method (_R_A4)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH

AMIGA FUNKTION
```

1.120 pexec.library/pOS_OpenClass()

```
PROTOTYP
      pOS_NClass *res = pOS_OpenClass
        (
          pOS_ExecBase *execbase,
          const CHAR *name,
          ULONG version
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
```

```
        Zeiger auf pExec-Library
      name (_R_A0)
      version (_R_D0)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH
      pOS_CloseClass(), pOS_OpenLibrary()

AMIGA FUNKTION
```

1.121 pexec.library/pOS_CloseClass()

```
PROTOTYP
      VOID pOS_CloseClass
        (
          pOS_ExecBase *execbase,
          pOS_NClass *class
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)

SIEHE AUCH
      pOS_OpenClass()

AMIGA FUNKTION
```

1.122 pexec.library/pOS_GetMemberAdr()

```
PROTOTYP
      APTR res = pOS_GetMemberAdr
        (
          pOS_ExecBase *execbase,
          const pOS_Class *class,
          ULONG index,
          APTR object
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)
```

```
      index (_R_D0)
      object (_R_A1)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH

AMIGA FUNKTION
```

1.123 pexec.library/pOS_MoveUpClassPtr()

```
PROTOTYP
      BOOL pOS_MoveUpClassPtr
        (
          pOS_ExecBase *execbase,
          const pOS_Class *class,
          APTR object,
          pOS_Class **newclass,
          APTR *newobject
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)
      object (_R_A1)
      newclass (_R_A2)
      newobject (_R_A3)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH
      pOS_MoveDownClassPtr()

AMIGA FUNKTION
```

1.124 pexec.library/pOS_MoveDownClassPtr()

```
PROTOTYP
      BOOL res = pOS_MoveDownClassPtr
        (
          pOS_ExecBase *execbase,
          const pOS_Class *class,
          APTR object,
          pOS_Class **newclass,
          APTR *newobject
```

```
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)
      object (_R_A1)
      newclass (_R_A2)
      newobject (_R_A3)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH
      pOS_MoveUpClassPtr()

AMIGA FUNKTION
```

1.125 pexec.library/pOS_GetUpObjectAdr()

```
PROTOTYP
      APTR res = pOS_GetUpObjectAdr
        (
          pOS_ExecBase *execbase,
          const pOS_Class *class,
          APTR object,
          ULONG number
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)
      object (_R_A1)
      number (_R_D0)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH

AMIGA FUNKTION
```

1.126 pexec.library/pOS_GetNClass()

```
PROTOTYP
```

```
      const pOS_NClass *res = pOS_GetNClass
        (
          pOS_ExecBase *execbase,
          const pOS_Class *class
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH

AMIGA FUNKTION
```

1.127 pexec.library/pOS_GetIMemberAdr()

```
PROTOTYP
      APTR res = pOS_GetIMemberAdr
        (
          pOS_ExecBase *execbase,
          ULONG memberindex,
          APTR object
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      memberindex (_R_D0)
      object (_R_A0)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH

AMIGA FUNKTION
```

1.128 pexec.library/pOS_GetObjectRootAdr()

```
PROTOTYP
      APTR res = pOS_GetObjectRootAdr
        (
```

```
          pOS_ExecBase *execbase,
          const pOS_Class *class,
          APTR object
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)
      object (_R_A1)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH

AMIGA FUNKTION
```

1.129 pexec.library/pOS_CreateClassGrp()

```
PROTOTYP
      pOS_NClass *res = pOS_CreateClassGrp
        (
          pOS_ExecBase *execbase,
          pOS_ClassGrp *group,
          const CHAR *name,
          const pOS_NClass *basisclass,
          const pOS_Class *newclass
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      group (_R_A0)
      name (_R_A1)
      basisclass (_R_A2)
      newclass (_R_A3)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH
      pOS_DeleteClassGrp()

AMIGA FUNKTION
```

1.130 pexec.library/pOS_DeleteClassGrp()

```
PROTOTYP
      VOID pOS_DeleteClassGrp
        (
          pOS_ExecBase *execbase,
          pOS_ClassGrp *group,
          pOS_NClass *class
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      group (_R_A0)
      class (_R_A1)

SIEHE AUCH
      pOS_CreateClassGrp()

AMIGA FUNKTION
```

1.131 pexec.library/pOS_CreateClass()

```
PROTOTYP
      __ARID__ pOS_NClass *res = pOS_CreateClass
        (
          pOS_ExecBase *execbase,
          _R_A0 const CHAR *classname,
          _R_A1 const CHAR *superclassname,
          _R_A2 const pOS_NClass *superclass,
          _R_A3 APTR displayfunc,
          _R_D0 size_t objectsize,
          _R_D1 ULONG superclassversion
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      classname (_R_A0)
      superclassname (_R_A1)
      superclass (_R_A2)
      displayfunc (_R_A3)
      objectsize (_R_D0)
      superclassversion (_R_D1)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH
      pOS_DeleteClass()
```

```
AMIGA FUNKTION
```

1.132 pexec.library/pOS_DeleteClass()

```
PROTOTYP
      VOID pOS_DeleteClass
        (
          pOS_ExecBase *execbase,
          __ARID__ pOS_NClass *class
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      class (_R_A0)

SIEHE AUCH
      pOS_CreateClass()

AMIGA FUNKTION
```

1.133 pexec.library/pOS_DebugClassI()

```
PROTOTYP
      BOOL res = pOS_DebugClassI
        (
          pOS_ExecBase *execbase,
          const VOID *object,
          const CHAR *classname,
          const pOS_NClass *class,
          BOOL print
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      object (_R_A0)
      classname (_R_A1)
      class (_R_A2)
      print (_R_D0)


ERGEBNIS
      res (_R_D0)
```

```
SIEHE AUCH
      pOS_DebugClassAbs()

AMIGA FUNKTION
```

1.134 pexec.library/pOS_DebugClassAbs()

```
PROTOTYP
      BOOL res = pOS_DebugClassAbs
        (
          pOS_ExecBase *execbase,
          const VOID *object,
          const pOS_Class *masterclass,
          const CHAR *classname,
          const pOS_NClass *class,
          BOOL print
        );

FUNKTION

PARAMETER
      execbase (_R_LB)
        Zeiger auf pExec-Library
      object (_R_A0)
      masterclass (_R_A1)
      classname (_R_A2)
      class (_R_A3)
      print (_R_D0)

ERGEBNIS
      res (_R_D0)

SIEHE AUCH
      pOS_DebugClassI()

AMIGA FUNKTION
```