



PERQ Systems
Corporation

**ACCENT
LANGUAGES MANUAL
Volume 2**

December 7, 1984

This manual is for use with Accent Release S6.

Copyright © 1984 PERQ Systems Corporation
2600 Liberty Avenue
P. O. box 2600
Pittsburgh, PA 15230
(412) 355-0900

Accent is a trademark of Carnegie-Mellon University.

Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its Spice Project.

This document is not to be reproduced in any form or transmitted in whole or in part without the prior written authorization of PERQ Systems Corporation.

The information in this document is subject to change without notice and should not be construed as a commitment by PERQ Systems Corporation. The company assumes no responsibility for any errors that may appear in this document.

PERQ Systems Corporation will make every effort to keep customers apprised of all documentation changes as quickly as possible. The Reader's Comments card is distributed with this document to request users' critical evaluation to assist us in preparing future documentation.

PERQ, PERQ2, LINQ, and Qnix are trademarks of PERQ Systems Corporation.

ACCENT LANGUAGES MANUAL

TABLE OF CONTENTS

VOLUME 1

Preface

Pascal:

PERQ / Accent Pascal Extensions

Pascal Library

PasMac: A Pascal Macro-Processor

VOLUME 2

Pascal (con'd)

Module Index

Kraut: PERQ Pascal Remote

Symbolic Debugger

Matchmaker: The Accent Remote Procedure

Call Language

C:

C System Interfaces

PERQ C Programming

Index

MODULE INDEX

December 7, 1984

Copyright © 1984 PERQ Systems Corporation
2600 Liberty Avenue
P. O. Box 2600
Pittsburgh, PA 15230
(412) 355-0900

Accent is a trademark of Carnegie-Mellon University.

Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its Spice Project.

This document is not to be reproduced in any form or transmitted in whole or in part without the prior written authorization of PERQ Systems Corporation.

The information in this document is subject to change without notice and should not be construed as a commitment by PERQ Systems Corporation. The company assumes no responsibility for any errors that may appear in this document.

PERQ Systems Corporation will make every effort to keep customers apprised of all documentation changes as quickly as possible. The Reader's Comments card is distributed with this document to request users' critical evaluation to assist us in preparing future documentation.

PERQ, PERQ2, LINQ, and Qnix are trademarks of PERQ Systems Corporation.

Table of Contents

	<u>Page</u>
<u>1. Introduction</u>	<u>MD-1</u>
<u>2. Modules</u>	<u>MD-3</u>
<u>3. Definitions</u>	<u>MD-7</u>

1. Introduction

This document contains an index of the public modules in the Pascal library (LibPascal). Chapter 2 lists the modules and the files in which they are contained. Chapter 3 lists all of the definitions that are exported from the public modules.

2. Modules

<u>MODULE</u>	<u>FILE</u>
AccCall	accall.pas
AccentType	accenttype.pas
AccInt	accentuser.pas
ALoad	aload.pas
Auth	authuser.pas
AuthDefs	authdefs.pas
BootInfo	bootinfo.pas
CFileDefs	cfiledefs.pas
Cload	cload.pas
Clock	clock.pas
CommandDefs	commanddefs.pas
CommandParse	commandparse.pas
Configuration	configuration.pas
ControlStore	controlstore.pas
CoreLoad	coreload.pas
DiskUtils	diskutils.pas
Dynamic	dynamic.pas
EnvMgr	envmgruser.pas
EnvMgrDefs	envmgrdefs.pas
EtherTypes	etheratypes.pas
EtherUser	etheruser.pas
Except	except.pas
ExtraCmdParse	extracmdparse.pas
IFileDefs	ifiledefs.pas
IO	iouser.pas
IODefs	iodefs.pas
IPCRecordIO	ipcrecordio.pas
KeyTran	keytran.pas
KeyTranDefs	keytrandefs.pas

ModGetEvent	modgetevent.pas
MsgN	msgnuser.pas
NameErrors	nameerrors.pas
Net10MB	net10mbuser.pas
Net10MBDefs	net10mbdefs.pas
NoEchoInput	noechoinput.pas
NSTypes	nstypes.pas
OldTimeStamp	oldtimestamp.pas
PascalInit	pascalinit.pas
PasLong	paslong.pas
PasReal	pasreal.pas
PathName	pathname.pas
PMatch	pmatch.pas
ProcMgr	procmgruser.pas
ProcMgrDefs	procmgrdefs.pas
QMapDefs	qmapdefs.pas
Reader	reader.pas
RealFunctions	realfunctions.pas
RunDefs	rundefs.pas
SaltError	salterror.pas
SaphEmrExceptions	saphemrexceptions.pas
SaphEmrServer	saphemrserver.pas
Sapph	sapphuser.pas
SapphDefs	sapphdefs.pas
SapphFileDefs	sapphfiledefs.pas
SapphLoadFile	sapphloadfile.pas
SegDefs	segdefs.pas
Sesame	sesameuser.pas
SesameDefs	sesamedefs.pas
SesDisk	sesdiskuser.pas
SesDiskDefs	sesdiskdefs.pas
Spawn	spawn.pas
SpawnInitFlags	spawninitflags.pas
Spice_String	spice_string.pas
Stream	stream.pas

SymDefs	symdefs.pas
SysType	systype.pas
Time	timeuser.pas
TimeDefs	timedefs.pas
TS	tsuser.pas
TSDefs	tsdefs.pas
ViewKern	viewkern.pas
ViewPt	viewptuser.pas
WindowUtils	windowutils.pas
Writer	writer.pas

3. Definitions

The definitions that are exported from the public modules in the Pascal library are listed alphabetically below. The module is shown in brackets after the definition name (the file in which each module is found is shown in Chapter 1). Following the module name is information to serve as a quick reference when you are using the definition.

A_String [*Net10MBDefs*] type = string[81]
Abort [*Except*] exception(Message: String)
AbsoluteDef [*CFileDefs*] const = 13
AbsoluteLocalDef [*CFileDefs*] const = 14
Accent_Version [*AccInt*] function(ServPort: port; Var
 AccVersion: DevPartString): GeneralReturn
AccentError [*Dynamic*] exception(R: GeneralReturn; M: String)
AccentVersion [*Dynamic*] const = True
AccErr [*AccentType*] const = 100
Access_Rights [*SesDiskDefs*] type = integer
Access_Type [*IFileDefs*] type = (Read_Access, Write_Access,
 Owner_Access)
Acknowledge [*IO*] exception(TransactionID: long)
AddExtension [*PathName*] procedure(var fileName: Path_Name;
 Extension: String)
AddrToBlkNum [*DiskUtils*] function(Addr: DiskAddress);
 DiskBlockNumber
AddSearchWord [*CommandParse*] procedure(table:
 pWord_Search_Table; WordKey: integer; WordString:
 Cmnd_String)

AddToKeyTable [*KeyTran*] function(Key1: RawIn; Key2: TranOut; tab: pKeyTab; var conflictDepth: Integer): GeneralReturn

Adjust [*Spice_String*] procedure(var Str: PString; Len: Integer)

All_Read_Access [*SesDiskDefs*] const = #100

All_Write_Access [*SesDiskDefs*] const = #200

AllocatePort [*AccInt*] function(ServPort: port; var LocalPort: Port; BackLog: Integer): GeneralReturn

AllocCommandNode [*CommandParse*] function(WordClass: Word_Type; WordString: Cmnd_String): pCommand_Word_List

ALLPORTS [*AccentType*] const = -1

ALLPTS [*AccentType*] const = 1

ALoadError [*ALoad*] exception(s: string_255)

AlwaysEof [*CommandParse*] procedure(var ChPool: pCharacter_Pool; var PoolLength: Char_Pool_Index)

APath_Name [*SesameDefs*] type = string[Path_Name_Size]

AppendChar [*Spice_String*] procedure(var Str: PString; c: Char)

AppendString [*Spice_String*] procedure(var Str1: PString; Str2: PString)

ArcCos [*RealFunctions*] function(X: Real): Real

ArcCosLarge [*RealFunctions*] exception(X: Real)

ArcSin [*RealFunctions*] function(X: Real): Real

ArcSinLarge [*RealFunctions*] exception(X: Real)

ArcTan [*RealFunctions*] function(X: Real): Real

ArcTan2 [*RealFunctions*] function(Y, X: Real): Real

ArcTan2Zero [*RealFunctions*] exception(Y, X: Real)

Arguments [*AccentType*] type = record ReturnValue:
 GeneralReturn; case integer of 1: (Msg: ptrMsg; MaxWait: long; Option: integer; PtOption: integer); 2: (Ports: ptrPortBitArray; MsgType: long); 3: (SrcAddr: VirtualAddress; DstAddr: VirtualAddress; NumWords: long; Delete: boolean; Create: boolean; Mask: long; DontShare: boolean); 4: (NumCmnds: integer; GPIBBuffer: GPBuffer); 5: (NormalOrEmergency: boolean);

EnableOrDisable: boolean); 6: (SleepID: long); 7: (RectPort: Port; X1: integer; Y1: integer; X2: integer; Y2: integer; Kind: integer); 8: (SrcRect: Port; DstRect: Port; Action: integer; Height: integer; Width: integer; SrcX: integer; SrcY: integer; DstX: integer; DstY: integer); 9: (Rect: Port; FontRect: Port; Funct: integer; FirstX: integer; FirstY: integer; MaxX: integer; FirstChar: integer; MaxChar: integer; StrPtr: Pointer; Rslt: integer); 10: (LockDoLock: boolean; LockPortPtr: ptrLPortArray; LockPortCnt: long); 11: (MsgWType: long; MsgWPortPtr: ptrLPortArray; MsgWPortCnt: long) end

ARunLoad [*ALoad*] procedure(RunFileName: Path_Name; p: pointer; filesize: long; hiskport: port; LoadDebug: boolean)

ASKUSER [*SapphDefs*] const = -32005

ASTInconsistency [*AccentType*] const = AccErr+46

ASTRecord [*BootInfo*] type = integer

AsyncData [*IO*] exception(SequenceNumber: long; DataBuf: pDataBuf; DataBuf_Cnt: long; Status: IOStatusBlk)

ASyncIO [*IO*] procedure(IOPort: ServerIOPort; Command: IOCommand; CmdBlk: pCmdBlk; CmdBlk_Cnt: long; DataBuf: pDataBuf; DataBuf_Cnt: long; DataTransferCnt: long; TimeOut_Arg: long; NotifyOnCompletion: boolean)

Attention [*IO*] exception(AtnCause: long)

Auth_Error_Base [*AuthDefs*] const = 5000

Auth_Var [*AuthDefs*] type = String[Auth_Var_Size]

Auth_Var_Size [*AuthDefs*] const = 30

AuthorizationServerGone [*SesDiskDefs*] const = SesDisk_Error_Base+3

AuthPortIncorrect [*AuthDefs*] const = Auth_Error_Base+3

AvailableVM [*AccInt*] function(ServPort: port; var NumBytes: Long): GeneralReturn

BackLogValue [*AccentType*] type = 0..MAXBACKLOG

BadAlignment [*Dynamic*] exception

BadBase [*Stream*] exception(FileName: SName; Base: Integer)

BadCreateMask [*AccentType*] const = AccErr+52

BadDateTime [*Time*] exception
BadExit [*Except*] exception
BadFile [*IFileDefs*] const = 18
BadHeap [*Dynamic*] exception(H: HeapNumber)
BadHeapNumber [*Dynamic*] const = 0
BadIdTable [*Stream*] exception(FileName: SName)
BadIPCName [*AccentType*] const = AccErr+19
BadKernelMsg [*AccentType*] const = AccErr+27
BadMsg [*AccentType*] const = AccErr+22
BADMSGID [*AccentType*] const = 1
BadMsgType [*AccentType*] const = AccErr+18
BadName [*SesameDefs*] const = Sesame_Error_Base+4
BadPartitionName [*SesDiskDefs*] const =
 SesDisk_Error_Base+1
BadPartitionType [*SesDiskDefs*] const = SesDisk_Error_Base+2
BadPatterns [*PMatch*] exception
BadPointer [*Dynamic*] exception
BadPriority [*AccentType*] const = AccErr+30
BadRectangle [*AccentType*] const = AccErr+53
BADREPLY [*AccentType*] const = 3
BadRights [*AccentType*] const = AccErr+8
BadSearchlistSyntax [*EnvMgrDefs*] const = Env_Error_Base+3
BadSegment [*AccentType*] const = AccErr+34
BadSegType [*AccentType*] const = AccErr+33
BadTrap [*AccentType*] const = AccErr+31
BadVPTable [*AccentType*] const = AccErr+39
BadWildName [*SesameDefs*] const = Sesame_Error_Base+7
BaseType [*SymDefs*] type = (T_Unknown, T_Char, T_Boolean,
 T_Integer, T_Enumerated, T_Real, T_Long, T_Pointer,
 T_Var, T_Routine, T_File, T_String, T_Set, T_Record,
 T_Array, T_Misc)
BigByteSize [*IODefs*] const = 2048
BigLongSize [*IODefs*] const = 512
BigWordSize [*IODefs*] const = 1024
BIRRecord [*BootInfo*] type = packed record case integer of 1:

(IntBlk: array[0..255] of integer); 2: (OvlTable:
array[0..11] of VirtualAddress; VP: VirtualAddress; PV:
VirtualAddress; PVList: VirtualAddress; Sector:
VirtualAddress; PCB: VirtualAddress; AST:
VirtualAddress; AccentQueue: VirtualAddress; AccentFont:
VirtualAddress; AccentCursor: VirtualAddress;
AccentScreen: VirtualAddress; ScreenSize: integer; FreeVP:
integer; FreeAST: integer; SchedProc: integer; InitProc:
integer; BootChar: integer; NumProc: integer; StackSize:
integer; GlobalSize: integer; NumSVReg: integer;
TrapCode: integer; TrapArgs: VirtualAddress; MemBoard:
integer; AccentStdCursor: VirtualAddress; AccentRoTemp:
VirtualAddress; DefaultPartitionName: String[19];
IgnoreRunFile: Boolean; MachineInfo: MachineInfoRec;
Filler: array[0..49-WordSize(AstRecord)] of integer;
FirstAst: ASTRecord; EtherIOArea: VirtualAddress;
UserPtr: VirtualAddress; SVContext: record SV_CS:
integer; SV_GP: integer; SV_LP: integer; SV_LocalSize:
integer; SV_TrapCount: integer; SV_FirstRN: integer;
SV_PC_Vector: array[0..121] of integer end) end

Bit1 [*AccentType*] type = 0..1

Bit10 [*AccentType*] type = 0..1023

Bit11 [*AccentType*] type = 0..2047

Bit12 [*AccentType*] type = 0..4095

Bit13 [*AccentType*] type = 0..8191

Bit14 [*AccentType*] type = 0..16383

Bit15 [*AccentType*] type = 0..32767

Bit16 [*AccentType*] type = integer

Bit2 [*AccentType*] type = 0..3

Bit3 [*AccentType*] type = 0..7

Bit32 [*AccentType*] type = packed record case integer of 1:
(DblWord: array[0..1] of integer); 3: (Bit32Ptr: pBit32); 4:
(AnyPtr: pointer); 5: (Byte: packed array[0..3] of Bit8); 6:
(PageOffset: Bit8; LswPage: Bit8; MswPage: Bit16); 7:
(Lng: Long); 8: (Blk: integer; Index: Bit12; Imag: Bit4); 13:

(Field4: Bit8; Field3: Bit8; Field2: Bit8; Field1: Bit7;
Field0: Bit1); 14: (Word0: Bit16; Word1: Bit16); 15:
(Byte0: Bit8; Byte1: Bit8; Byte2: Bit8; Byte3: Bit8); end

Bit4 [*AccentType*] type = 0..15

Bit5 [*AccentType*] type = 0..31

Bit6 [*AccentType*] type = 0..63

Bit64 [*AccentType*] type = record lsw: long; msw: long; end

Bit7 [*AccentType*] type = 0..127

Bit8 [*AccentType*] type = 0..255

Bit9 [*AccentType*] type = 0..511

BitTable [*Spice_String*] type = set of Char

BlankStarString [*SaltError*] const = ''

BlkNumToAddr [*DiskUtils*] function(Disk: Integer; Blk:
DiskBlockNumber): DiskAddress

BlockNumber [*IFileDefs*] type = -1..32767

BootBlockLocation [*BootInfo*] const = #20000000000

BootDisk [*DiskUtils*] const = 0

BorderOverhead [*SapphDefs*] const = 5

BOTTOM [*SapphDefs*] const = 32000

BreakKind [*Spice_String*] type = set of BreakType

BreakPointTrap [*AccentType*] const = AccErr+45

BreakRecord [*Spice_String*] type = record Breakers: set of Char;
Omittors: set of Char; Flags: BreakKind end

BreakTable [*Spice_String*] type = ^BreakRecord

BreakType [*Spice_String*] type = (Append, Retain, Skip,
FoldUp, FoldDown, Inclusive, Exclusive)

BusyRectangle [*AccentType*] const = AccErr+57

ByteSizeOfAbsoluteLocalSymbol [*CFileDefs*] function(var sym:
string): long

ByteSizeOfAbsoluteSymbol [*CFileDefs*] function(var sym:
string): long

ByteSizeOfLibrarySymbol [*CFileDefs*] function(var libname:
string): long

CantFork [*AccentType*] const = AccErr+29

Cat3 [*Spice_String*] function(Str1, Str2, Str3: pstring): pstring

Cat4 [*Spice_String*] function(Str1, Str2, Str3, Str4: pstring);
 pstring
Cat5 [*Spice_String*] function(Str1, Str2, Str3, Str4, Str5: pstring);
 pstring
Cat6 [*Spice_String*] function(Str1, Str2, Str3, Str4, Str5, Str6:
 pstring); pstring
cChCmd [*KeyTranDefs*] const = 0
cdBlueDown [*SapphDefs*] const = 258
cdBlueUp [*SapphDefs*] const = 262
cdDiffPosResponse [*SapphDefs*] const = 267
cdGreenDown [*SapphDefs*] const = 259
cdGreenUp [*SapphDefs*] const = 263
cdListener [*SapphDefs*] const = 269
cdNoEvent [*SapphDefs*] const = 268
cdPosResponse [*SapphDefs*] const = 266
cdRegionExit [*SapphDefs*] const = 264
cdTimeout [*SapphDefs*] const = 265
cdWhiteDown [*SapphDefs*] const = 257
cdWhiteUp [*SapphDefs*] const = 261
cdYellowDown [*SapphDefs*] const = 256
cdYellowUp [*SapphDefs*] const = 260
CF_IOBoard [*Configuration*] function: Cf_IOBoardType
Cf_IOBoardType [*Configuration*] type = (Cf_CIO, Cf_EIO)
Cf_Monitor [*Configuration*] function: Cf_MonitorType
Cf_MonitorType [*Configuration*] type = (Cf_Landscape,
 Cf_Portrait)
Cf_Network [*Configuration*] function: Cf_NetworkType
Cf_NetworkType [*Configuration*] type = (Cf_CMUNet,
 Cf_10MBitNet)
Cf_OldZ80 [*Configuration*] function: Boolean
CFileVersion [*CFileDefs*] const = -4
ChainHead [*CFileDefs*] const = 12
ChangeExtensions [*PathName*] procedure(var Name:
 Path_Name; EList: Extension_List; NewExt: string)
ChangeKeyTable [*KeyTran*] function(OldKey: RawIn; Key1:

RawIn; Key2: TranOut; tab: pKeyTab): GeneralReturn
ChangeUserParams [*Auth*] function(ServPort: Port; UserName:
Auth_Var; CurrentPassword: Auth_Var; ChangePassword:
Boolean; NewPassword: Auth_Var; NewProfile:
APath_Name; NewShell: APath_Name): GeneralReturn
Char_Pool_Index [*CommandDefs*] type = long
Character_Pool [*CommandDefs*] type = packed array[0..0] of
char
ChArray [*Stream*] type = packed array[1..1] of char
Check_Type [*AuthDefs*] type = (Check_Login, Check_User)
CheckHeap [*Dynamic*] procedure(S: HeapNumber)
CheckIn [*MsgN*] function(ServPort: Port; PortsName: string;
Signature: Port; PortsID: Port): GeneralReturn
CheckOut [*MsgN*] function(ServPort: Port; PortsName: string;
Signature: Port): GeneralReturn
CheckUser [*Auth*] function(ServPort: Port; UserName: Auth_Var;
PassWord: Auth_Var; var UserRec: UserRecord):
GeneralReturn
CImpInfo [*SegDefs*] type = record case boolean of true:
(ModuleName: SNArray; FileName: FNString); false: (Ary:
array[0..0] of integer) end
CLoadNotCFile [*Cload*] const = -1
CLoadProcess [*Cload*] function(FileName: APath_Name; var
FileInMem: pointer; var FileSize: long; Proc: Port;
LoadDebug: Boolean): GeneralReturn
CloseIO [*IO*] function(IOPort: ServerIOPort): GeneralReturn
Cmd_EmptyCmdLine [*ExtraCmdParse*] const = -3
Cmd_NotFound [*ExtraCmdParse*] const = WS_NotFound
Cmd_NotInsMaybeSwitches [*ExtraCmdParse*] const = -4
Cmd_NotUnique [*ExtraCmdParse*] const = WS_NotUnique
Cmd_SomeError [*ExtraCmdParse*] const = -5
CmdParse_Error_Base [*CommandDefs*] const = 4200
Cmnd_String [*CommandParse*] type = String[MaxCmndString]
cNoCmd [*KeyTranDefs*] const = 1
command_file_leadin_char [*CommandParse*] const = '

Command_File_List [*CommandParse*] type = RECORD
 cmdFile: Text; isCharDevice: Boolean; next:
 pCommand_File_List; END

Command_Word_List [*CommandParse*] type = packed record
 ptrWordString: pWord_String; DeallocWordString:
 boolean; case WordClass: Word_Type of in_arg, out_arg,
 command_file: (NextArg: pCommand_Word_List);
 switch_arg: (NextSwitch: pCommand_Word_List;
 ValueOfSwitch: pCommand_Word_List;
 CorrespondingArg: pCommand_Word_List); switch_value:
 ()); end

CommandBlock [*CommandDefs*] type = record WordCount:
 long; WordDirIndex: Char_Pool_Index; WordArrayPtr:
 pCharacter_Pool; WordArray_Cnt: Char_Pool_Index; end

comment_leadin_char [*CommandParse*] const = '#'

CommentLen [*SegDefs*] const = 80

CompactIcons [*Sapph*] procedure(ServPort: Window)

CompletePathName [*PathName*] function(var WildPathName:
 Wild_Path_Name; ImplicitSearchList: Env_Var_Name;
 FirstOnly: boolean; var Cursor: integer): long

ComputeProgress [*WindowUtils*] procedure(Current, Max: Long)

Concat [*Spice_String*] function(Str1, Str2: PString): PString

Confirm_NO [*ExtraCmdParse*] const = 2

Confirm_Switches [*ExtraCmdParse*] const = 3

Confirm_YES [*ExtraCmdParse*] const = 1

ConfirmUser [*Auth*] function(ServPort: Port; UserAuthPort: Port;
 var UserID: User_ID; var UserMachineName: Auth_Var):
 GeneralReturn

ConnectionInheritance [*Spawn*] type = (NewOne, Given,
 GivenReg)

ControlChar [*Stream*] type = 0..#37

ConvertPoolToString [*CommandParse*] function(ChPool:
 pCharacter_Pool; FirstChar: Char_Pool_Index;
 StringLength: Char_Pool_Index): Cmnd_String

ConvertStringToPool [*CommandParse*] procedure(CnvStr:

Cmnd_String; var ChPool: pCharacter_Pool; var
PoolLength: Char_Pool_Index)

ConvertToNewVersion [*KeyTran*] function(OldKeyName:
path_name; var pKeyNew: pKeyTab): GeneralReturn

ConvUpper [*Spice_String*] procedure(var Str: PString)

CopyEnvConnection [*EnvMgr*] function(ServPort: Port;
OldConnection: Port; var NewConnection: port):
GeneralReturn

CoreRunLoad [*CoreLoad*] function(FileName: Path_Name; P:
pointer; filesize: long; Process: port; LoadDebug: boolean):
GeneralReturn

Cos [*RealFunctions*] function(X: Real): Real

CosH [*RealFunctions*] function(x: real): real

CosHLarge [*RealFunctions*] exception(X: Real)

CosLarge [*RealFunctions*] exception(X: Real)

CoTan [*RealFunctions*] function(X: Real): Real

CoveredRectangle [*AccentType*] const = AccErr+56

CreateCursorSet [*ViewPt*] function(ServPort: Viewport; Offsets:
OffsetPairArray; pCursorData: pPatternMapArray;
pCursorData_Cnt: long): CursorSet

CreateHeap [*Dynamic*] function: HeapNumber

CreateProcess [*AccInt*] function(ServPort: port; var
HisKernelPort: port; var HisDataPort: port): GeneralReturn

CreateRectangle [*AccInt*] function(ServPort: port; RectPort: port;
BaseAddr: VirtualAddress; ScanWidth: Integer; BaseX:
Integer; BaseY: Integer; MaxX: Integer; MaxY: Integer;
IsFont: Boolean): GeneralReturn

CreateSegment [*AccInt*] function(ServPort: port; ImagSegPort:
port; SegmentKind: SpiceSegKind; InitialSize: Integer;
MaxSize: Integer; Stable: Boolean; var Segment: SegID):
GeneralReturn

CreateWindow [*Sapph*] function(ServPort: Window;
fixedPosition: boolean; var leftx: integer; var topy: integer;
fixedSize: boolean; var width: integer; var height: integer;
hasTitle: boolean; hasborder: boolean; title: TitStr; var

progName: ProgStr; hasIcon: boolean; var vp: Viewport);
Window

CRLFConvention [*SysType*] const = false

CursorArrayRec [*SapphFileDefs*] type = record firstBlock:
 Packed Record numCursors: Integer; filler: Integer; offsets:
 Array[0..126] of record x: Integer; y: Integer; End; End;
 cursors: array[0..0] of PatternMap; end

CursorFunction [*SapphDefs*] type = (cfScreenOff, cfBroken,
 cfOR, cfXOR, cfCursorOff)

CursorOp [*TSDefs*] type = 0..6

CursorSet [*SapphDefs*] type = Port

CVD [*Spice_String*] function(Str: PString): integer

CVH [*Spice_String*] function(Str: PString): integer

CVHS [*Spice_String*] function(I: integer): PString

CVHSS [*Spice_String*] function(I: integer; W: integer): PString

CvInt [*Spice_String*] function(Str: PString; R: integer): integer

CvL [*Spice_String*] function(Str: PString; Radix: integer): long

CvLS [*Spice_String*] function(I: long; W: integer; Radix: integer;
 Fill: Pstring): Pstring

CVN [*Spice_String*] function(I: integer; W: integer; B: integer;
 Fill: Pstring): Pstring

CVO [*Spice_String*] function(Str: PString): integer

CVOS [*Spice_String*] function(I: integer): Pstring

CVOSS [*Spice_String*] function(I: integer; W: integer): Pstring

CVS [*Spice_String*] function(I: integer): Pstring

CVSS [*Spice_String*] function(I: integer; W: integer): Pstring

CvUp [*Spice_String*] function(Str: PString): PString

Data2State [*CFileDefs*] const = 2

Data_Format [*SesameDefs*] type = long

DATAPORT [*AccentType*] const = 2

DataState [*CFileDefs*] const = 1

Date_Fields [*TimeDefs*] type = packed record Year: integer;
 Month: 1..12; Day: 1..31; Weekday: 0..6; end

DateString [*ALoad*] function(date: Internal_Time): String

DBLINDSIZE [*JFileDefs*] const = 2

DeallocatePort [*AccInt*] function(ServPort: port; LocalPort: Port;
Reason: Long): GeneralReturn

DeAllocIconVP [*Sapph*] procedure(ServPort: Window)

DebugMessage [*ProcMgrDefs*] type = record Head: Msg; tKPort:
TypeType; KPort: Port; tArg1: TypeType; Arg1: Long;
tArg2: TypeType; Arg2: Long; end

DEFAULTBACKLOG [*AccentType*] const = 0

DefaultInputName [*Stream*] var: STRING[255]

DefaultOutputName [*Stream*] var: STRING[255]

DEFAULTPTS [*AccentType*] const = 0

DefinedGlobal [*CFileDefs*] const = InitializedSymbol

DefinedLocal [*CFileDefs*] const = LocalLabel

DefineFullSize [*Sapph*] procedure(ServPort: Window; exceptW:
Window)

DeleteChars [*Spice_String*] procedure(Var Str: PString; Index,
Size: Integer)

DeleteFromKeyTable [*KeyTran*] function(Key: RawIn; tab:
pKeyTab): GeneralReturn

DeleteOp [*TSDefs*] type = 0..2

DeleteRegion [*ViewPt*] procedure(ServPort: Viewport;
regionNum: integer)

DeleteSearchWord [*CommandParse*] procedure(table:
pWord_Search_Table; WordString: Cmnd_String)

DeleteWindow [*Sapph*] procedure(ServPort: Window)

DensityType [*IODefs*] type = (SingleDensity, DoubleDensity)

Deposit [*AccInt*] function(ServPort: port; RegOrStack: Boolean;
Index: Integer; Value: Integer): GeneralReturn

DestroyChPool [*CommandParse*] procedure(var ChPool:
pCharacter_Pool; var PoolLength: Char_Pool_Index)

DestroyCommandList [*CommandParse*] procedure(var argList:
pCommand_Word_List)

DestroyCommandParse [*CommandParse*] procedure

DestroyHeap [*Dynamic*] procedure(S: HeapNumber)

DestroyKeyTable [*KeyTran*] function(Tab: pKeyTab):
GeneralReturn

DestroyRectangle [*AccInt*] function(ServPort: port; RectPort:
port): GeneralReturn

DestroyRegions [*ViewPt*] procedure(ServPort: Viewport)

DestroySearchTable [*CommandParse*] procedure(var table:
pWord_Search_Table)

DestroySegment [*AccInt*] function(ServPort: port; Segment:
SegID): GeneralReturn

DestroyViewport [*ViewPt*] procedure(ServPort: Viewport)

DestroyVPCursors [*ViewPt*] procedure(ServPort: CursorSet)

DeviceRecord [*DiskUtils*] type = packed record InfoBlk:
DiskAddress; InUse: boolean; RootPartition: PartString end

DevPartString [*AccentType*] type = string[MAXDPCHARS]

DForm_16_Bit [*SesameDefs*] const = 16

DForm_32_Bit [*SesameDefs*] const = 32

DForm_36_Bit [*SesameDefs*] const = 36

DForm_8_Bit [*SesameDefs*] const = 8

DForm_CRLF_Text [*SesameDefs*] const = #413

DForm_LF_Text [*SesameDefs*] const = #410

DForm_Press [*SesameDefs*] const = #1000

DForm_Unspecified [*SesameDefs*] const = 0

DIBAddress [*DiskUtils*] const = #30000000000

Dir_Separator [*SesameDefs*] const = '/'

DirBlock [*IFileDefs*] type = packed record case integer of 1:
(FSData: FSDataEntry; Direct: array[0..DIRECTSIZE-1] of
DiskAddr; Indirect: array[0..INDSIZE-1] of DiskAddr;
DblInd: array[0..DBLINDSIZE-1] of DiskAddr; SegKind:
SpiceSegKind; NumBlksInUse: integer; LastBlk: integer;
LastAddr: DiskAddr; LastNegBlk: integer; LastNegAddr:
DiskAddr); 2: (DFSDData: FSDataEntry; Addr: pDirBlock;
Segment: SegID; PPort: Port; NumPages: integer;
HighBlock: integer; forptr: pDirBlock; backptr: pDirBlock;
NoSegment: boolean; Attached: boolean); 3: (Entry:
array[0..FILESPERDIRBLK-1] of DirEntry) end

DirectIO [*AccInt*] function(ServPort: port; var CmdBlk:
DirectIOArgs; var DataHdr: Header; var Data: DiskBuffer):

GeneralReturn

DirectIOArgs [*AccentType*] type = Record IOStatus: Integer;
UnitNumber: Integer; PhysAddress: Long; Command:
DirIOCommands; end

DirectoryNotEmpty [*SesameDefs*] const =
Sesame_Error_Base+3

DirectoryNotFound [*SesameDefs*] const =
Sesame_Error_Base+2

DIRECTSIZE [*IFileDefs*] const = 64

DirEntry [*IFileDefs*] type = packed record InUse: boolean;
Deleted: boolean; Archived: boolean; EType: 0..#17777;
ID: EntryVal; Filename: POSSimpleName end

DirFile [*IFileDefs*] const = 3

DirIOCommands [*AccentType*] type = (DirIOInit, DirIORRead,
DirIOWrite, DirIORReadCheck, DirIOWriteCheck,
DirIOTrackRead, DirIOTrackWrite, DirIOPParamRead,
DirIOBootRead, DirIOBootWrite, DirIOCclose)

DisablePrivils [*PascalInit*] function(Proc: PORT): GeneralReturn

DiskAddr [*AccentType*] type = packed record case integer of 1:
(lng: long); 2: (byte: packed array[0..3] of Bit8) end

DiskAddress [*DiskUtils*] type = Long

DISKBITS [*DiskUtils*] const = #300000000000

DiskBlock [*DiskUtils*] type = packed record case integer of 1:
(Addr: array[0..(PAGEWORDSIZE div 2)-1] of Long); 2:
(IntData: array[0..PAGEWORDSIZE-1] of integer); 3:
(ByteData: packed array[0..PAGEBYTESIZE-1] of Bit8);
4: (FSData: FSDataEntry; Direct: array[0..DIRECTSIZE-1] of
DiskAddress; Indirect: array[0..INDSIZE-1] of
DiskAddress; DblInd: array[0..DBLINDSIZE-1] of
DiskAddress; SegKind: SpiceSegKind; NumBlksInUse:
integer; LastBlk: integer; LastAddr: DiskAddress;
LastNegBlk: integer; LastNegAddr: DiskAddress); 5:
(FreeHead: DiskAddress; FreeTail: DiskAddress; NumFree:
DiskAddress; RootDirID: DiskAddress; BadSegID:
DiskAddress; BootTable: array[0..25] of PhysicalAddress;

InterpTable: array[0..25] of PhysicalAddress; PartName:
packed array[1..8] of char; PartStart: DiskAddress; PartEnd:
DiskAddress; SubParts: array[0..63] of DiskAddress;
PartRoot: DiskAddress; PartKind: PartitionType;
PartDevice: DiskType); 7: (Filling: Packed Array[0..9] of
integer; IsPartRelative: Boolean); 8: (intrec: intblock); 9:
(Params: DiskParams); 10: (Data: Diskbuffer) end

DiskBlockNumber [*DiskUtils*] type = long

DiskBuffer [*AccentType*] type = Packed
Array[0..DISKBUFSIZE*2-1] of Bit8

DISKBUFSIZE [*AccentType*] const = PAGEBYTESIZE div 2

DiskErr [*AccentType*] const = AccErr+32

DiskHeader [*DiskUtils*] type = Packed Record Case Integer of 0:
(SerialNumber: DiskAddress; LogBlock: integer; Filler:
Integer; PrevAdr: DiskAddress; NextAdr: DiskAddress); 1:
(intrec: inthdr) End

DiskInterface [*AccentType*] type = (EIO, CIO, FlopDrives,
MultiBus, Enet)

DiskParams [*AccentType*] type = Record DskBootSize: Integer;
DskSectors: Integer; DskNumHeads: Integer;
DskNumCylinders: Integer; DskSecCyl: Long; Case
HDiskType: DiskType of D5Inch: (WriteCompCyl: Integer;
LandingZone: Integer); D14Inch: (c24MByte: Boolean);
End

DiskType [*AccentType*] type = (DUnused, D5Inch, D14Inch,
D8Inch, DSMD, DFloppy)

DisposeP [*Dynamic*] procedure(var Where: pointer; L: integer)

Distress [*I/O*] exception(TransactionID: long; DataBuf: pDataBuf;
DataBuf_Cnt: long; Status: IOStatusBlk)

DivZero [*Except*] exception

DONTCARE [*SapphDefs*] const = -32004

DONTWAIT [*AccentType*] const = 1

DstryCmdFiles [*CommandParse*] procedure(var inF:
pCommand_File_List)

Dummy [*AccentType*] const = AccErr+0

Dump [*Except*] exception(Message: String)
DynamicVersion [*Dynamic*] const = '1.0'
DynDebug [*Dynamic*] const = false
DynPrint [*Dynamic*] const = false
DynStats [*Dynamic*] procedure(Tr, Dump: boolean)
E10Address [*Net10MBDefs*] type = record High: integer; Mid:
 integer; Low: integer; end
E10BothClear [*Net10MB*] function(ServPort: Port; PacketPort:
 E10Port; Which: E10FilterType): GeneralReturn
E10ByteCount [*Net10MBDefs*] const = 3002
E10ByteData [*Net10MBDefs*] type = packed array[1..1500] of
 0..255
E10BytesInHeader [*Net10MBDefs*] const = 14
E10CRC [*Net10MBDefs*] const = 3005
E10FilterType [*Net10MBDefs*] type = integer
E10GetAdd [*Net10MB*] function(ServPort: Port; var Addr:
 E10Address): GeneralReturn
E10MaxDataBytes [*Net10MBDefs*] const = 1500
E10Message [*Net10MBDefs*] type = record Head: Msg; Body:
 E10MsgArray; end
E10MinDataBytes [*Net10MBDefs*] const = 46
E10MsgArray [*Net10MBDefs*] type = array[1..124] of integer
E10NoFreeStructures [*Net10MB*] exception
E10NoNet [*Net10MBDefs*] const = 3003
E10OK [*Net10MBDefs*] const = 3000
E10Packet [*Net10MBDefs*] type = packed record Dest:
 E10Address; Src: E10Address; PType: E10Type; BData:
 E10ByteData end
E10PktByteSize [*NSTypes*] const = 2*WordSize(E10Packet)
E10Port [*Net10MBDefs*] type = port
E10PortClear [*Net10MB*] function(ServPort: Port; PacketPort:
 E10Port): GeneralReturn
E10Receive [*Net10MB*] exception(ServPort: Port; Buff:
 pE10Packet; NumBytes: long)
E10RecvFailed [*Net10MB*] exception(Why: integer)

E10RetVal [*Net10MBDefs*] type = integer
E10Send [*Net10MB*] function(ServPort: Port; Buff: pE10Packet;
 NumBytes: long): GeneralReturn
E10SendError [*Net10MBDefs*] const = 3004
E10SendFailed [*Net10MB*] exception(Why: integer)
E10SetFilter [*Net10MB*] function(ServPort: Port; PacketPort:
 E10Port; Which: E10FilterType): GeneralReturn
E10Stats [*Net10MB*] procedure(ServPort: Port; var Stats:
 E10StatsRec)
E10StatsRec [*Net10MBDefs*] type = record CRC: long; Collision:
 long; Runt: long; Dribble: long; Drabble: long; Snt: long;
 Rcvd: long; Retry: long; end
E10TimeOut [*Net10MBDefs*] const = 3001
E10Type [*Net10MBDefs*] type = integer
E10TypeClear [*Net10MB*] function(ServPort: Port; Which:
 E10FilterType): GeneralReturn
E10WordsInHeader [*Net10MBDefs*] const = 7
EBadReply [*EtherUser*] exception
ELevel1Abort [*SaphEmrExceptions*] exception
ELevel1Debug [*SaphEmrExceptions*] exception
ELevel2Abort [*SaphEmrExceptions*] exception
ELevel3Abort [*SaphEmrExceptions*] exception
EMERGENCYMSG [*AccentType*] const = 1
EmergMsg [*Except*] exception
EMPort [*PascalInit*] var: port
EnableNotifyExceptions [*ViewPt*] procedure(ServPort: Viewport;
 notifyPort: Port; changed: boolean; exposed: boolean)
EnablePrivs [*PascalInit*] function(Proc: PORT): GeneralReturn
EnableRectangles [*Acclnt*] function(ServPort: port; RectList:
 PtrPortArray; RectList_Cnt: long; Enable: Boolean):
 GeneralReturn
EnableWinListener [*Sapph*] procedure(ServPort: Window;
 EmergPort: Port)
Entry_All [*SesameDefs*] const = 0
Entry_Data [*SesameDefs*] type = record case Entry_Type of

```
Entry_File: (); Entry_Directory: (); Entry_Port: (EDPort:  
Port);#400: (EDBytes: packed array[0..255] of bit8);#401:  
(EDWords: array[0..127] of integer);#402: (EDLongs:  
array[0..63] of long);#403: (EDString: string[255]); end  
Entry_Directory [SesameDefs] const = 2  
Entry_File [SesameDefs] const = 1  
Entry_Foreign [SesDiskDefs] const = 6  
Entry_List [SesameDefs] type = ^Entry_List_Array  
Entry_List_Array [SesameDefs] type = array[0..0] of  
    Entry_List_Record  
Entry_List_Record [SesameDefs] type = record EntryName:  
    Entry_Name; EntryVersion: long; EntryType: Entry_Type;  
    NameStatus: Name_Status; end  
Entry_Name [SesameDefs] type = string[Entry_Name_Size]  
Entry_Name_Size [SesameDefs] const = 80  
Entry_Port [SesameDefs] const = 3  
Entry_RESERVED [SesameDefs] type = 4..#377  
Entry_Type [SesameDefs] type = 0..#77777  
Entry_UserDefined [SesameDefs] type = #400..#77777  
Env_Element [EnvMgrDefs] type = string[Env_Element_Size]  
Env_Element_Array [EnvMgrDefs] type = array[0..0] of  
    Env_Element  
Env_Element_Size [EnvMgrDefs] const = 255  
Env_Error_Base [EnvMgrDefs] const = 1600  
env_quoted_bracket_char [CommandParse] const = ''  
Env_Scan_Array [EnvMgrDefs] type = array[0..0] of  
    Env_Scan_Record  
Env_Scan_List [EnvMgrDefs] type = ^Env_Scan_Array  
Env_Scan_Record [EnvMgrDefs] type = record VarName:  
    Env_Var_Name; VarType: Env_Var_Type; VarScope:  
    Env_Var_Scope; end  
env_var_bracket_char [CommandParse] const = ^^  
Env_Var_Name [EnvMgrDefs] type =  
    string[Env_VarName_Size]  
Env_Var_Scope [EnvMgrDefs] type = (Env_Normal,
```

Env_Local, Env_Global)
Env_Var_Type [*EnvMgrDefs*] type = (Env_String,
Env_SearchList)
Env_Variable [*EnvMgrDefs*] type = ^Env_Element_Array
Env_VarName_Size [*EnvMgrDefs*] const = Entry_Name_Size
EnvCompletePathName [*PathName*] function(EnvConnection:
Port; var WildPathName: Wild_Path_Name;
ImplicitSearchList: Env_Var_Name; FirstOnly: boolean;
var Cursor: integer): long
EnvDisconnect [*EnvMgr*] function(ServPort: Port):
GeneralReturn
EnvFindWildPathnames [*PathName*] function(EnvConnection:
Port; var WildPathName: Path_Name; ImplicitSearchList:
Env_Var_Name; FirstOnly: boolean; NameFlags:
Name_Flags; EntryType: Entry_Type; var FoundInFirst:
boolean; var DirName: APath_Name; var EntryList:
Entry_List; var EntryList_Cnt: long): GeneralReturn
EnvMgr_Version [*EnvMgr*] function(ServPort: Port; var Versn:
string): GeneralReturn
EnvVariableNotFound [*EnvMgrDefs*] const =
Env_Error_Base+1
eofChar [*CommandParse*] const = chr(0)
eolnChar [*CommandParse*] const = chr(#12)
ErAnyError [*CommandDefs*] const = CmdParse_Error_Base+14
ErBadCmd [*CommandDefs*] const = CmdParse_Error_Base+2
ErBadQuote [*CommandDefs*] const = CmdParse_Error_Base+13
ErBadSwitch [*CommandDefs*] const = CmdParse_Error_Base+1
ErCmdNotUnique [*CommandDefs*] const =
CmdParse_Error_Base+8
ErCmdParam [*CommandDefs*] const =
CmdParse_Error_Base+6
EReceive [*AccCall*] function(var xxmsg: Msg; MaxWait: long;
PortOpt: PortOption; Option: ReceiveOption):
GeneralReturn
EReceiveFailed [*EtherUser*] exception(Why: GeneralReturn)

ErElementNotFound [*KeyTranDefs*] const =
 KeyTran_Error_Base+1
ErHashZero [*KeyTranDefs*] const = KeyTran_Error_Base+4
ErIllCharAfter [*CommandDefs*] const =
 CmdParse_Error_Base+12
ErIllegalKeyVersion [*KeyTranDefs*] const =
 KeyTran_Error_Base+2
ErNoCmdParam [*CommandDefs*] const =
 CmdParse_Error_Base+4
ErNoFreeSpaces [*KeyTranDefs*] const = KeyTran_Error_Base+3
ErNoOutFile [*CommandDefs*] const = CmdParse_Error_Base+9
ErNoSwParam [*CommandDefs*] const =
 CmdParse_Error_Base+3
ErNotValidTable [*KeyTranDefs*] const =
 KeyTran_Error_Base+5
ErOneInput [*CommandDefs*] const = CmdParse_Error_Base+10
ErOneOutput [*CommandDefs*] const =
 CmdParse_Error_Base+11
ErrorMsgPMBroadcast [*SaltError*] procedure(GR:
 GeneralReturn; ER_Type: GR_Error_Type; ProgName:
 String; InMsg: PString)
ErSwNotUnique [*CommandDefs*] const =
 CmdParse_Error_Base+7
ErSwParam [*CommandDefs*] const = CmdParse_Error_Base+5
EscKind [*KeyTranDefs*] type = (CodeNormal, CodeSame,
 EscReturn, EscNoop)
ESendFailed [*EtherUser*] exception(Why: GeneralReturn)
EStack [*Except*] exception
EStackTooDeep [*AccentType*] const = AccErr+42
EtherHeader [*EtherTypes*] type = PACKED RECORD Src:
 0..255; Dst: 0..255; Typ: INTEGER; END
EtherIDBase [*EtherTypes*] const = 800
EtherPacket [*EtherTypes*] type = RECORD CASE Integer OF 0:
 (Header: EtherHeader; DataWords:
 ARRAY[0..MaxEtherWords-WordSize(EtherHeader)-1] OF

INTEGER); 1: (Words: ARRAY[0..MaxEtherWords-1] OF
INTEGER); 2: (LongHeader: EtherHeader; LongWords:
ARRAY[0..(MaxEtherWords-WordSize(EtherHeader)) div
2-1] OF LONG); 3: (ConfigHdr: EtherHeader; SkipC:
integer; ConfigBytes: PACKED
ARRAY[0..2*(MaxEtherWords-WordSize(EtherHeader)-1)]
OF Bit8) END

EtherPacketBytes [*EtherTypes*] const =
2*WordSize(MsgEtherPacket)

EtherTyp3ConfigTest [*EtherTypes*] const = #220

EtherTypConfigTest [*EtherTypes*] const = #220

EtherTypEchoMe [*EtherTypes*] const = #700

EtherTypIAmAnEcho [*EtherTypes*] const = #701

EtherTypPup [*EtherTypes*] const = #1000

EventRec [*SapphDefs*] type = record vp: Viewport; x, y: Integer;
region: Integer; code: KeyState; End

EViewPtChanged [*SaphEmrExceptions*] exception(vp: Viewport;
newx, newy, neww, newh, newRank: Integer)

EViewPtExposed [*SaphEmrExceptions*] exception(vp: Viewport;
ra: pRectArray; numRectangles: Long)

Examine [*AccInt*] function(ServPort: port; RegOrStack: Boolean;
Index: Integer; var Value: Integer): GeneralReturn

ExceptVersion [*Except*] const = '3.4'

ExDirFile [*IFileDefs*] const = 4

Exec [*Spawn*] function(VAR ChildKPort: Port; VAR ChildDPort:
Port; ProcessName: APath_Name; HisCommand:
CommandBlock): GeneralReturn

ExerciseParseEngine [*CommandParse*] function(ChPool:
pCharacter_Pool; PoolLength: Char_Pool_Index; procedure
ReadPool(var Pool: pCharacter_Pool; var PLen:
Char_Pool_Index); var inputs: pCommand_Word_List; var
outputs: pCommand_Word_List; var switches:
pCommand_Word_List): GeneralReturn

ExitAllCmdFiles [*CommandParse*] procedure(var inF:
pCommand_File_List)

ExitCmdFile [*CommandParse*] procedure(var inF:
 pCommand_File_List)

ExitGRError [*PascalInit*] exception(GR: GeneralReturn)

ExitProgram [*PascalInit*] exception

Exp [*RealFunctions*] function(X: Real): Real

ExpandPathName [*PathName*] function(var WildPathName:
 Wild_Path_Name; ImplicitSearchList: Env_Var_Name):
 GeneralReturn

ExpandWindow [*Sapph*] procedure(ServPort: Window)

ExpLarge [*RealFunctions*] exception(X: Real)

EXPLICITDEALLOC [*AccentType*] const = 0

ExpSmall [*RealFunctions*] exception(X: Real)

Extension_List [*PathName*] type =
 string[Extension_String_Size]

Extension_String_Size [*PathName*] const = 80

ExtractAllRights [*AccInt*] function(ServPort: port; PortIndex:
 Long; var PortRight: port; var PortType: Integer):
 GeneralReturn

ExtractEvent [*ModGetEvent*] function(repMsg: Pointer; var w:
 ViewPort; var e: EventRec): Boolean

ExtractSimpleName [*PathName*] procedure(Name: Path_Name;
 var StartTerminal: integer; var StartVersion: integer)

Failure [*AccentType*] const = AccErr+24

FEarray [*RunDefs*] type = array[FileIndex] of FileEntry

FHdr_Access_Rights_Offset [*SesDiskDefs*] const = 56

FIBlk [*IFileDefs*] const = -1

File_Data [*SesameDefs*] type = pointer

File_Header [*SesameDefs*] type = packed record
 FileSize: long;
 DataFormat: long; PrintName: Print_Name; Author:
 Group_ID; CreationDate: Internal_Time; AccessID:
 Group_ID; AccessDate: Internal_Time;
 FHDR_RESERVED: array[56..64] of integer; end

FileEntry [*RunDefs*] type = record
 FileType: LinkFileType;
 FileLocation: long; FileBlocks: long; FileName:
 APath_Name; WriteDate: Internal_Time; Version: long;

end

FileIndex [*RunDefs*] type = 0..RMAXFILE

FileKind [*Stream*] type = (BlockStructured, CharacterStructured)

FileLength [*SegDefs*] const = 100

FILESPERDIRBLK [*IFileDefs*] const = 16

FileType [*Stream*] type = packed record Flag: packed record case integer of 0: (CharReady: boolean; FEoln: boolean; FEof: boolean; FNotReset: boolean; FNotOpen: boolean; FNotRewrite: boolean; FExternal: boolean; FBusy: boolean; FKind: FileKind; FLastWasEof: boolean; Funused: 0..63); 1: (skip1: 0..3; ReadError: 0..7); 2: (skip2: 0..15; WriteError: 0..3) end; EolCh, EofCh, EraseCh, NoiseCh: ControlChar; OmitCh: set of ControlChar; FileNum: integer; FIndex: integer; Length: integer; BlockNumber: integer; StreamP: pStreamF; LengthInBlocks: integer; LastBlockLength: integer; SizeInWords: integer; SizeInBits: 0..16; ElsPerWord: 0..16; Element: record case integer of 1: (C: char); 2: (W: array[0..0] of integer) end end

FindExtendedFileName [*PathName*] function(var FileName: Path_Name; ExtensionList: Extension_List; ImplicitSearchList: Env_Var_Name; FirstOnly: boolean): GeneralReturn

FindExtendedPathName [*PathName*] function(var PathName: Path_Name; ExtensionList: Extension_List; ImplicitSearchList: Env_Var_Name; FirstOnly: boolean; var EntryType: Entry_Type; var NameStatus: Name_Status): GeneralReturn

FindFileName [*PathName*] function(var FileName: Path_Name; ImplicitSearchList: Env_Var_Name; FirstOnly: boolean): GeneralReturn

FindPathName [*PathName*] function(var PathName: Path_Name; ImplicitSearchList: Env_Var_Name; FirstOnly: boolean; var EntryType: Entry_Type; var NameStatus: Name_Status): GeneralReturn

FindTypedName [*PathName*] function(var PathName:

Path_Name; ExtensionList: Extension_List;
ImplicitSearchList: Env_Var_Name; FirstOnly: boolean;
var EntryType: Entry_Type; var NameStatus:
Name_Status); GeneralReturn

FindWildPathnames [*PathName*] function(var WildPathName:
Path_Name; ImplicitSearchList: Env_Var_Name;
FirstOnly: boolean; NameFlags: Name_Flags; EntryType:
Entry_Type; var FoundInFirst: boolean; var DirName:
APath_Name; var EntryList: Entry_List; var
EntryList_Cnt: long): GeneralReturn

FinEther [*EtherUser*] procedure

FinishDiskUtils [*DiskUtils*] procedure(Disk: integer)

First_User [*AuthDefs*] const = 1

FirstBlock [*CFileDefs*] type = recordFileVersion: integer;
FileSize: long; FileTimeStamp: Internal_Time;
SymbolAreaSize: long; TextAreaSize: long; DataAreaSize:
long; BSSAreaSize: long; DestAddr: long; StartAddr: long;
MainAddr: long; InitialLocalSize: long; StackBaseAddress:
long; StackSizeInPages: long; end

FirstItemNotDefined [*EnvMgrDefs*] const = Env_Error_Base+5

FIRSTNONRESERVEDPORT [*AccentType*] const = 3

FirstUserIndex [*PascalInit*] const = 6

FiveDeep [*AccentType*] const = AccErr+38

FloppyResultStatus [*IODefs*] type = packed record StatusType:
FloppyResultType; Unused: Bit6; case FloppyResultType of
NoStatus: (); HeadChange, DriveSense, CmdResults: (Unit:
Bit2; Head: Bit1; case FloppyResultType of DriveSense:
(TwoSided: boolean; AtTrack0: boolean; DriveReady:
boolean; WriteProtected: boolean; DriveFault: boolean);
HeadChange, CmdResults: (NotReady: boolean;
EquipFault: boolean; SeekEnd: boolean; IntrCode: (Normal,
Abnormal, InvalidCmd, DriveRdyChange); case
FloppyResultType of HeadChange: (PresentCylinder: Bit8);
CmdResults: (NoAddrMark: boolean; NotWritable: boolean;
NoData: boolean; Unused1: Bit1; Overrun: boolean;

DataError: boolean; Unused2: Bit1; TrackEnd: boolean;
NoDataAddrMark: boolean; BadTrack: boolean; ScanFail:
boolean; ScanHit: boolean; WrongCylinder: boolean;
DataCRCError: boolean; ControlMark: boolean; Unused3:
Bit1; CylinderID: Bit8; HeadID: Bit8; SectorID: Bit8;
SectorSizeCode: Bit8))) end

FloppyResultType [*IODEfs*] type = (NoStatus, HeadChange,
DriveSense, CmdResults)

FlushEvents [*Sapph*] function(ServPort: Window): boolean

FNString [*SegDefs*] type = String[FileLength]

FontCharWidthVector [*ViewPt*] procedure(ServPort: Viewport;
ch: char; var dx: integer; var dy: integer)

FontHeadOverhead [*SapphFileDefs*] const = #404

FontMap [*SapphFileDefs*] type = ^FontMapRec

FontMapRec [*SapphFileDefs*] type = packed record Height:
integer; Base: integer; Index: array[0..#177] of packed
record Offset: 0..767; Line: 0..63; Width: integer; end;
Filler: array[0..1] of integer; Pat: Array[0..0] of integer;
end

FontSize [*ViewPt*] procedure(ServPort: Viewport; var name:
String; var PointSize: integer; var Rotation: integer; var
FaceCode: integer; var maxWidth: integer; var maxHeight:
integer; var xOrigin: integer; var yOrigin: integer; var
fixedWidth: boolean; var fixedHeight: boolean)

FontStringWidthVector [*ViewPt*] procedure(ServPort: Viewport;
str: VPStr255; firstCh: integer; lastch: integer; var dx:
integer; var dy: integer)

FontWordWidth [*SapphFileDefs*] const = 48

ForBootFile [*SpawnInitFlags*] const = FALSE

Fork [*AccInt*] function(ServPort: port; var HisKernelPort: port;
var HisDataPort: port; var Ports: PtrPortArray; var
Ports_Cnt: long): GeneralReturn

ForLibFile [*SpawnInitFlags*] const = NOT ForBootFile

FSDataEntry [*IFileDefs*] type = packed record FileBlocks:
integer; FileBits: 0..4096; FileSparse: Boolean; Unused:

bit2; FileCreateDate: TimeStamp; FileWriteDate:
TimeStamp; FileAccessDate: TimeStamp; FileType: bit2;
FileRights: IntAccessFlags; FileOwner: Bit10;
FileDateFormat: long; Filename: PartialPathName; end

FullLn [*Stream*] function(var F: Text): Boolean

FullWindowState [*Sapph*] procedure(ServPort: Window; var
leftx: integer; var topy: integer; var outerwidth: integer; var
outerHeigh: integer; var rank: integer; var hasBorder:
boolean; var hasTitle: boolean; var isListener: boolean; var
name: ProgStr; var title: TitStr)

FwdCode [*EtherTypes*] const = 2

GeneralReturn [*AccentType*] type = integer

GetAbsoluteDef [*CFileDefs*] procedure(var BytePtr: long; var
ByteAddress: long; var Name: LString)

GetAbsoluteLocalDef [*CFileDefs*] procedure(var BytePtr: long;
var ByteAddress: long; var Name: LString)

getAsmLong [*CFileDefs*] function(var ByteAddr: long): long

GetAsmString [*CFileDefs*] function(var ByteAddr: long): LString

GetB [*Stream*] procedure(var F: Filetype)

GetBreak [*Spice_String*] function: BreakTable

GetC [*Stream*] procedure(var F: Filetype)

GetChainHead [*CFileDefs*] procedure(var BytePtr: long; var
AreaDesg: Bit8; var ByteAreaOff: long)

GetCharacterPool [*ExtraCmdParse*] procedure(prompt:
Cmnd_String; var InputFile: Text; var ChPool:
pCharacter_Pool; var PoolLength: Char_Pool_Index)

GetCmd [*ExtraCmdParse*] function(prompt: Cmnd_String;
SearchTable: pWord_Search_Table; var CmdName:
Cmnd_String; var inF: pCommand_File_List; var inputs:
pCommand_Word_List; var outputs:
pCommand_Word_List; var switches:
pCommand_Word_List; var ErrorGR: GeneralReturn):
integer

GetCodeByte [*CFileDefs*] function(var ByteAddr: long): Bit8

GetCodeWord [*CFileDefs*] function(var ByteAddr: long): integer

GetConfirm [*ExtraCmdParse*] function(prompt: Cmnd_String;
def: integer; var switches: pCommand_Word_List): integer

GetCursor [*ViewPt*] procedure(ServPort: CursorSet; cursIndex:
integer; var pCursorData: Pattern; var xOffset: integer; var
yOffset: integer)

GetCursorSet [*ViewPt*] procedure(ServPort: CursorSet; var
Offsets: OffsetPairArray; var pCursorData:
pPatternMapArray; var pCursorData_Cnt: long)

GetDateTime [*Time*] function(ServPort: port): Internal_Time

GetDiskInfo [*DiskUtils*] procedure(UnitNumber: Integer; Var
NHeads, NCyls, NSectors, Bootsize: Integer; Var DType:
DiskType)

GetDiskPartitions [*AccInt*] function(ServPort: port; interface:
DiskInterface; log_unit: InterfaceInfo; var unitnum: Integer;
var DevName: DevPartString; var PartL: Pointer; var
PartL_Cnt: long): GeneralReturn

GetEnvVariable [*EnvMgr*] function(ServPort: Port; Name:
Env_Var_Name; SearchScope: Env_Var_Scope; var
Variable: Env_Variable; var Variable_Cnt: long; var
VarType: Env_Var_Type; var ActualScope:
Env_Var_Scope): GeneralReturn

GetEvent [*Sapph*] function(ServPort: Window; howWait:
KeyHowWait): EventRec

GetEventPort [*ModGetEvent*] procedure(ServPort: Window;
howWait: KeyHowWait; retPort: Port)

GetFullViewport [*ViewPt*] function(ServPort: Viewport):
Viewport

GetFullWindow [*Sapph*] function(ServPort: Window): Window

GetIconViewport [*Sapph*] procedure(ServPort: Window; var
iconvp: Viewport; var width: integer; var height: integer)

GetIconWindow [*Sapph*] function(ServPort: Window): Window

GetIOSleepID [*AccCall*] function(var SleepID: long):
GeneralReturn

GetIthWordPtr [*CommandParse*] function(i: long; CmndBlock:
CommandBlock): pWord_String

GetLibraryDef [*CFileDefs*] procedure(var BytePtr: long; var LibraryFileName: LString; var LibraryTimeStamp: Internal_Time; var LibraryLoc: long)

GetListenerWindow [*Sapph*] function(ServPort: Window): Window

GetOffsetDef [*CFileDefs*] procedure(var BytePtr: long; var WdOffsetAmt: long)

GetParsedUserInput [*ExtraCmdParse*] function(prompt: Cmnd_String; var inF: pCommand_File_List; var inputs: pCommand_Word_List; var outputs: pCommand_Word_List; var switches: pCommand_Word_List): GeneralReturn

GetPermSegPort [*AccInt*] function(ServPort: port; var PermSegmentPort: port): GeneralReturn

GetPortIndexStatus [*AccInt*] function(ServPort: port; PortIndex: Long; var Backlog: Integer; var NWaitingMsgs: Integer; var EWaitingMsgs: Integer; var PortRight: port; var PortType: Integer): GeneralReturn

GetPortStatus [*AccInt*] function(ServPort: port; PortRight: port; var Backlog: Integer; var NWaitingMsgs: Integer; var EWaitingMsgs: Integer; var PortIndex: Long; var PortType: Integer): GeneralReturn

GetPrimaryDef [*CFileDefs*] procedure(var BytePtr: long; var SimKind: Bit8; var AreaDesg: Bit8; var AreaOffset: long; var SymSize: long; var SymName: LString)

GetRectangleParms [*AccInt*] function(ServPort: port; RectPort: port; var BaseAddr: VirtualAddress; var ScanWidth: Integer; var BaseX: Integer; var BaseY: Integer; var MaxX: Integer; varMaxY: Integer; var IsFont: Boolean): GeneralReturn

GetRegionCursor [*ViewPt*] procedure(ServPort: Viewport; regionNum: integer; var cursorImage: CursorSet; var cursIndex: integer; var cursFunc: CursorFunction; var track: boolean)

GetRegionParms [*ViewPt*] procedure(ServPort: Viewport;

regionNum: integer; var absolute: boolean; var speed: integer; var minx: integer; var maxx: integer; var miny: integer; var maxy: integer; var modx: integer; var posx: integer; var mody: integer; var posy: integer)

GetScreenParameters [*Sapph*] procedure(ServPort: Window; var width: integer; var height: integer)

GetShellCmd [*ExtraCmdParse*] function(SearchTable: pWord_Search_Table; var CmdName: Cmnd_String; var inF: pCommand_File_List; var inputs: pCommand_Word_List; var outputs: pCommand_Word_List; var switches: pCommand_Word_List; var ErrorGR: GeneralReturn): integer

GetStringTime [*Time*] function(ServPort: port; TimeFormat: integer): String

GetSysFont [*ViewPt*] function(ServPort: Viewport): Viewport

GetTranslatedEvent [*KeyTran*] function(pKey: pKeyTab; win: window; howwait: KeyHowWait): KeyEvent

GetUserName [*Auth*] function(ServPort: Port; UserID: User_ID; var UserName: Auth_Var): GeneralReturn

GetUserTime [*Time*] function(ServPort: port): User_Time

GetViewportBit [*ViewPt*] function(ServPort: Viewport; x: integer; y: integer; var value: boolean): boolean

GetViewportRectangle [*ViewPt*] function(ServPort: Viewport; x: integer; y: integer; width: integer; height: integer; var Data: pVPIIntegerArray; var Data_Cnt: long; var WordsAcross: integer; ux: integer; uy: integer): boolean

GetVPRank [*ViewPt*] function(ServPort: Viewport): integer

GetWinNames [*Sapph*] procedure(ServPort: Window; var names: pWinNameArray; var names_Cnt: long; var curListenIndex: integer)

GetWinProcess [*Sapph*] function(ServPort: Window): Port

GlobalProcedure [*CFileDefs*] const = 3

GPIBuffer [*AccentType*] type = packed array[1..8] of Bit8

GPIBDevCmdHead [*IODefs*] type = packed record Options:

packed record SetInt0Mask: boolean; SetInt1Mask: boolean;
OmitBusConfig: boolean; OmitUnListen: boolean; case
boolean of true: (HoldOffOnEOI: boolean; unused2: Bit3);
false: (OmitGoToStandby: boolean; WaitOnData: boolean;
ForceEOI: boolean; unused1: Bit1); end; Int0Mask: Bit8;
Int1Mask: Bit8; PrimAddr: Bit8; SecAddr: Bit8; case
boolean of true: (ReadCount: Bit8); false: () end

GPIBSenseStatus [*IODefs*] type = record IntStat0: Bit8; IntStat1:
Bit8; IntAddrStat: Bit8; IntBusStat: Bit8; IntAddrSwch:
Bit8; IntCmdPass: Bit8; CurAddrStat: Bit8; CurBusStat:
Bit8; CurAddrSwch: Bit8; CurCmdPass: Bit8; end

GPIBWriteRegister [*IODefs*] type = packed record case
RegNum: Bit8 of 0: (); 1: (RegVal: Bit8) end

GR_Error_Type [*SaltError*] type = (GR_Warning, GR_Error,
GR_FatalError)

GRError [*PascalInit*] exception(GR: GeneralReturn)

GRErrorMsg [*SaltError*] procedure(GR: GeneralReturn;
ER_Type: GR_Error_Type; ProgName: String; InMsg:
PString; var OutMsg: PString)

Group_ID [*SesameDefs*] type = long

GRStdErr [*SaltError*] function(GR: GeneralReturn; ER_Type:
GR_Error_Type; InMsg: PString; var OutMsg: PString):
boolean

GRStdError [*SaltError*] procedure(GR: GeneralReturn;
ER_Type: GR_Error_Type; InMsg: PString; var OutMsg:
PString)

GRWriteErrorMsg [*SaltError*] procedure(GR: GeneralReturn;
ER_Type: GR_Error_Type; ProgName: String; InMsg:
PString)

GRWriteStdError [*SaltError*] procedure(GR: GeneralReturn;
ER_Type: GR_Error_Type; InMsg: PString)

HASHFRAMESIZE [*IFileDefs*] const = 31

HashmaskNumber [*KeyTran*] const = 31

Header [*AccentType*] type = packed Array[0..15] of Bit8

HeapAddress [*Dynamic*] type = record case integer of 0: (Offset:

HeapOffset; Segment: HeapNumber); 1: (AnyPtr: Pointer);
2: (LongInt: Long); end

HeapNumber [Dynamic] type = integer

HeapOffset [Dynamic] type = integer

iBlueMask [KeyTran] const = #04

IconAutoUpdate [Sapph] procedure(ServPort: Window; allowed:
boolean)

IconHeight [SapphDefs] const = 64

IconWidth [SapphDefs] const = 64

IDClrEtherFilter [EtherTypes] const = EtherIDBase+3

IDClrPupFilter [EtherTypes] const = EtherIDBase+6

Identifier [Stream] type = string[IdentLength]

IdentifyWindow [Sapph] procedure(ServPort: Window)

IdentLength [Stream] const = 8

IdentTable [Stream] type = array[0..1] of Identifier

IDGetEtherAddress [EtherTypes] const = EtherIDBase+1

IdNotDefined [Stream] exception(FileName: SName; Id:
Identifier)

IdNotUnique [Stream] exception(FileName: SName; Id:
Identifier)

IDRClrEtherFilter [EtherTypes] const = EtherIDBase+103

IDRClrPupFilter [EtherTypes] const = EtherIDBase+106

IDRecvEtherPacket [EtherTypes] const = EtherIDBase+104

IDRecvPupPacket [EtherTypes] const = EtherIDBase+107

IDRGetEtherAddress [EtherTypes] const = EtherIDBase+101

IDRSetEtherFilter [EtherTypes] const = EtherIDBase+102

IDRSetPupFilter [EtherTypes] const = EtherIDBase+105

IDSendEtherPacket [EtherTypes] const = EtherIDBase+4

IDSendPupPacket [EtherTypes] const = EtherIDBase+7

IDSetEtherFilter [EtherTypes] const = EtherIDBase+2

IDSetPupFilter [EtherTypes] const = EtherIDBase+5

iGreenMask [KeyTran] const = #10

iLeftMask [KeyTran] const = #02

IllegalBacklog [AccentType] const = AccErr+10

IllegalScanWidth [AccentType] const = AccErr+55

iMiddleMask [*KeyTran*] const = #01
ImpArray [*RunDefs*] type = packed array[0..4096] of SegIndex
ImpIndex [*RunDefs*] type = 0..RMAXIMPORT
Impossible [*ViewPt*] exception(s: String)
ImproperEntryType [*SesameDefs*] const =
 Sesame_Error_Base+11
in_out_separator_char [*CommandParse*] const = '~'
InactiveSegment [*AccentType*] const = AccErr+47
Index1Unquoted [*PathName*] function(S: Wild_Path_Name; C:
 char): integer
IndexInterpose [*AccInt*] function(ServPort: port; MyPort: Port;
 HisIndex: Long; var HisPort: Port): GeneralReturn
INDSIZE [*IFileDefs*] const = 32
Inf [*Spice_String*] const = ,32742
InitAccInt [*AccInt*] procedure(RPort: port)
InitAuth [*Auth*] procedure(RPort: port)
InitCmdFile [*CommandParse*] procedure(var inF:
 pCommand_File_List)
InitCommandParse [*CommandParse*] procedure
InitDiskUtils [*DiskUtils*] procedure(Disk: integer)
InitDynamic [*Dynamic*] procedure
InitEnvMgr [*EnvMgr*] procedure(RPort: port)
InitEther [*EtherUser*] procedure(RPort: port; Heap: integer)
InitExceptions [*Except*] procedure
Initial [*Spice_String*] function(Str1, Str2: PString): boolean
InitializedSymbol [*CFileDefs*] const = 5
InitIO [*IO*] procedure(Rport: port)
InitMsgId [*PascalInit*] const = 32896
InitMsgN [*MsgN*] procedure(RPort: port)
InitMsgSize [*PascalInit*] const = WORDSIZE(InitMsgType)*2
InitMsgType [*PascalInit*] type = RECORD Head: Msg;
 DefaultType: TypeType; DefInName: STRING[255];
 DefOutName: STRING[255]; PassPortType: TypeType;
 PassedPorts: PtrPortArray; UWinSharedType: TypeType;
 UWinShared: Boolean; EMTType: TypeType; EMPort: Port;

WindowType: TypeType; UserWindow: Port; TSType:
TypeType; UserTS: Port; WordCountType: TypeType;
WordCount: long; WordDirIndexType: TypeType;
WordDirIndex: long; WordArrayPtrType: TypeType;
WordArrayPtrEltSize: integer; WordArrayPtrTName:
integer; WordArray_Cnt: long; WordArrayPtr:
pCharacter_Pool; ComputeEnvType: TypeType;
ComputingEnvironment: Long; END

InitNet10MB [*Net10MB*] procedure(Rport: port)

InitPascal [*PascalInit*] procedure

InitProcess [*PascalInit*] procedure(AmIClone: BOOLEAN)

InitProcMgr [*ProcMgr*] procedure(RPort: port)

InitSapph [*Sapph*] procedure

InitSesame [*Sesame*] procedure(RPort: port)

InitSesDisk [*SesDisk*] procedure(RPort: port)

InitStream [*Stream*] procedure

InitTime [*Time*] procedure(RPort: port)

InitTS [*TS*] procedure(Rport: port)

InitViewPt [*ViewPt*] procedure

InitWordSearchTable [*CommandParse*] procedure(var table:
pWord_Search_Table; CaseSensitive: boolean)

InPorts [*PascalInit*] var: ptrPortArray

InPorts_Cnt [*PascalInit*] var: long

InsertAllRights [*AccInt*] function(ServPort: port; PortIndex:
Long; PortRight: port; PortType: Integer): GeneralReturn

InsertChars [*Spice_String*] procedure(Source: PString; var Dest:
PString; Index: Integer)

IntAccessFlags [*IFileDefs*] type = bit4

intblock [*DiskUtils*] type = record IntArray:
array[0..PAGEWORDSIZE-1] of integer; end

InterceptSegmentCalls [*AccInt*] function(ServPort: port; var
OldSysPorts: PtrAllPortArray; var OldSysPorts_Cnt: long;
var SysPorts: PtrPortArray; var SysPorts_Cnt: long):
GeneralReturn

InterfaceInfo [*AccentType*] type = Packed Record Case

DiskInterface Of EIO, CIO, FlopDrives: (Unit: Integer);
ENet: (EAddr: Packed Array[0..2] of integer); End

Internal_Time [TimeDefs] type = record Weeks: integer;
 MSecInWeek: long; end

inthdr [DiskUtils] type = record IntArray: array[0..7] of integer;
 end

Intr [AccentType] const = AccErr+12

InvalidateMemory [AccInt] function(ServPort: port; Address:
 VirtualAddress; NumBytes: Long): GeneralReturn

InvalidDirectoryVersion [SesameDefs] const =
 Sesame_Error_Base+6

InvalidVersion [SesameDefs] const = Sesame_Error_Base+5

InxCase [Except] exception

IO_Version [IO] function(ServerPort: ServerNamePort):
 IOString_80

IOAborted [IODefs] const = 4043

IOAsynch [IO] function(InP: Pointer): boolean

IOBadBaudRate [IODefs] const = 4012

IOBadBlockingFactor [IODefs] const = 4040

IOBadBufferSize [IODefs] const = 4035

IOBadCmdBlkCount [IODefs] const = 4008

IOBadCylinderNumber [IODefs] const = 4021

IOBadDataByteCount [IODefs] const = 4009

IOBadHeadNumber [IODefs] const = 4022

IOBadPortReference [IODefs] const = 4006

IOBadRegisterNumber [IODefs] const = 4010

IOBadSectorNumber [IODefs] const = 4020

IOBadTrack [IODefs] const = 4031

IOBadUserEventPort [IODefs] const = 4005

IOBaseMsgID [IODefs] const = 4000

IOCCancelled [IODefs] const = 4037

IOCircBufOverFlow [IODefs] const = 4017

IOCmdBlk [IODefs] type = packed record CmdIDTag: long; case
 integer of 0: (case integer of 0: (CmdByte: packed
 array[0..0] of Bit8); 1: (CmdWord: packed array[0..0] of

integer); 2: (WriteReg: packed array[stretch(0)..stretch(0)] of packed record RegNum: Bit8; RegVal: Bit8 end)); 1: (case IOCommand of IOSetAttention: (GPIBEnableATN: boolean); IOSetStream: (GPIBEnableStream: boolean; GPIBBlockingFactor: integer); IOSetBufferSize: (GPIBBufferSize: long); IOWriteRegisters: (GPIBWriteReg: packed array[stretch(0)..stretch(0)] of GPIBWriteRegister); IODevRead, IODevWrite: (GPIBDevCmdBlk: GPIBDevCmdHead)); 2: (case IOCommand of IOSetAttention: (RS232EnableATN: boolean); IOSetStream: (RS232EnableStream: boolean; RS232BlockingFactor: integer); IOSetBufferSize: (RS232BufferSize: long); IOSetBaud: (RS232TxBaud: Bit8; RS232RxBaud: Bit8); IOWriteRegisters: (RS232WriteReg: packed array[stretch(0)..stretch(0)] of SIOWriteRegister)); 3: (case IOCommand of IOSetAttention: (SpeechEnableATN: boolean); IOSetBufferSize: (SpeechBufferSize: long); IOSetBaud: (SpeechTxRate: integer); IOWriteRegisters: (SpeechWriteReg: packed array[stretch(0)..stretch(0)] of SIOWriteRegister)); 4: (case IOCommand of IOSetAttention: (FloppyEnableATN: boolean); IOSetDensity: (FloppyDensity: DensityType); IORead, IOWrite, IOFormat, IOSeek, IORecalibrate, IOReadID, IOSenseDrive: (FloppyUnit: Bit8; FloppyHead: Bit8; case IOCommand of IORead, IOWrite, IOFormat, IOSeek: (FloppyCylinder: Bit8; case IOCommand of IORead, IOWrite: (FloppySector: Bit8); IOFormat: (FloppyFmtData: Bit8)))) end

IOCommand [*IODefs*] type = (IOSense, IOReset,
IOWriteRegisters, IOFlushInput, IOFlushOutPut, IORead,
IOWrite, IOWriteEOI, IOReadHiVol, IOWriteHiVol,
IODevRead, IODevWrite, IOSetBaud, IOSetStream,
IOSetAttention, IOAbort, IOSuspend, IOResume, IOSeek,
IORecalibrate, IOFormat, IOReadID, IOSenseDrive,
IOSetDensity, IOSetBufferSize, IONullCmd)

IOCompletion [*IO*] exception(TransactionID: long; DataBuf:
pDataBuf; DataBuf_Cnt: long; Status: IOStatusBlk)

IOCylinderMisMatch [*IODefs*] const = 4032

IODataCRCError [*IODefs*] const = 4029

IODeviceNotFree [*IODefs*] const = 4003

IODeviceNotReady [*IODefs*] const = 4024

IODeviceNotWritable [*IODefs*] const = 4027

IODriveReadyChanged [*IODefs*] const = 4033

IOEBSE [*EtherTypes*] const = #4

IOEFUZ [*EtherTypes*] const = #5

IOEIOC [*EtherTypes*] const = #0

IOEndOfFrame [*IODefs*] const = 4018

IOEndOfInput [*IODefs*] const = 4019

IOEPTL [*EtherTypes*] const = #2

IOERNT [*EtherTypes*] const = #3

IOErr [*IODefs*] const = 4000

IOETIM [*EtherTypes*] const = #1

IOExclusionFailure [*IODefs*] const = 4036

IOFramingError [*IODefs*] const = 4016

IOGetTime [*Clock*] function: long

IOHeaderCRCError [*IODefs*] const = 4030

IOIllegalCommand [*IODefs*] const = 4007

IOInvalidIOPort [*IODefs*] const = 4004

IOMessage [*IODefs*] type = record Head: Msg; Body:
array[0..1023] of integer; end

IOMissingDataAddrMark [*IODefs*] const = 4025

IOMissingHeaderAddrMark [*IODefs*] const = 4026

IOMustBeAsynchronous [*IODefs*] const = 4039

IOMustEnableAsyncIO [*IODefs*] const = 4041

IONoDataFound [*IODefs*] const = 4013

IONotEnoughData [*IODefs*] const = 4034

IONotEnoughRoom [*IODefs*] const = 4011

IOOverRun [*IODefs*] const = 4014

IOParityError [*IODefs*] const = 4015

IOSectorNotFound [*IODefs*] const = 4028

IOSenseStatusBlk [*IODefs*] type = packed record StatusCnt:
 Bit8; case integer of 1: (GPIBStatus: GPIBSenseStatus); 2:
 (SIOStatus: SIOSenseStatus); 3: (FloppyStatus:
 FloppyResultStatus); 0: (case integer of 1: (StatusByte:
 packed array[stretch(1)..stretch(1)] of Bit8); 2:
 (StatusWord: packed array[stretch(1)..stretch(1)] of
 integer); 3: (SByte: packed array[stretch(1)..stretch(12)] of
 Bit8)) end

IOServerFull [*IODefs*] const = 4042

IOStartIOComplete [*IODefs*] const = 4038

IOStatusBlk [*IODefs*] type = packed record CmdIDTag: long;
 HardStatus: integer; SoftStatus: integer;
 CmdBytesTransferred: long; DataBytesTransferred: long;
 DeviceStatus: IOSenseStatusBlk; end

IOString_80 [*IODefs*] type = string[80]

IOSuccess [*IODefs*] const = 101

IOTimeOut [*IODefs*] const = 4002

IOUndefinedError [*IODefs*] const = 4001

IOUndeterminedEquipFault [*IODefs*] const = 4023

iRightMask [*KeyTran*] const = #10

IsChild [*AccentType*] const = AccErr+36

IsParent [*AccentType*] const = AccErr+35

IsPattern [*PMatch*] function(var str: pms255): boolean

IsQuotedChar [*PathName*] function(S: Wild_Path_Name; Index:
 integer): boolean

IsStreamDevice [*Stream*] function(S: SName): integer

iWhiteMask [*KeyTran*] const = #02

iYellowMask [*KeyTran*] const = #01

JumpControlStore [*ControlStore*] procedure(Adrs: integer)

KBFlushBoardOutput [*Stream*] procedure(var F: FileType)

KERNELPORT [*AccentType*] const = 1

KeyBACKSPACE [*KeyTran*] const = 8

KeyBREAK [*KeyTran*] const = 128

KeyControlBACKSPACE [*KeyTran*] const = 128+8

KeyControlBREAK [*KeyTran*] const = 130

KeyControlDEL [*KeyTran*] const = 128+127
KeyControlDOWNARROW [*KeyTran*] const = 29
KeyControlESC [*KeyTran*] const = 128+27
KeyControlHELP [*KeyTran*] const = 128+7
KeyControlINS [*KeyTran*] const = 128+27
KeyControlLEFTARROW [*KeyTran*] const = 30
KeyControlLF [*KeyTran*] const = 128+10
KeyControlNOSCROLL [*KeyTran*] const = 14
KeyControlOOPS [*KeyTran*] const = 128+21
KeyControlRETURN [*KeyTran*] const = 128+13
KeyControlRIGHTARROW [*KeyTran*] const = 31
KeyControlSETUP [*KeyTran*] const = 6
KeyControlSHIFTBREAK [*KeyTran*] const = 131
KeyControlSPACE [*KeyTran*] const = 128+32
KeyControlTAB [*KeyTran*] const = 128+9
KeyControlUPARROW [*KeyTran*] const = 28
KeyDEL [*KeyTran*] const = 127
KeyDOWNARROW [*KeyTran*] const = 129
KeyENTER [*KeyTran*] const = 140
KeyESC [*KeyTran*] const = 27
KeyEvent [*KeyTranDefs*] type = record vp: viewport; X, Y:
 integer; region: Integer; Cmd: 0..255; Ch: char; end
KeyHELP [*KeyTran*] const = 7
KeyHowWait [*SapphDefs*] type = (KeyWaitDiffPos,
 KeyDontWait, KeyWaitEvent)
KeyINS [*KeyTran*] const = 27
KeyKind [*KeyTranDefs*] type = (Standard, Control, Mouse,
 NonStandard)
KeyLEFTARROW [*KeyTran*] const = 130
KeyLF [*KeyTran*] const = 10
KeyMap [*KeyTranDefs*] type = packed record hashkey: KeyState;
 cmd: 0..255; chval: char; escty: EscKind; next:
 0..MAXMAP; region: 0..WILDREGION; end
KeyN0 [*KeyTran*] const = 150
KeyN1 [*KeyTran*] const = 151

KeyN2 [*KeyTran*] const = 152
KeyN3 [*KeyTran*] const = 153
KeyN4 [*KeyTran*] const = 154
KeyN5 [*KeyTran*] const = 156
KeyN6 [*KeyTran*] const = 157
KeyN7 [*KeyTran*] const = 158
KeyN8 [*KeyTran*] const = 159
KeyN9 [*KeyTran*] const = 160
KeyNCOMMA [*KeyTran*] const = 146
KeyNMINUS [*KeyTran*] const = 147
KeyNOSCROLL [*KeyTran*] const = 12
KeyNPERIOD [*KeyTran*] const = 148
KeyOOPS [*KeyTran*] const = 21
KeyPF1 [*KeyTran*] const = 132
KeyPF2 [*KeyTran*] const = 133
KeyPF3 [*KeyTran*] const = 134
KeyPF4 [*KeyTran*] const = 139
KeyRETURN [*KeyTran*] const = 13
KeyRIGHTARROW [*KeyTran*] const = 131
KeySETUP [*KeyTran*] const = 11
KeySHIFTBREAK [*KeyTran*] const = 129
KeySPACE [*KeyTran*] const = 32
KeyState [*SapphDefs*] type = packed record case integer of 1:
 (code: 0..#177; cntrlon: boolean; special: boolean; EscCl:
 0..#7; mbuttons: 0..#17); 2: (fullCode: 0..#777;
 escClAndMButtons: 0..#177); 3: (ks: integer); end
KeyTAB [*KeyTran*] const = 9
KeyTran_Error_Base [*KeyTranDefs*] const = 4600
KeyTransTab [*KeyTranDefs*] type = packed record Version:
 0..255; CmdState: 0..255; RawKeyBoard: boolean;
 EscState: 0..7; NumMaps: 0..MAXMAP; HashMask:
 integer; Len: integer; Class: packed array[0..MaxCode] of
 KeyKind; Map: packed array[1..1] of KeyMap; end
KeyTranVersion [*KeyTranDefs*] const = 4
KeyUPARROW [*KeyTran*] const = 128

KOENName [*CFileDefs*] function(x: Bit8): string
LandScapeBitHeight [*SapphDefs*] const = 1024
LandScapeBitWidth [*SapphDefs*] const = 1280
Language [*SegDefs*] type = (Pascal, Fortran, Imp)
LargeNumber [*Stream*] exception(FileName: SName)
LargeReal [*Stream*] exception(FileName: SName)
LastRecMsg [*IPCRecordIO*] var: RRecMsg
LibraryDef [*CFileDefs*] const = 90
LineFunct [*SapphDefs*] type = (DrawLine, EraseLine, XORLine)
LinkFileType [*RunDefs*] type = (SegFile, RunFile, DataFile,
 SymsFileType, QMapFileType)
LinkTypeStr [*ALoad*] function(typ: LinkFileType): string
ListLoggedInUsers [*Auth*] function(ServPort: Port; var UserList:
 Logged_User_List; var UserList_Cnt: long): GeneralReturn
LMsg [*AccCall*] type = record H: Msg; D:
 array[1..MAXMSGDATA] of integer; end
Ln [*RealFunctions*] function(X: Real): Real
LoadControlStore [*ControlStore*] procedure(var F: MicroFile)
LoadFontData [*ViewPt*] function(ServPort: Viewport; FontData:
 pVPIIntegerArray; FontData_Cnt: long): Viewport
LoadFontFile [*SapphLoadFile*] function(vp: Viewport; fileName:
 VPStr255): Viewport
LoadKeyTable [*KeyTran*] function(name: Path_Name; var pKey:
 pKeyTab): generalreturn
LoadMicroInstruction [*ControlStore*] procedure(Adrs: integer;
 MI: MicroInstruction)
LoadVPCursors [*SapphLoadFile*] function(vp: viewport;
 fileName: VPStr255; var numCursors: Integer): CursorSet
LoadVPPicture [*SapphLoadFile*] function(vp: Viewport;
 fileName: VPStr255; width, height: Integer): Viewport
LocalLabel [*CFileDefs*] const = 6
LocalProcedure [*CFileDefs*] const = 2
LOCALPT [*AccentType*] const = 2
LockPorts [*AccCall*] function(LockThem: boolean; Ports:
 ptrLPortArray; PortsCount: long): GeneralReturn

Log10 [*RealFunctions*] function(X: Real): Real
Logged_User [*AuthDefs*] type = record UserID: User_ID;
 UserName: Auth_Var; MachineName: Auth_Var; End
Logged_User_Array [*AuthDefs*] type = array[0..0] of
 Logged_User
Logged_User_List [*AuthDefs*] type = ^Logged_User_Array
LoginUser [*Auth*] function(ServPort: Port; UserName: Auth_Var;
 Password: Auth_Var; MachineName: Auth_Var; var
 UserAuthPort: Port; var UserRec: UserRecord);
 GeneralReturn
LOGMSGS [*AccCall*] const = false
LogoutUser [*Auth*] function(ServPort: Port): GeneralReturn
LogSmall [*RealFunctions*] exception(X: Real)
Lookup [*MsgN*] function(ServPort: Port; PortsName: string; var
 PortsID: Port): GeneralReturn
Lop [*Spice_String*] function(var Str: PString): PString
LPortArray [*AccentType*] type =
 array[stretch(0)..stretch(#77777)] of Port
LString [*CFileDefs*] type = String[255]
M_CHILDFORKREPLY [*AccentType*] const = #100+#11
M_DEBUGMSG [*AccentType*] const = #100+#12
M_GENERALKERNELREPLY [*AccentType*] const = #100+6
M_KERNELMSGERROR [*AccentType*] const = #100+7
M_MSGACCEPTED [*AccentType*] const = #100+2
M_OWNERSHIPRIGHTS [*AccentType*] const = #100+3
M_PARENTFORKREPLY [*AccentType*] const = #100+#10
M_PORTDELETED [*AccentType*] const = #100+1
M_RECEIVERIGHTS [*AccentType*] const = #100+4
Machine_Name [*AuthDefs*] type = String[255]
MachineInfoRec [*BootInfo*] type = packed record case boolean
 of false: (int: integer); true: (WCSSize: 0..15; Reserved:
 0..3; IsPortrait: Boolean; BoardRev: 0..31; OldZ80:
 boolean; CMUNet: boolean; Reserved2: 0..3) end
MAINKLUDGE [*RunDefs*] const = 255
MakeAnEmptyKeyTable [*KeyTran*] function(Raw: boolean; var

pKey: pkeytab): GeneralReturn
MakeViewport [*ViewPt*] function(ServPort: Viewport; x: integer;
y: integer; w: integer; h: integer; rank: integer; memory:
boolean; courteous: boolean; transparent: boolean):
Viewport
MakeWinListener [*Sapph*] procedure(ServPort: Window)
MapAddr [*DiskUtils*] function(LogAddr: DiskAddress; var Disk:
integer): DiskAddress
MapFull [*AccentType*] const = AccErr+25
MAPNIL [*KeyTranDefs*] const = 0
Max_QmapRoutines [*QMapDefs*] const = 251
Max_SymRoutines [*SymDefs*] const = 251
Max_Users [*AuthDefs*] const = 1023
MAXBACKLOG [*AccentType*] const = 63
MaxCode [*SapphDefs*] const = 127+16
MaxCoord [*SapphDefs*] const = 16000
MAXDEVICES [*AccentType*] const = 5
MAXDISKS [*AccentType*] const = 4
MAXDPCHARS [*AccentType*] const = 25
MaxEtherWords [*EtherTypes*] const = 748
MAXLOGMESS [*AccCall*] const = 20
MAXMAP [*KeyTranDefs*] const = 511
MAXMSGDATA [*AccCall*] const = 20
MaxNumRectangles [*SapphDefs*] const = 256 div 4
MAXPARTCHARS [*AccentType*] const = 8
MAXPARTITIONS [*AccentType*] const = 30
MAXPORTS [*AccentType*] const = 256
MAXPROCS [*AccentType*] const = 63
MaxPStringSize [*Spice_String*] const = 255
MaxPupWords [*EtherTypes*] const = 533
MaxSignal [*ProcMgrDefs*] const = SignalBase+63
MaxSize [*SapphDefs*] const = 16000
MemFault [*AccentType*] const = AccErr+6
MemProtection [*AccentType*] type =
 READONLY..READWRITE

MessagesWaiting [AccCall] function(MsgType: long; var Ports:
ptrLPortArray; var PortsCount: long): GeneralReturn

MicroBinary [ControlStore] type = record Adrs: integer; MI:
MicroInstruction end

MicroFailure [AccentType] const = AccErr+41

MicroFile [ControlStore] type = file of MicroBinary

MicroInstruction [ControlStore] type = packed record case
integer of 0: (Word1: integer; Word2: integer; Word3:
integer); 1: (Jmp: 0..15; Cnd: 0..15; Z: 0..255; SF: 0..15;
F: 0..3; ALU: 0..15; H: 0..1; W: 0..1; B: 0..1; A: 0..7; Y:
0..255; X: 0..255); 2: (JmpCnd: 0..255; Fill1: 0..255; SFF:
0..63; ALU0: 0..1; ALU1: 0..1; ALU23: 0..3) end

MicroSeconds [AccentType] type = long

MinCoord [SapphDefs] const = -16000

MinEtherWords [EtherTypes] const = WordSize(EtherHeader)

MinPupWords [EtherTypes] const = WordSize(PupHeader)+1

MinSignal [ProcMgrDefs] const = SignalBase

MiscEtc_CompilerTemp [SymDefs] const = 0

MiscEtc_Register_16Bit [SymDefs] const = 1

MiscEtc_Register_32Bit [SymDefs] const = 2

ModifyRegion [ViewPt] procedure(ServPort: Viewport;
regionNum: integer; leftx: integer; topy: integer; width:
integer; height: integer)

ModifyVP [ViewPt] procedure(ServPort: Viewport; newlx:
integer; newty: integer; newwidth: integer; newheight:
integer; newrank: integer; wantVpChEx: boolean)

ModifyWindow [Sapph] procedure(ServPort: Window; newleftx:
integer; newtopy: integer; newouterwidth: integer;
newouterheight: integer; newRank: integer)

ModNameLength [RunDefs] const = 19

ModString [RunDefs] type = String[ModNameLength]

MoveWords [AccCall] function(SrcAddr: VirtualAddress; var
DstAddr: VirtualAddress; NumWords: long; Delete:
boolean; Create: boolean; Mask: long; DontShare: boolean):
GeneralReturn

MParity [*Except*] exception

Msg [*AccentType*] type = record SimpleMsg: boolean; MsgSize: long; MsgType: long; LocalPort: Port; RemotePort: Port; ID: long; end

Msg_Version [*MsgN*] function(ServPort: Port; var Version: string): GeneralReturn

MsgClrEtherFilter [*EtherTypes*] type = RECORD Head: Msg; TTyp: TypeType; Typ: INTEGER; TProcessPort: TypeType; ProcessPort: Port END

MsgClrPupFilter [*EtherTypes*] type = RECORD Head: Msg; TSocket: TypeType; Socket: Long; TProcessPort: TypeType; ProcessPort: Port END

MsgEtherPacket [*EtherTypes*] type = RECORD Head: Msg; TPacket: TypeType; Packet: EtherPacket; END

MsgGetEtherAddress [*EtherTypes*] type = RECORD Head: Msg END

MsgIndex [*PascalInit*] const = 5

MsgInterrupt [*AccentType*] const = AccErr+43

MsgLog [*AccCall*] type = record Init: integer; MsgsSent: long; MsgsRec: long; NxtMsg: integer; LMsgs: array[0..MAXLOGMESS] of record Sent: boolean; InProg: boolean; GR: GeneralReturn; M: LMsg; end; end

MsgPacket [*EtherTypes*] type = RECORD Head: Msg; TPackeT: TypeType; END

MsgPortStatus [*MsgN*] function(ServPort: Port; PortsID: Port; var GlobalPort: long; var Owner: long; var Receiver: long; var SrcID: long; var SeqNum: long; var NetWaiting: boolean; var NumQueued: integer; var Blocked: boolean; var Locked: boolean; var RecvQueue: integer; var DataOffset: long; var InSrcID: long; var InSeqNum: long): GeneralReturn

MsgPupPacket [*EtherTypes*] type = RECORD Head: Msg; TPackeT: TypeType; Packet: PupPacket; END

MsgRClrEtherFilter [*EtherTypes*] type = RECORD Head: Msg; TTyp: TypeType; Typ: INTEGER; TAnswer: TypeType;

Answer: BOOLEAN END

MsgRClrPupFilter [*EtherTypes*] type = RECORD Head: Msg;
TSocket: TypeType; Socket: Long; TAnswer: TypeType;
Answer: Boolean END

MsgRecvEtherPacket [*EtherTypes*] type = MsgEtherPacket

MsgRecvPupPacket [*EtherTypes*] type = MsgPupPacket

MsgRGetEtherAddress [*EtherTypes*] type = RECORD Head:
Msg; TEtherAddress: TypeType; EtherAddress: INTEGER
END

MsgRSetEtherFilter [*EtherTypes*] type = RECORD Head: Msg;
TTyp: TypeType; Typ: INTEGER; TAnswer: TypeType;
Answer: BOOLEAN END

MsgRSetPupFilter [*EtherTypes*] type = RECORD Head: Msg;
TSocket: TypeType; Socket: Long; TAnswer: TypeType;
Answer: Boolean END

MsgSendEtherPacket [*EtherTypes*] type = MsgEtherPacket

MsgSendPupPacket [*EtherTypes*] type = MsgPupPacket

MsgSetEtherFilter [*EtherTypes*] type = RECORD Head: Msg;
TTyp: TypeType; Typ: INTEGER; TProcessPort:
TypeType; ProcessPort: Port END

MsgSetPupFilter [*EtherTypes*] type = RECORD Head: Msg;
TSocket: TypeType; Socket: Long; TProcessPort:
TypeType; ProcessPort: Port END

MsgTooBig [*AccentType*] const = AccErr+20

MulOvfl [*Except*] exception

MultiLevelProgress [*WindowUtils*] procedure(Level: integer;
Current, Max: Long)

MultiStreamProgress [*WindowUtils*] procedure(Level: integer;
var F: File)

Name_Flags [*SesameDefs*] type = 0..#3

Name_Status [*SesameDefs*] type = 0..#7

NameAmbiguous [*ProcMgrDefs*] const = ProcMgrBase+8

NameBase [*NameErrors*] const = 1000

NameNotCheckedIn [*NameErrors*] const = NameBase+1

NameNotFound [*SesameDefs*] const = Sesame_Error_Base+1

NameNotYours [*NameErrors*] const = NameBase+0
NameServerPort [*PascalInit*] var: port
Net10MBAynch [*Net10MB*] function(*InP*: Pointer): boolean
Net_Version [*Net10MB*] function(*ServPort*: Port): A_String
NetFail [*AccentType*] const = AccErr+11
NETWORKTROUBLE [*AccentType*] const = 2
NewP [*Dynamic*] procedure(*S*: HeapNumber; *A*: integer; var
 Where: pointer; *L*: integer)
NewToOldTime [*OldTimeStamp*] function(*NewTime*:
 Internal_Time): TimeStamp
NextExtension [*PathName*] function(var *EList*: Extension_List):
 string
NextOp [*Except*] exception
NFlag_Deleted [*SesameDefs*] const = #000001
NFlag_NoNormal [*SesameDefs*] const = #000002
NFlag_RESERVED [*SesameDefs*] const = #177774
NilPointer [*Dynamic*] exception
No_User [*AuthDefs*] const = 0
NoAccess [*SesameDefs*] const = Sesame_Error_Base+9
NoAvailablePages [*AccentType*] const = AccErr+37
NoChildren [*ProcMgrDefs*] const = ProcMgrBase+6
NoEchoRead [*NoEchoInput*] function: PString
NoMorePorts [*AccentType*] const = AccErr+9
NOREPLY [*AccentType*] const = 4
NORMALMSG [*AccentType*] const = 0
NormMsg [*Except*] exception
NotACoreFile [*CoreLoad*] const = -1
NotADirectory [*SesameDefs*] const = Sesame_Error_Base+12
NotAFile [*SesameDefs*] const = Sesame_Error_Base+8
NotAFont [*AccentType*] const = AccErr+58
NotAnIPCCall [*AccentType*] const = AccErr+17
NotAPort [*AccentType*] const = AccErr+7
NotASystemAddress [*AccentType*] const = AccErr+50
NotAUserAddress [*AccentType*] const = AccErr+51
NotBoolean [*Stream*] exception(*FileName*: SName)

NotCurrentProcess [*AccentType*] const = AccErr+28
NotEnoughRoom [*AccentType*] const = AccErr+16
NotIdentifier [*Stream*] exception(FileName: SName)
NotNumber [*Stream*] exception(FileName: SName)
NotOpen [*Stream*] exception
NotPortReceiver [*AccentType*] const = AccErr+14
NotReal [*Stream*] exception(FileName: SName)
NotReset [*Stream*] exception(FileName: SName)
NotRewrite [*Stream*] exception(FileName: SName)
NotSamePartition [*SesameDefs*] const = Sesame_Error_Base+10
NotTextFile [*Stream*] exception(FileName: SName)
NotYourChild [*AccentType*] const = AccErr+21
NSMsgByteSize [*NSTypes*] const = 2*WordSize(NSMsgType)
NSMsgType [*NSTypes*] type = record Head: Msg; TType:
 TypeType; TName: integer; TSizeinBits: integer;
 TNumElts: long; pPkt: pE10Packet end
NStat_Deleted [*SesameDefs*] const = #000001
NStat_High [*SesameDefs*] const = #000002
NStat_Low [*SesameDefs*] const = #000004
NStat_RESERVED [*SesameDefs*] const = #177770
Null_CommandBlock [*CommandDefs*] function: CommandBlock
NULLPORT [*AccentType*] const = 0
NULLViewPort [*SapphDefs*] const = NullPort
NullWindow [*SapphDefs*] const = NullPort
NUMMSGTYPES [*AccentType*] const = 2
NUMPRIORITIES [*AccentType*] const = 16
NumProgressBars [*SapphDefs*] const = 2
NUMQUEUES [*AccentType*] const =
 NUMSLEEPQS+NUMPRIORITIES+5
NUMSLEEPQS [*AccentType*] const = 32
OFFSCREEN [*SapphDefs*] const = -32002
OFFSETBASE [*CFileDefs*] const = 16000
OffsetDef [*CFileDefs*] const = 11
OffsetPairArray [*SapphFileDefs*] type = array[0..126] of record
 x: integer; y: integer; end

OldCurrentTime [*OldTimeStamp*] function: TimeStamp
OldToNewTime [*OldTimeStamp*] function(OldTime:
 TimeStamp): Internal_Time
OpenCmdFile [*CommandParse*] function(FileName:
 pWord_String; var inF: pCommand_File_List):
 GeneralReturn
OpenIO [*IO*] function(ServerPort: ServerNamePort; var IOPort:
 ServerIOPort; UserPort: UserEventPort): GeneralReturn
Other [*AccentType*] const = AccErr+13
OutOfImagSegments [*AccentType*] const = AccErr+49
OutOfPCSpace [*AccentType*] const = AccErr+23
OutOfRectangleBounds [*AccentType*] const = AccErr+54
OUTREGION [*SapphDefs*] const = 0
OvflI [*Except*] exception
OvrReal [*Except*] exception
Owner_NO_Read_Access [*IFileDefs*] const = #01
Owner_NO_Write_Access [*IFileDefs*] const = #02
Owner_Read_Access [*SesDiskDefs*] const = #1
Owner_Write_Access [*SesDiskDefs*] const = #2
PacketBytes [*EtherTypes*] const = 2*WordSize(MsgPacket)
PacketStats [*MsgN*] function(ServPort: Port; Reset: boolean; var
 PacketsQued: long; var PacketsSent: long; var PacketsRecd:
 long; var AckSsent: long; var AckSrecd: long; var
 ReXmitsSent: long; var ReXmitsRecd: long; var
 BlocksSent: long; var BlocksRecd: long): GeneralReturn
Pad [*Spice_String*] function(Str: PString; TotalLen: integer;
 PadCh: Char; Where: integer): Pstring
PAGEBITS [*AccentType*] const = 8
PAGEBYTESIZE [*AccentType*] const = 512
PAGEWORDSIZE [*AccentType*] const = PAGEBYTESIZE div
 2
ParseChPool [*CommandParse*] function(ChPool:
 pCharacter_Pool; PoolLength: Char_Pool_Index; var
 inputs: pCommand_Word_List; var outputs:
 pCommand_Word_List; var switches:

pCommand_Word_List): GeneralReturn
ParseCommand [*CommandParse*] function(var inputs:
 pCommand_Word_List; var outputs:
 pCommand_Word_List; var switches:
 pCommand_Word_List): GeneralReturn
ParseEtherAddress [*EtherUser*] function(MsgP:
 pMsgRGetEtherAddress): INTEGER
ParseEtherClear [*EtherUser*] function(MsgP:
 pMsgRClrEtherFilter; VAR Typ: INTEGER): BOOLEAN
ParseEtherFilter [*EtherUser*] function(MsgP:
 pMsgRSetEtherFilter; VAR Typ: INTEGER): BOOLEAN
ParseEtherPacket [*EtherUser*] function(MsgP:
 pMsgRecvEtherPacket; Packet: POINTER): INTEGER
ParseIllegalCharInEnvNam [*CommandDefs*] const =
 CmdParse_Error_Base+19
ParseIllegalCharInInRed [*CommandDefs*] const =
 CmdParse_Error_Base+22
ParseIllegalCharInOutRed [*CommandDefs*] const =
 CmdParse_Error_Base+23
ParseIllegalCharInQuoted [*CommandDefs*] const =
 CmdParse_Error_Base+20
ParseIllegalCharInShPara [*CommandDefs*] const =
 CmdParse_Error_Base+24
ParseIllegalCharInSwName [*CommandDefs*] const =
 CmdParse_Error_Base+17
ParseIllegalCharInSwVal [*CommandDefs*] const =
 CmdParse_Error_Base+18
ParseInternalFault [*CommandDefs*] const =
 CmdParse_Error_Base+15
ParseOnlyCmdAllowed [*CommandDefs*] const =
 CmdParse_Error_Base+21
ParsePupClear [*EtherUser*] function(MsgP: pMsgRClrPupFilter;
 VAR Socket: Long): BOOLEAN
ParsePupFilter [*EtherUser*] function(MsgP: pMsgRSetPupFilter;
 VAR Socket: Long): BOOLEAN

ParsePupPacket [*EtherUser*] procedure(MsgP:
 pMsgRecvPupPacket; PupPacket: POINTER)

ParseWordTooLong [*CommandDefs*] const =
 CmdParse_Error_Base+16

PartData [*SesDiskDefs*] type = record PartName: DevPartString;
 DevName: DevPartString; PartRootDir: SegID;
 PartNumFree: long; PartMounted: boolean; PartStart:
 DiskAddr; PartEnd: DiskAddr; PartKind: PartitionType;
 PartExUse: boolean; end

PartDisMount [*AccInt*] function(ServPort: port): GeneralReturn

PartDList [*SesDiskDefs*] type = array[1..MAXPARTITIONS] of
 PartData

PartialNameSize [*IFileDefs*] const = 80

PartialPathName [*IFileDefs*] type = string[PartialNameSize]

PartInfo [*AccentType*] type = record PartHeadFree: DiskAddr;
 PartTailFree: DiskAddr; PartInfoBlk: DiskAddr;
 PartRootDir: SegID; PartNumOps: integer; PartNumFree:
 long; PartInUse: boolean; PartMounted: boolean;
 PartDevice: integer; PartStart: DiskAddr; PartEnd:
 DiskAddr; PartKind: PartitionType; PartName: PartString;
 PartExUse: boolean; Unused: long; PartDiskRel: boolean;
 end

PartitionFull [*AccentType*] const = AccErr+59

PartitionType [*AccentType*] type = (Root, UnUsed, Segment,
 PLX)

PartList [*AccentType*] type = array[1..MAXPARTITIONS] of
 PartInfo

PartMount [*AccInt*] function(ServPort: port; PartName:
 DevPartString; ExUse: Boolean; var RootId: SegID; var
 PartKind: PartitionType; var PartPort: port; var PartS:
 DiskAddr; var PartE: DiskAddr): GeneralReturn

PartRecord [*DiskUtils*] type = record PartHeadFree:
 DiskAddress; PartTailFree: DiskAddress; PartInfoBlk:
 DiskAddress; PartRootDir: DiskAddress; PartNumOps:
 integer; PartNumFree: Bit32; PartInUse: boolean;

PartMounted: boolean; PartDevice: integer; PartStart:
DiskAddress; PartEnd: DiskAddress; PartKind:
PartitionType; PartName: PartString; PartExUse: boolean;
PartPort: Port end

PartString [*AccentType*] type = string[MAXPARTCHARS]

PascalProcedure [*CFileDefs*] const = 1

PASCALRECORD [*IPCRecordIO*] const = #12345

PassType [*AuthDefs*] type = Long

PassWordIncorrect [*AuthDefs*] const = Auth_Error_Base+2

PastEof [*Stream*] exception(FileName: SName)

Path_Name [*PathName*] type = string[Path_Name_Size]

Path_Name_Size [*SesameDefs*] const = 255

PattDebug [*PMatch*] procedure(v: boolean)

Pattern [*SapphFileDefs*] type = ^PatternMap

PatternMap [*SapphFileDefs*] type = array[0..63, 0..3] of integer

PatternMapView [*SapphFileDefs*] type = array[0..0] of
PatternMap

PattMap [*PMatch*] function(var str, inpatt, outpatt, outstr:
pms255; fold: boolean): boolean

PattMatch [*PMatch*] function(var str, pattern: pms255; fold:
boolean): boolean

pBit32 [*AccentType*] type = ^Bit32

pCharacter_Pool [*CommandDefs*] type = ^Character_Pool

PCInData [*CFileDefs*] const = DataState

PCInData2 [*CFileDefs*] const = Data2State

PCInText [*CFileDefs*] const = TextState

pCmdBlk [*IODefs*] type = Pointer

pCmnd_String [*CommandParse*] type = ^Cmnd_String

pCommand_File_List [*CommandParse*] type =
^Command_File_List

pCommand_Word_List [*CommandParse*] type =
^Command_Word_List

PCStateName [*CFileDefs*] function(x: Bit8): string

pCursorArrayRec [*SapphFileDefs*] type = ^CursorArrayRec

pDataBuf [*IODefs*] type = Pointer

pDirBlock [*IFileDefs*] type = ^DirBlock
pE10Message [*Net10MBDefs*] type = ^E10Message
pE10Packet [*Net10MBDefs*] type = ^E10Packet
pEtherPacket [*EtherTypes*] type = ^EtherPacket
pFEarray [*RunDefs*] type = ^FEarray
pFirstBlock [*CFileDefs*] type = ^FirstBlock
PhysicalAddress [*AccentType*] type = long
pImpArray [*RunDefs*] type = ^ImpArray
pImpInfo [*SegDefs*] type = ^CImpInfo
pIOCmdblk [*IODefs*] type = ^IOCmdblk
pIOMessage [*IODefs*] type = ^IOMessage
pKeyTab [*KeyTranDefs*] type = ^KeyTransTab
pLMsg [*AccCall*] type = ^LMsg
PMAddCtlWindow [*ProcMgr*] function(ServPort: port;
 CtlWindow: window; NewCtlWindow: window);
 GeneralReturn
PMBroadcast [*ProcMgr*] function(ServPort: port; s: string);
 GeneralReturn
PMChangeGroup [*ProcMgr*] function(ServPort: port; ProcPort:
 port; NewWindow: window); GeneralReturn
PMDebug [*ProcMgr*] function(ServPort: port; ProcID: string);
 GeneralReturn
PMDebugProcess [*ProcMgr*] function(ServPort: port; ProcPort:
 port; Reason: long); GeneralReturn
PMGetProcPorts [*ProcMgr*] function(ServPort: port; ProcPort:
 port; var hisWindow: window; var hisTypescript: typescript;
 var hisEMConn: port); GeneralReturn
PMGetStatus [*ProcMgr*] function(ServPort: port; ProcID: string;
 var Stats: StatList; var Stats_Cnt: long); GeneralReturn
PMGetTimes [*ProcMgr*] function(ServPort: port; ProcPort: port;
 var LoadTime: long; var RunTime: long; var ElapsedTime:
 long); GeneralReturn
PMGetWaitID [*ProcMgr*] function(ServPort: port; ProcPort:
 port; var WaitID: long); GeneralReturn
PMGroupSignal [*ProcMgr*] procedure(ServPort: port;

CtlWindow: window; Signal: SignalName)
PMIndex [*PascalInit*] const = 4
PMKill [*ProcMgr*] function(ServPort: port; ProcID: string):
 GeneralReturn
PMPort [*PascalInit*] var: port
PMProcessSignal [*ProcMgr*] procedure(ServPort: port; ProcPort:
 port; Signal: SignalName)
PMRegisterProcess [*ProcMgr*] function(ServPort: port;
 HisKPort: port; HisDPort: port; ProgName: string;
 HisWindow: window; HisTypescript: typescript; EMConn:
 port; Parent: port): GeneralReturn
PMRemoveCtlWindow [*ProcMgr*] function(ServPort: port;
 CtlWindow: window): GeneralReturn
PMResume [*ProcMgr*] function(ServPort: port; ProcID: string):
 GeneralReturn
pms255 [*PMatch*] type = String[255]
PMSaveLoadTime [*ProcMgr*] function(ServPort: port; ProcPort:
 port; LoadTime: long): GeneralReturn
PMSetDebugPort [*ProcMgr*] function(ServPort: port; ProcPort:
 port; DebugPort: port; DebugSignalOnly: boolean):
 GeneralReturn
PMSetPriority [*ProcMgr*] function(ServPort: port; ProcID:
 string; priority: integer): GeneralReturn
PMSetSignal [*ProcMgr*] function(ServPort: port; ProcPort: port;
 Signal: SignalName; Action: SignalAction): GeneralReturn
PMSetSignalPort [*ProcMgr*] function(ServPort: port; ProcPort:
 port; SignalPort: port): GeneralReturn
pMsgClrEtherFilter [*EtherTypes*] type = ^MsgClrEtherFilter
pMsgClrPupFilter [*EtherTypes*] type = ^MsgClrPupFilter
pMsgGetEtherAddress [*EtherTypes*] type =
 ^MsgGetEtherAddress
pMsgLog [*AccCall*] type = ^MsgLog
pMsgRClrEtherFilter [*EtherTypes*] type = ^MsgRClrEtherFilter
pMsgRClrPupFilter [*EtherTypes*] type = ^MsgRClrPupFilter
pMsgRecvEtherPacket [*EtherTypes*] type =

^MsgRecvEtherPacket
pMsgRecvPupPacket [*EtherTypes*] type = **^MsgRecvPupPacket**
pMsgRGetEtherAddress [*EtherTypes*] type =
 ^MsgRGetEtherAddress
pMsgRSetEtherFilter [*EtherTypes*] type = **^MsgRSetEtherFilter**
pMsgRSetPupFilter [*EtherTypes*] type = **^MsgRSetPupFilter**
pMsgSendEtherPacket [*EtherTypes*] type =
 ^MsgSendEtherPacket
pMsgSendPupPacket [*EtherTypes*] type = **^MsgSendPupPacket**
pMsgSetEtherFilter [*EtherTypes*] type = **^MsgSetEtherFilter**
pMsgSetPupFilter [*EtherTypes*] type = **^MsgSetPupFilter**
PMSignalByName [*ProcMgr*] function(ServPort: port; ProcID:
 string; ProcessOrGroup: boolean; Signal: SignalName):
 GeneralReturn
PMSuspend [*ProcMgr*] function(ServPort: port; ProcID: string):
 GeneralReturn
PMTerminate [*ProcMgr*] function(ServPort: port; ProcPort: port;
 Reason: long): GeneralReturn
pNSMsgType [*NSTypes*] type = **^NSMsgType**
Port [*AccentType*] type = long
PortArray [*AccentType*] type = array[0..MAXPORTS-1] of Port
PortBitArray [*AccentType*] type = packed
 array[0..MAXPORTS-1] of boolean
PortDeath [*AccentType*] type =
 EXPLICITDEALLOC..NETWORKTROUBLE
PortFull [*AccentType*] const = AccErr+3
PortInterpose [*AccInr*] function(ServPort: port; MyPort: Port;
 HisPort: port; var MyNewPort: Port): GeneralReturn
PortOption [*AccentType*] type = DEFAULTPTS..LOCALPT
PortraitBitHeight [*SapphDefs*] const = 1024
PortraitBitWidth [*SapphDefs*] const = 768
PortsWithMessages [*AccCall*] function(MsgType: long; var
 ports: PortBitArray): GeneralReturn
POS_Dir_Separator [*IFileDefs*] const = '>'
PosC [*Spice_String*] function(Str: PString; c: char): integer

POSSimpleName [*IFileDefs*] type = string[SimpleNameSize]
PosString [*Spice_String*] function(Source, Mask: PString):
 Integer
PosSystem [*SysType*] const = false
Power [*RealFunctions*] function(X, Y: Real): Real
PowerBig [*RealFunctions*] exception(X, Y: Real)
PowerI [*RealFunctions*] function(X: Real; Y: Integer): Real
PowerNeg [*RealFunctions*] exception(X, Y: Real)
PowerSmall [*RealFunctions*] exception(X, Y: Real)
PowerZero [*RealFunctions*] exception(X, Y: Real)
pPatternMapArray [*SapphFileDefs*] type = ^PatternMapArray
pPupData [*EtherTypes*] type = ^PupData
pPupPacket [*EtherTypes*] type = ^PupPacket
pQMap [*QMapDefs*] type = ^QCodeMap
pQMap_Buffer [*QMapDefs*] type = ^QMap_Buffer
pQMap_ByteBuffer [*QMapDefs*] type = ^QMap_ByteBuffer
pQMapDict [*QMapDefs*] type = ^QMapDict
PReadln [*Stream*] procedure(var F: Filetype)
pRectArray [*SapphDefs*] type = ^RectArray
PREVIEW [*AccentType*] const = 0
PrimaryDef [*CFileDefs*] const = 10
Print_Name [*SesameDefs*] type = string[Print_Name_Size]
Print_Name_Size [*SesameDefs*] const = 80
PriorID [*AccentType*] type = 0..NUMPRIORITIES-1
PROCESSDEATH [*AccentType*] const = 1
ProcessDeathMsg [*ProcMgrDefs*] type = record Head: Msg;
 tWaitID: TypeType; WaitID: long; tReason: TypeType;
 Reason: long; tLoadTime: TypeType; LoadTime: long;
 tRunTime: TypeType; RunTime: long; tElapsedTime:
 TypeType; ElapsedTime: long; end
ProcessDeathMsgID [*ProcMgrDefs*] const = 3800
ProcessDisowned [*ProcMgrDefs*] const = ProcMgrBase+9
ProcID [*AccentType*] type = integer
ProcMgr_Version [*ProcMgr*] function(ServPort: port): string
ProcMgrBase [*ProcMgrDefs*] const = 3600

ProcState [*AccentType*] type = (Supervisor, Privileged,
BadSupervisor, User)

ProgressInTitle [*SapphDefs*] const = 1

ProgStr [*SapphDefs*] type = String[ProgStrLength]

ProgStrLength [*SapphDefs*] const =
(IconWidth-2*BorderOverhead) div SysFontWidth

Prompt_Indention_String [*ExtraCmdParse*] const = ''

pRunHead [*RunDefs*] type = ^RunHead

pSEarray [*RunDefs*] type = ^SEarray

pSegBlock [*SegDefs*] type = ^SegBlock

pSourceMap [*QMapDefs*] type = ^SourceMap

PStatus [*AccentType*] type = record State: ProcState; Priority:
PriorID; MsgPending: boolean; EMsgPending: boolean;
MsgEnable: boolean; EMsgEnable: boolean; LimitSet:
boolean; SVStkInCore: boolean; QueueID: QID; SleepID:
ptrInteger; RunTime: long; LimitTime: long end

pStreamBuffer [*Stream*] type = ^StreamBuffer

pStreamF [*Stream*] type = ^StreamF

PString [*Spice_String*] type = String[MaxPStringSize]

ptrAllPortArray [*AccentType*] type = ^PortArray

ptrArguments [*AccentType*] type = ^Arguments

ptrBIRecord [*BootInfo*] type = ^BIRecord

ptrBoolean [*AccentType*] type = ^boolean

ptrDirIOArgs [*AccentType*] type = ^DirectioArgs

ptrDiskBlock [*DiskUtils*] type = ^DiskBlock

ptrDiskHeader [*DiskUtils*] type = ^DiskHeader

ptrFSDataEntry [*IFileDefs*] type = ^FSDataEntry

ptrInteger [*AccentType*] type = ^integer

ptrLPortArray [*AccentType*] type = ^LPortArray

ptrMsg [*AccentType*] type = ^Msg

ptrPartDList [*SesDiskDefs*] type = ^PartDList

ptrPartList [*AccentType*] type = ^PartList

ptrPort [*AccentType*] type = ^Port

ptrPortArray [*AccentType*] type = ^PortArray

ptrPortBitArray [*AccentType*] type = ^PortBitArray

pTSCharArray [*TSDefs*] type = ^TSCharArray
PupCMUNet [*EtherTypes*] const = #52
PupData [*EtherTypes*] type = RECORD CASE INTEGER OF 0:
 (Chars: PACKED ARRAY[0..1043] OF CHAR); 1: (Bytes:
 PACKED ARRAY[0..1043] OF 0..255); 2: (Words:
 PACKED ARRAY[0..521] OF INTEGER); 3: (HLongs:
 PACKED ARRAY[0..260] OF PupHLong); 4: (Ports:
 PACKED ARRAY[0..260] OF PupPort); END
PupEFTPAck [*EtherTypes*] const = #31
PupEFTPBort [*EtherTypes*] const = #33
PupEFTPData [*EtherTypes*] const = #30
PupEFTPEnd [*EtherTypes*] const = #32
PupEFTPSocket [*EtherTypes*] const = #20*#200000
PupError [*EtherTypes*] const = #4
PupHeader [*EtherTypes*] type = PACKED RECORD Len:
 INTEGER; Typ: 0..255; TC: 0..255; Id: PupHLong; Dst:
 PupPort; Src: PupPort END
PupHLong [*EtherTypes*] type = RECORD CASE INTEGER OF
 1: (Lng: Long); 2: (Hgh: Integer; Low: Integer); END
PupLLong [*EtherTypes*] type = RECORD CASE INTEGER OF
 1: (Lng: Long); 2: (Low: Integer; Hgh: Integer) END
PupNameSocket [*EtherTypes*] const = #4*#200000
PupPacket [*EtherTypes*] type = RECORD CASE INTEGER OF
 0: (Header: PupHeader; Data: PupData); 1: (ChkSums:
 ARRAY[0..532] OF Integer) END
PupPacketBytes [*EtherTypes*] const =
 2*WordSize(MsgPupPacket)
PupPort [*EtherTypes*] type = PACKED RECORD CASE integer
 OF 0: (Host: 0..255; Net: 0..255; Soc: PupHLong); 1: (All:
 PACKED ARRAY[0..5] OF Char) END
PushRegion [*ViewPt*] procedure(ServPort: Viewport; regionNum:
 integer; leftx: integer; topy: integer; width: integer; height:
 integer)
PutAbsoluteDef [*CFileDefs*] procedure(var BytePtr: long;
 ByteAddress: long; Name: LString)

PutAbsoluteLocalDef [*CFileDefs*] procedure(var BytePtr: long;
ByteAddress: long; Name: LString)
PutB [*Stream*] procedure(var F: Filetype)
PutC [*Stream*] procedure(var F: FileType)
PutChainHead [*CFileDefs*] procedure(var BytePtr: long;
AreaDesg: Bit8; ByteAreaOff: long)
PutLibraryDef [*CFileDefs*] procedure(var BytePtr: long;
LibraryFileName: LString; LibraryTimeStamp:
Internal_Time; LibraryLoc: long)
PutOffsetDef [*CFileDefs*] procedure(var BytePtr: long;
WdOffsetAmt: long)
PutPrimaryDef [*CFileDefs*] procedure(var BytePtr: long;
SimKind: Bit8; AreaDesg: Bit8; AreaOffset: long; SymSize:
long; SymName: LString)
PutViewportBit [*ViewPt*] procedure(ServPort: Viewport; x:
integer; y: integer; value: boolean)
PutViewportRectangle [*ViewPt*] procedure(ServPort: Viewport;
Funct: RopFunct; x: integer; y: integer; width: integer;
height: integer; Data: pVPIIntegerArray; Data_Cnt: long;
WordsAcross: integer; ux: integer; uy: integer)
pVar_CharAr [*SymDefs*] type = ^Var_CharAr
pVar_IntAr [*SymDefs*] type = ^Var_IntAr
pVarDictEntry [*SymDefs*] type = ^VarDictEntry
pVPCharArray [*SapphDefs*] type = ^VPCharArray
pVPIIntegerArray [*SapphDefs*] type = ^VPIIntegerArray
pVPPortArray [*SapphDefs*] type = ^VPPortArray
pWinNameArray [*SapphDefs*] type = ^WinNameArray
pWord_Search_Table [*CommandParse*] type = POINTER
pWord_String [*CommandParse*] type = ^Word_String
PWriteln [*Stream*] procedure(var F: Filetype)
QCodeMap [*QMapDefs*] type =
 Array[0..QMap_entries_per_block-1] of QMapInfoRecord
QCodeVersion [*SegDefs*] const = 5
QID [*AccentType*] type = 0..NUMQUEUES
QMap_Buffer [*QMapDefs*] type = array[0..PageWordSize-1] of

integer
QMap_ByteBuffer [*QMapDefs*] type = array[0..PageByteSize-1]
of bit8
QMap_entries_per_block [*QMapDefs*] const = 256 div
WSQMapInfoRecord
QMap_ptr_type [*QMapDefs*] type = record case integer of 0:
(P: POINTER); 1: (Buffer: pQMap_Buffer); 2: (ByteBuffer:
pQMap_ByteBuffer); 2: (p: pDirBlk); 2: (pDict:
pQMapDict); 3: (pMap: pQMap); 4: (pSource:
pSourceMap); end
QMapDict [*QMapDefs*] type = Record CompID: Internal_Time;
SourceFileBlock: Integer; Routines:
array[0..Max_QmapRoutines] of integer; CompID:
TimeStamp; SourceFileBlock: Integer; Routines:
array[0..252] of integer; End
QMapInfoRecord [*QMapDefs*] type = packed record
QCodeNumber: integer; SourceLineNumber: integer;
BlockNumber: integer; Offset: 0..511; SourceFileName:
0..127; end
QuitMultiProgress [*WindowUtils*] procedure(Level: integer)
QuitProgress [*WindowUtils*] procedure
Quote_Char [*SesameDefs*] const = '\''
quoted_text_bracket_char [*CommandParse*] const = """
QVerRange [*SegDefs*] type = 0..255
RaiseP [*Except*] procedure(ES, ER, PStart,PEnd: Integer)
RandomProgress [*WindowUtils*] procedure
RawIn [*KeyTran*] type = packed record case integer of 1: (region:
integer; code: 0..#177; cntrlOn: boolean; special: boolean;
escCl: 0..#7; mButtons: 0..#17); 2: (reg: integer; hashks:
integer); end
ReadBoolean [*Reader*] procedure(var F: FileType; var X:
boolean)
ReadCh [*Reader*] procedure(var F: FileType; var X: char; Field:
integer)
ReadChArray [*Reader*] procedure(var F: FileType; var X:

ChArray; Max, Len: integer)
ReadD [*PasLong*] procedure(var F: FileType; var X: long; B: integer)
ReadExtendedFile [*PathName*] function(var PathName:
Path_Name; ExtensionList: Extension_List;
ImplicitSearchList: Env_Var_Name; var Data: File_Data;
var ByteCount: long): GeneralReturn
ReadFile [*PathName*] function(var PathName: Path_Name; var
Data: File_Data; var ByteCount: long): GeneralReturn
ReadIdentifier [*Reader*] procedure(var F: FileType; var X:
integer; var IT: IdentTable; L: integer)
ReadInteger [*Reader*] procedure(var F: FileType; var X: integer)
READONLY [*AccentType*] const = 0
ReadProcessMemory [*AccInt*] function(ServPort: port; Address:
VirtualAddress; NumBytes: Long; var Data: Pointer; var
Data_Cnt: long): GeneralReturn
ReadR [*PasReal*] procedure(var F: FileType; var value: real)
ReadSegment [*AccInt*] function(ServPort: port; Segment: SegID;
Offset: Integer; NumPages: Integer; var Data: Pointer; var
Data_Cnt: long): GeneralReturn
ReadString [*Reader*] procedure(var F: FileType; var X: String;
Max, Len: integer)
READWRITE [*AccentType*] const = 1
ReadX [*Reader*] procedure(var F: FileType; var X: integer; B:
integer)
RealDivZero [*Except*] exception
RealMIndefinite [*RealFunctions*] const = Recast(#20000000001,
Real)
RealMInfinity [*RealFunctions*] const = Recast(#37740000000,
Real)
RealMLargest [*RealFunctions*] const = Recast(#37737777777,
Real)
RealMSmallest [*RealFunctions*] const = Recast(#20040000000,
Real)
RealPIndefinite [*RealFunctions*] const = Recast(#00000000001,

Real)
RealPInfinity [RealFunctions] const = Recast(#17740000000,
Real)
RealPLargest [RealFunctions] const = Recast(#17737777777,
Real)
RealPSmallest [RealFunctions] const = Recast(#00040000000,
Real)
RealWriteError [Stream] exception(FileName: SName)
Receive [AccCall] function(var xxmsg: Msg; MaxWait: long;
PortOpt: PortOption; Option: ReceiveOption):
GeneralReturn
RECEIVEIT [AccentType] const = 1
ReceiveOption [AccentType] type = PREVIEW..RECEIVEWAIT
RECEIVEWAIT [AccentType] const = 2
RecMsg [IPCRecordIO] type = record Head: Msg; RecType:
TypeType; RecName: integer; RecSize: integer; RecElts:
long; RecPtr: pointer end
RECMMSGSIZE [IPCRecordIO] const = WordSize(RecMsg)*2
RECORDIOBITS [DiskUtils] const = #140000
RecRecord [IPCRecordIO] function(var localport: Port; var
remoteport: Port; var id: long; var MsgType: long; var
recptr: Pointer; var recsize: integer): GeneralReturn
Rectangle [SapphDefs] type = record lx, ty, w, h: integer; end
RectArray [SapphDefs] type = Array[1..MaxNumRectangles] of
Rectangle
RectColor [AccCall] function(Rectangle: Port; Action: integer; X:
integer; Y: integer; Width: integer; Height: integer):
GeneralReturn
RectColorFunct [SapphDefs] type = (RectBlack, RectWhite,
RectInvert)
RectDrawLine [AccCall] function(DstRectangle: Port; Kind:
integer; X1, Y1, X2, Y2: integer): GeneralReturn
RectPutString [AccCall] function(DstRectangle: Port;
FontRectangle: Port; Action: integer; var FirstX: integer;
var FirstY: integer; StrPtr: Pointer; FirstChar: integer; var

MaxChar: integer): GeneralReturn
RectRasterOp [*AccCall*] function(DstRectangle: Port; Action:
integer; DstX: integer; DstY: integer; Width: integer;
Height: integer; SrcRectangle: Port; SrcX: integer; SrcY:
integer): GeneralReturn
RectScroll [*AccCall*] function(Rectangle: Port; X: integer; Y:
integer; Width: integer; Height: integer; Xamt: integer;
Yamt: integer): GeneralReturn
RemoveExtension [*PathName*] procedure(var fileName:
Path_Name; Extension: String)
RemoveWindow [*Sapph*] procedure(ServPort: Window)
RemoveWindowAttentionFlag [*WindowUtils*] procedure
RemoveWindowErrorFlag [*WindowUtils*] procedure
RemoveWindowRequestFlag [*WindowUtils*] procedure
ReplaceChars [*Spice_String*] procedure(var Str: PString; NewS:
PString; Index: integer)
REPLY [*AccentType*] const = 2
ReplyCode [*EtherTypes*] const = 1
ReserveCursor [*Sapph*] procedure(ServPort: Window; reserve:
boolean)
ReserveScreen [*ViewPt*] procedure(ServPort: Viewport; reserve:
boolean)
ResetError [*Stream*] exception(FileName: SName)
ResetHeap [*Dynamic*] procedure(S: HeapNumber)
ResolveSearchList [*EnvMgr*] function(ServPort: Port; Name:
Env_Var_Name; FirstOnly: boolean; var Variable:
Env_Variable; var Variable_Cnt: long; var FirstDefined:
boolean): GeneralReturn
RestoreWindow [*Sapph*] procedure(ServPort: Window)
Resume [*AccInt*] function(ServPort: port): GeneralReturn
RevPosC [*Spice_String*] function(Str: PString; c: char): integer
RevPosString [*Spice_String*] function(Source, Mask: PString):
Integer
RewriteError [*Stream*] exception(FileName: SName)
RFDELAYEDMAIN [*RunDefs*] const = 255

RFFORMAT [*RunDefs*] const = 5
RMAXFILE [*RunDefs*] const = 255
RMAXIMPORT [*RunDefs*] const = 4095
RMAXSEG [*RunDefs*] const = RFDELAYEDMAIN-1
RopFunct [*SapphDefs*] type = (RRpl, RNot, RAnd, RAndNot,
 ROr, ROrNot, RXor, RXNor)
RoundTo [*CFileDefs*] procedure(var Value: long; Boundary:
 integer)
RRecMsg [*IPCRecordIO*] type = record Head: Msg; RecType:
 TypeType; RecName: integer; RecSize: integer; RecElts:
 long; RecPtr: pointer; filler: array[1..100] of integer end
RRECMGSIZE [*IPCRecordIO*] const =
 WordSize(RRecMsg)*2
RS110 [*IODefs*] const = 1
RS1200 [*IODefs*] const = 5
RS150 [*IODefs*] const = 2
RS19200 [*IODefs*] const = 9
RS2400 [*IODefs*] const = 6
RS300 [*IODefs*] const = 3
RS4800 [*IODefs*] const = 7
RS600 [*IODefs*] const = 4
RS9600 [*IODefs*] const = 8
RSExt [*IODefs*] const = 0
RtoIOvfl [*Except*] exception
RunHead [*RunDefs*] type = record RFormat: integer; InitSeg:
 SegIndex; MainSeg: SegIndex; LinkVersion: string[20];
 LinkLocation: string[20]; LinkDate: Internal_Time;
 LinkCommand: string; Version: long; SegEntryOffset: long;
 NumSegs: SegIndex; ImplIndexOffset: long; NumImports:
 ImplIndex; FileEntryOffset: long; NumFiles: FileIndex; end
RUNHEADLOC [*RunDefs*] const = PageWordSize
SaphEmrServer [*SaphEmrServer*] function(InP, RepP:
 POINTER): boolean
Sapph_Version [*Sapph*] function(ServPort: Window): String
SapphIndex [*PascalInit*] const = 2

SapphPort [*PascalInit*] var: Port
SapphSystem [*SysType*] const = true
SaveKeyTable [*KeyTran*] function(Tab: pKeyTab; OutFileName:
Path_Name): generalReturn
Scan [*Spice_String*] function(var S: Pstring; BT: breakable; var
BRK: Pstring): Pstring
ScanEnvVariables [*EnvMgr*] function(ServPort: Port;
SearchScope: Env_Var_Scope; var EnvScanList:
Env_Scan_List; var EnvScanList_Cnt: long):
GeneralReturn
ScreenOp [*TSDefs*] type = 0..3
ScreenToVPCoords [*ViewPt*] procedure(ServPort: Viewport;
scrX: integer; scrY: integer; var x: integer; var y: integer)
Searchlist_Separator [*EnvMgrDefs*] const = ':'
SearchlistLoop [*EnvMgrDefs*] const = Env_Error_Base+4
SEarray [*RunDefs*] type = array[SegIndex] of SegEntry
SECName [*CFileDefs*] function(x: Bit8): string
SegBaseAddr [*CFileDefs*] function(Segnum: integer): long
SegBlock [*SegDefs*] type = packed record case integer of 0:
(ProgramSegment: boolean; LongIds: boolean;
DbgInfoExists: boolean; OptimizedCode: boolean;
ThirtyChIds: boolean; SegBlkFiller: 0..7; QVersion:
QVerRange; ModuleName: SNArray; FileName: FNString;
NumSeg: integer; ImportBlock: integer; GDBSize: integer;
Version: String[CommentLen]; Comment:
String[CommentLen]; Source: Language; PreLinkBlock:
integer; RoutDescBlock: integer; DiagBlock: integer;
QMapFileName: FNString; SymFileName: FNString;
ComplId: TimeStamp); 1: (OffsetRD: integer; RoutsThisSeg:
integer); 2: (Block: array[0..255] of integer) end
SegEntry [*RunDefs*] type = record ModuleName: ModString;
GDBSize: long; GDBLocation: long; SegFile: FileIndex;
SegHdrOffset: long; SegHdrSize: long; CodeOffset: long;
CodeSize: long; ImportsOffset: long; ImportsSize: long;
ProcNamesOffset: long; ProcNamesSize: long; SymsFile:

FileIndex; SymsOffset: long; SymsSize: long; QMapFile:
 FileIndex; QMapOffset: long; QMapSize: long; FirstImport:
 ImpIndex; NumImports: integer; end

SegFileType [*SegDefs*] type = file of SegBlock

SegID [*AccentType*] type = long

SegIndex [*RunDefs*] type = 0..RFDELAYEDMAIN

SegLength [*SegDefs*] const = 8

SegmentAlreadyExists [*AccentType*] const = AccErr+48

SegmentOf [*CFileDefs*] function(Addr: long): integer

Send [*AccCall*] function(var xxmsg: Msg; MaxWait: long;
 Option: SendOption): GeneralReturn

SendEtherAddress [*EtherUser*] function(Reply: Port; MaxWait:
 long; Option: SendOption): GeneralReturn

SendEtherClear [*EtherUser*] function(Typ: INTEGER; Listener:
 Port; Reply: Port; MaxWait: long; Option: SendOption):
 GeneralReturn

SendEtherFilter [*EtherUser*] function(Typ: INTEGER; Listener:
 Port; Reply: Port; MaxWait: long; Option: SendOption):
 GeneralReturn

SendEtherPacket [*EtherUser*] function(Packet: POINTER;
 PacketWords: INTEGER; MaxWait: long; Option:
 SendOption): GeneralReturn

SendOption [*AccentType*] type = WAIT..REPLY

SendPupClear [*EtherUser*] function(Socket: Long; Listener:
 Port; Reply: Port; MaxWait: long; Option: SendOption):
 GeneralReturn

SendPupFilter [*EtherUser*] function(Socket: Long; Listener:
 Port; Reply: Port; MaxWait: long; Option: SendOption):
 GeneralReturn

SendPupPacket [*EtherUser*] function(PupPacket: POINTER;
 MaxWait: long; Option: SendOption): GeneralReturn

SendRecord [*IPCRecordIO*] function(localport: Port; remoteport:
 Port; id: long; MsgType: long; recptr: Pointer; recsize:
 integer): GeneralReturn

ServerFull [*IO*] exception(TransactionID: long)

ServerIOPort [*IODefs*] type = port
ServerNamePort [*IODefs*] type = port
Sesame_Error_Base [*SesameDefs*] const = 1200
Sesame_Version [*Sesame*] function(ServPort: Port; var
 VersionStr: string): GeneralReturn
SesameIndex [*PascalInit*] const = 1
SesAuthServerPort [*SesDisk*] function(ServPort: Port;
 AuthServerPort: Port): GeneralReturn
SesConnect [*SesDisk*] function(ServPort: Port; RegPort: Port):
 GeneralReturn
SesDirectio [*SesDisk*] function(ServPort: Port; var CmdBlk:
 DirectIOArgs; var DataHdr: Header; var Data: DiskBuffer):
 GeneralReturn
SesDisk_Error_Base [*SesDiskDefs*] const = 29400
SesDisk_Version [*SesDisk*] function(ServPort: Port; var
 VerString: string): GeneralReturn
SesDiskDisMount [*SesDisk*] function(ServPort: Port; PartName:
 string): GeneralReturn
SesDiskMount [*SesDisk*] function(ServPort: Port; PartName:
 string): GeneralReturn
SesEnterForeignSesamoid [*SesDisk*] function(ServPort: Port;
 APathName: APath_Name; ForeignPort: Port;
 ForeignPrefix: APath_Name): GeneralReturn
SesEnterSegID [*SesDisk*] function(ServPort: Port; var
 APathName: APath_Name; SegmentID: SegID):
 GeneralReturn
SesGetAccessRights [*SesDisk*] function(ServPort: Port; var
 APathName: APath_Name; var Owner: User_ID; var
 Rights: Access_Rights): GeneralReturn
SesGetDefaultAccess [*SesDisk*] function(ServPort: Port; var
 DefaultAccess: Access_Rights): GeneralReturn
SesGetDiskPartitions [*SesDisk*] function(ServPort: Port; var
 PartL: ptrPartDList; var NumElts: integer): GeneralReturn
SesGetFileHeader [*Sesame*] function(ServPort: Port;
 APathName: APath_Name; var FileHeader: File_Header):

GeneralReturn

SesGetNetPort [*SesDisk*] function(ServPort: Port; var
SesNetPort: Port): GeneralReturn

SesGetSegID [*SesDisk*] function(ServPort: Port; var APathName:
APath_Name; var SegmentID: SegID): GeneralReturn

SesGetSegName [*SesDisk*] function(ServPort: Port; SegmentID:
SegID; var APathName: APATH_Name): GeneralReturn

SesLookupForeignSesamoid [*SesDisk*] function(ServPort: Port;
APATH_Name: APATH_Name; var ForeignPort: Port; var
ForeignPrefix: APATH_Name): GeneralReturn

SesMountDevice [*SesDisk*] function(ServPort: Port; interface:
DiskInterface; log_unit: InterfaceInfo; var unitnum: integer;
var PartL: ptrPartDList; var NumEltS: integer):
GeneralReturn

SesMsgServerPort [*SesDisk*] function(ServPort: Port;
MsgServerPort: Port): GeneralReturn

SesPort [*PascalInit*] var: port

SesReadBoth [*Sesame*] function(ServPort: Port; var APATHName:
APATH_Name; var Data: File_Data; var Data_Cnt: long; var
FileHeader: File_Header; var NameStatus: Name_Status):
GeneralReturn

SesReadFile [*Sesame*] function(ServPort: Port; var APATHName:
APATH_Name; var Data: File_Data; var Data_Cnt: long; var
DataFormat: Data_Format; var CreationDate:
Internal_Time; var NameStatus: Name_Status):
GeneralReturn

SesScanNames [*Sesame*] function(ServPort: Port;
WildAPATHName: Wild_APATH_Name; NameFlags:
Name_Flags; EntryType: Entry_Type; var DirectoryName:
APATH_Name; var EntryList: Entry_List; var
EntryList_Cnt: long): GeneralReturn

SesSetAccessRights [*SesDisk*] function(ServPort: Port; var
APATHName: APATH_Name; NewOwner: User_ID;
NewRights: Access_Rights): GeneralReturn

SesSetDefaultAccess [*SesDisk*] function(ServPort: Port;

NewDefaultAccess: Access_Rights): GeneralReturn
SesSetPOSWriteDate [*SesDisk*] function(ServPort: Port;
APathName: APath_Name; WriteDate: Internal_Time):
GeneralReturn
SetAsmLong [*CFileDefs*] procedure(var ByteAddr: long;
NewValue: long)
SetAsmString [*CFileDefs*] procedure(var ByteAddr: long;
NewValue: LString)
SetBackLog [*AccInt*] function(ServPort: port; LocalPort: port;
BackLog: Integer): GeneralReturn
SetBreak [*Spice_String*] procedure(BT: BreakTable; Break,
Omit: PString; Options: BreakKind)
SetCodeByte [*CFileDefs*] procedure(var ByteAddr: long;
NewValue: Bit8)
SetCodeLong [*CFileDefs*] procedure(var ByteAddr: long;
NewValue: long)
SetCodeWord [*CFileDefs*] procedure(var ByteAddr: long;
NewValue: integer)
SetCursor [*ViewPt*] procedure(ServPort: CursorSet; cursIndex:
integer; pCursorData: Pattern; xOffset: integer; yOffset:
integer)
SetCursorPos [*ViewPt*] procedure(ServPort: Viewport; x: integer;
y: integer)
SetDataLong [*CFileDefs*] procedure(var ByteAddr: long;
NewValue: long)
SetDateTime [*Time*] procedure(ServPort: port; ITime:
Internal_Time)
SetDebugPort [*AccInt*] function(ServPort: port; DebugPort: port):
GeneralReturn
SetEnvVariable [*EnvMgr*] function(ServPort: Port; Name:
Env_Var_Name; VarType: Env_Var_Type; VarScope:
Env_Var_Scope; Variable: Env_Variable; Variable_Cnt:
long): GeneralReturn
SetFontChar [*ViewPt*] procedure(ServPort: Viewport;
OneCharData: pVPIntegerArray; OneCharData_Cnt: long);

WordsAcross: integer; ch: char; CharWidth: integer)

SetKernelWindow [*AccInt*] function(ServPort: port; LeftX:
Integer; TopY: Integer; Width: Integer; Height: Integer;
Inverted: Boolean): GeneralReturn

SetLimit [*AccInt*] function(ServPort: port; ReplyPort: port; Limit:
Long): GeneralReturn

SetPagingSegment [*AccInt*] function(ServPort: port; Segment:
SegID): GeneralReturn

SetPMPort [*Sapph*] procedure(ServPort: Window; ProcCtlPort:
Port)

SetPortsWaiting [*AccCall*] function(var ports: PortBitArray):
GeneralReturn

SetPriority [*AccInt*] function(ServPort: port; Priority: PriorID):
GeneralReturn

SetProtection [*AccInt*] function(ServPort: port; Address:
VirtualAddress; NumBytes: Long; Protection: Integer):
GeneralReturn

SetRegionCursor [*ViewPt*] procedure(ServPort: Viewport;
regionNum: integer; cursorImage: CursorSet; cursIndex:
integer; cursFunc: CursorFunction; track: boolean)

SetRegionParms [*ViewPt*] procedure(ServPort: Viewport;
regionNum: integer; absolute: boolean; speed: integer;
minx: integer; maxx: integer; miny: integer; maxy: integer;
modx: integer; posx: integer; mody: integer; posy: integer)

SetScreenColor [*ViewPt*] procedure(ServPort: Viewport; invert:
boolean)

SetSystemZone [*Time*] procedure(ServPort: port; TimeZone:
integer; DSTWhenTimely: boolean)

SetTempSegPartition [*AccInt*] function(ServPort: port;
PartName: DevPartString): GeneralReturn

SetWindowAttention [*Sapph*] procedure(ServPort: Window; attn:
boolean)

SetWindowError [*Sapph*] procedure(ServPort: Window; error:
boolean)

SetWindowName [*Sapph*] procedure(ServPort: Window; var

progName: ProgStr)
SetWindowProgress [*Sapph*] procedure(ServPort: Window;
nestLevel: integer; value: Long; max: Long)
SetWindowRequest [*Sapph*] procedure(ServPort: Window;
requesting: boolean)
SetWindowTitle [*Sapph*] procedure(ServPort: Window; title:
TitStr)
shell_input_redirect [*CommandParse*] const = '<'
shell_output_redirect [*CommandParse*] const = '>'
shell_sequential_command [*CommandParse*] const = ','
shell_special_args_start [*CommandParse*] const = '['
shell_special_args_stop [*CommandParse*] const = ']'
ShowBreak [*Spice_String*] function(BT: BreakTable): PString
ShowPathAndTitle [*WindowUtils*] procedure(s: TitStr)
ShowRun [*ALoad*] procedure(p: pointer; MapFileName:
Path_Name)
ShowWindowAttentionFlag [*WindowUtils*] procedure
ShowWindowErrorFlag [*WindowUtils*] procedure
ShowWindowRequestFlag [*WindowUtils*] procedure
ShrinkWindow [*Sapph*] procedure(ServPort: Window)
SigDebug [*ProcMgrDefs*] const = MinSignal+36
SigLevel1Abort [*ProcMgrDefs*] const = MinSignal+37
SigLevel2Abort [*ProcMgrDefs*] const = MinSignal+38
SigLevel3Abort [*ProcMgrDefs*] const = MinSignal+39
SignalAction [*ProcMgrDefs*] type = (SigDefault, SigIgnore,
SigSend)
SignalBase [*ProcMgrDefs*] const = 3800
SignalMsg [*ProcMgrDefs*] type = record Head: Msg;
tCtlWindow: TypeType; CtlWindow: Window; tSignal:
TypeType; Signal: SignalName; end
SignalMsgID [*ProcMgrDefs*] const = 3801
SignalName [*ProcMgrDefs*] type = integer
SigResume [*ProcMgrDefs*] const = MinSignal+34
SigStatus [*ProcMgrDefs*] const = MinSignal+35
SigSuspend [*ProcMgrDefs*] const = MinSignal+33

SimpleName [*PathName*] function(*PathName*: Path_Name);
 Entry_Name

SimpleNameSize [*IFileDefs*] const = 25

Sin [*RealFunctions*] function(X: Real): Real

single_char_quote [*CommandParse*] const = '\'

SinH [*RealFunctions*] function(x: real): real

SinHLarge [*RealFunctions*] exception(X: Real)

SinLarge [*RealFunctions*] exception(X: Real)

SIOSenseStatus [*IODefs*] type = record RxCharAvailable:
 boolean; IntPending: boolean; TxBufferEmpty: boolean;
 DCD: boolean; SyncHunt: boolean; CTS: boolean;
 TransmitUnderRun: boolean; BreakAbort: boolean; AllSent:
 boolean; Residue: Bit3; ParityError: boolean; RxOverRun:
 boolean; CrcFramingError: boolean; EndOfFrame: boolean;
 end

SIOWriteRegister [*IODefs*] type = packed record case RegNum:
 Bit8 of 0: (); 1: (RegVal: Bit8) end

SmallReal [*Stream*] exception(FileName: SName)

SName [*Stream*] type = string[255]

SNArray [*SegDefs*] type = packed array[1..SegLength] of Char

SoftEnable [*AccCall*] function(NormOrEmerg: boolean; EnOrDis:
 boolean): GeneralReturn

SoftInterrupt [*AccInt*] function(ServPort: port; NormOrEmerg:
 Boolean; var EnOrDisable: Boolean): GeneralReturn

Source_entries_per_block [*QMapDefs*] const = 256 div
 WSSourceMapRecord

SourceMap [*QMapDefs*] type =
 Array[0..Source_entries_per_block-1] of
 SourceMapRecord

SourceMapRecord [*QMapDefs*] type = record FileName:
 APath_Name; CompID: Internal_Time; Filler:
 array[0..124] of integer; FileName: PathName; fileBlocks,
 fileBits: Integer; end

Spawn [*Spawn*] function(VAR ChildKPort: Port; VAR
 ChildDPort: Port; ProgName: APath_Name; ProcName:

STRING; HisCommand: CommandBlock; DebugIt:
BOOLEAN; ProtectChild: BOOLEAN; SapphConn:
ConnectionInheritance; pWindow: Port; pTypeScript: Port;
EMConn: ConnectionInheritance; pEMPort: Port;
PassedPorts: ptrPortArray; NPorts: LONG; LoaderDebug:
BOOLEAN); GeneralReturn

SpiceSegKind [*AccentType*] type = (Temporary, Permanent, Bad,
SegPhysical, Imaginary, Shadow)

Split [*Spawn*] function(VAR ChildKPort: Port; VAR ChildDPort:
Port): GeneralReturn

Sqrt [*RealFunctions*] function(X: Real): Real

SqrtNeg [*RealFunctions*] exception(X: Real)

Squeeze [*Spice_String*] function(Str: PString): PString

StatArray [*ProcMgrDefs*] type = array[0..0] of StatRecord

StatList [*ProcMgrDefs*] type = ^StatArray

StatRecord [*ProcMgrDefs*] type = record RunTime: long;
LoadTime: long; ElapsedTime: long; KernelPort: long;
Priority: integer; QueueID: integer; ProcName: string;
IconName: ProgStr; State: ProcState; end

Status [*AccInt*] function(ServPort: port; var NStats: PStatus):
GeneralReturn

str [*Spice_String*] function(Ch: char): PString

StrBadParm [*Spice_String*] exception(FuncName,
StringArgument: PString; ParmValue: integer)

StreamBuffer [*Stream*] type = record case integer of 0: (W:
array[0..255] of integer); 1: (B1: packed array[0..0] of
0..1); 2: (B2: packed array[0..0] of 0..3); 3: (B3: packed
array[0..0] of 0..7); 4: (B4: packed array[0..0] of 0..15); 5:
(B5: packed array[0..0] of 0..31); 6: (B6: packed
array[0..0] of 0..63); 7: (B7: packed array[0..0] of 0..127);
8: (B8: packed array[0..0] of 0..255); 9: (C: packed
array[0..255] of char); end

StreamClose [*Stream*] procedure(var F: FileType)

StreamF [*Stream*] type = record FName: SName; Buffer:
pStreamBuffer; BufSize: long; WordIndex: long; LastElt:

integer; EltIndex: integer; end

StreamFlushOutput [*Stream*] procedure(var F: Text)

StreamInit [*Stream*] procedure(var F: FileType; WordSize,
BitSize: integer; CharFile: boolean)

StreamKeyBoardReset [*Stream*] procedure(var F: Text)

StreamName [*Stream*] function(var F: FileType): SName

StreamOpen [*Stream*] procedure(var F: FileType; var Name:
SName; WordSize, BitSize: integer; CharFile: boolean;
OpenWrite: boolean)

StreamProgress [*WindowUtils*] procedure(var F: File)

StreamVersion [*Stream*] const = '3.9'

StrIndx [*Except*] exception

String_255 [*TimeDefs*] type = string[255]

Strip [*Spice_String*] function(Str: PString): PString

StripCurrent [*PathName*] function(var WildPathName:
Wild_Path_Name): GeneralReturn

StrLong [*Except*] exception

STSChangeEnv [*TS*] procedure(ServPort: Port; env: port)

STSChangeKeyTran [*TS*] function(ServPort: Port; keytrantab:
pKeyTab; tablesiz: long): GeneralReturn

STSCursorOp [*TS*] function(ServPort: Port; op: CursorOp; count:
Integer; var charpos: Integer; var linenum: Integer):
GeneralReturn

STSCursorPos [*TS*] function(ServPort: Port; var charpos: Integer;
var linenum: Integer; var x: Integer; var y: Integer):
GeneralReturn

STSDeleteOp [*TS*] function(ServPort: Port; op: DeleteOp; count:
Integer; var charpos: Integer; var linenum: Integer):
GeneralReturn

STSFlushInput [*TS*] procedure(ServPort: Port)

STSFlushOutput [*TS*] procedure(ServPort: Port)

STSFullLine [*TS*] function(ServPort: Port): boolean

STSFullOpen [*TS*] function(ServPort: Port; vp: Viewport; env:
port; fontName: TString255; doWrap: boolean; dispPages:
Integer): Typescript

STSFullOpenWindow [TS] function(ServPort: Port; w: Window;
env: port; fontName: TString255; doWrap: boolean;
dispPages: Integer): Typescript

STSGetChar [TS] function(ServPort: Port): char

STSGetString [TS] function(ServPort: Port): TString255

STSGiveKey [TS] function(ServPort: Port; untrankey: EventRec;
var fullline: boolean): GeneralReturn

STSGrabWindow [TS] function(ServPort: Port; kPort: port);
Window

STSInputStatus [TS] procedure(ServPort: Port; var empty:
boolean; var full: boolean)

STSMMoreMode [TS] function(ServPort: Port; var on: boolean):
GeneralReturn

STSMoveCursor [TS] function(ServPort: Port; var charpos:
Integer; var linenum: Integer): GeneralReturn

STSOpen [TS] function(ServPort: Port; vp: Viewport; env: port);
Typescript

STSOpenTerm [TS] function(ServPort: Port; vp: Viewport; w:
port; env: port; fontname: TString255; dispPages: Integer;
keytrantab: pKeyTab; tablesize: long): Typescript

STSOpenWindow [TS] function(ServPort: Port; w: Window; env:
port): Typescript

STSPutChar [TS] procedure(ServPort: Port; ch: char)

STSPutCharArray [TS] procedure(ServPort: Port; chars:
pTSCharArray; char_count: long; firstCh: Integer; lastCh:
Integer)

STSPutString [TS] procedure(ServPort: Port; s: TString255)

STSScreenOp [TS] function(ServPort: Port; op: ScreenOp;
topline: Integer; bottomline: Integer; count: Integer):
GeneralReturn

STSSetInput [TS] function(ServPort: Port; inputstr: TString255);
GeneralReturn

STSVpSize [TS] procedure(ServPort: Port; var charwidth: Integer;
var lines: Integer; var pixwidth: Integer; var pixheight:
Integer)

SubDeleteName [*Sesame*] function(ServPort: Port; APathName:
APath_Name): GeneralReturn

SubEnterName [*Sesame*] function(ServPort: Port; var
APathName: APath_Name; EntryType: Entry_Type;
EntryData: Entry_Data): GeneralReturn

SubLookUpName [*Sesame*] function(ServPort: Port; var
APathName: APath_Name; var EntryType: Entry_Type;
var EntryData: Entry_Data; var NameStatus: Name_Status):
GeneralReturn

SubReadFile [*Sesame*] function(ServPort: Port; APathName:
APath_Name; var Data: File_Data; var Data_Cnt: long):
GeneralReturn

SubReName [*Sesame*] function(ServPort: Port; OldAPathName:
APath_Name; var NewAPathName: APath_Name):
GeneralReturn

SubstrFor [*Spice_String*] function(Source: PString; Index, Size:
Integer): PString

SubstrTo [*Spice_String*] function(Source: PString; Index,
EndIndex: Integer): PString

SubTestName [*Sesame*] function(ServPort: Port; var APathName:
APath_Name; var EntryType: Entry_Type; var
NameStatus: Name_Status): GeneralReturn

SubWriteFile [*Sesame*] function(ServPort: Port; var APathName:
APath_Name; Data: File_Data; Data_Cnt: long;
DataFormat: Data_Format; var CreationDate:
Internal_Time): GeneralReturn

Success [*AccentType*] const = AccErr+1

Suspend [*AccInt*] function(ServPort: port): GeneralReturn

SwapFile [*IFileDefs*] const = 17

switch_leadin_char [*CommandParse*] const = '-'

SyncDisk [*AccInt*] function(ServPort: port; on: Boolean):
GeneralReturn

SyncIO [*IO*] function(IOPort: ServerIOPort; Command:
IOCommand; CmdBlk: pCmdBlk; CmdBlk_Cnt: long; var
DataBuf: pDataBuf; var DataBuf_Cnt: long);

DataTransferCnt: long; TimeOut_Arg: long; var Status:
 IOStatusBlk): GeneralReturn
SysFontHeight [*SapphDefs*] const = 13
SysFontName [*SapphDefs*] const = 'Fix13.Kst'
SysFontWidth [*SapphDefs*] const = 9
T_IntToString [*Time*] function(ServPort: port; ITime:
 Internal_Time; TimeFormat: integer): String
T_IntToUser [*Time*] function(ServPort: port; ITime:
 Internal_Time): User_Time
T_IntToZone [*Time*] function(ServPort: port; ITime:
 Internal_Time; TZone: Zone_Info): User_Time
T_Never [*Time*] function(ServPort: port): Internal_Time
T_StringToInt [*Time*] function(ServPort: port; STime:
 String_255; var Index: integer; var WhatIFound: integer):
 Internal_Time
T_StringToUser [*Time*] function(ServPort: port; STime:
 String_255; var Index: integer; var WhatIFound: integer):
 User_Time
T_UserToInt [*Time*] function(ServPort: port; UTime:
 User_Time): Internal_Time
T_UserToString [*Time*] function(ServPort: port; UTime:
 User_Time; TimeFormat: integer): String
Tan [*RealFunctions*] function(X: Real): Real
TanH [*RealFunctions*] function(x: real): real
TanLarge [*RealFunctions*] exception(X: Real)
Terminate [*AccInt*] function(ServPort: port; Reason: Long):
 GeneralReturn
TextState [*CFileDefs*] const = 0
TF_12_Hour [*TimeDefs*] const = #004000
TF_ANSI [*TimeDefs*] const = #000300
TF_ANSI_Ordinal [*TimeDefs*] const = #000340
TF_BankPad [*TimeDefs*] const = #020000
TF_Dashes [*TimeDefs*] const = #000000
TFDateFormat [*TimeDefs*] const = #000340
TF_FullMonth [*TimeDefs*] const = #000010

TF_FullWeekday [*TimeDefs*] const = #000002
TF_FullYear [*TimeDefs*] const = #000020
TF_Milliseconds [*TimeDefs*] const = #002000
TF_Never [*TimeDefs*] const = #100000
TF_NoColumns [*TimeDefs*] const = #040000
TF_NoDate [*TimeDefs*] const = #000004
TF_NoSeconds [*TimeDefs*] const = #001000
TF_NoTime [*TimeDefs*] const = #000400
TF_Reversed [*TimeDefs*] const = #000100
TF_Slashes [*TimeDefs*] const = #000140
TF_Spaces [*TimeDefs*] const = #000040
TF_TimeZone [*TimeDefs*] const = #010000
TF_Weekday [*TimeDefs*] const = #000001
This_Directory [*SesameDefs*] const = '.'/
Time_Fields [*TimeDefs*] type = packed record Hour: 0..24;
 Minute: 0..59; Second: 0..59; Millisecond: 0..999; end
TimeIndex [*PascalInit*] const = 0
TimeNotInitialized [*Time*] exception
TimeOut [*AccentType*] const = AccErr+2
TimeOutError [*Stream*] exception(FileName: SName)
TimePort [*PascalInit*] var: port
TimeStamp [*OldTimeStamp*] type = packed record Hour: 0..23;
 Day: 1..31; Second: 0..59; Minute: 0..59; Month: 1..12;
 Year: 0..63; end
TitleOverhead [*SapphDefs*] const = SysFontHeight+6
TitStr [*SapphDefs*] type = String[TitStrLength]
TitStrLength [*SapphDefs*] const = LandScapeBitWidth div
 SysFontWidth
TooManyHeaps [*Dynamic*] exception
TooManyReplies [*AccentType*] const = AccErr+5
Touch [*AccInt*] function(ServPort: port; Address:
 VirtualAddress): GeneralReturn
TP_Date [*TimeDefs*] const = #000002
TP_Never [*TimeDefs*] const = #000020
TP_RESERVED [*TimeDefs*] const = #177740

TP_Time [*TimeDefs*] const = #000004
TP_Weekday [*TimeDefs*] const = #000001
TP_Zone [*TimeDefs*] const = #000010
TraceHeap [*Dynamic*] procedure(S: HeapNumber; Trace:
 boolean)
TranOut [*KeyTran*] type = record cmd: 0..255; ch: char; escTy:
 escKind; end
TranslateKey [*KeyTran*] function(kt: pKeyTab; e: EventRec; var
 k: KeyEvent): GeneralReturn
TransMicro [*ControlStore*] type = packed record case integer of
 0: (Word1: integer; Word2: integer; Word3: integer); 1:
 (ALU23: 0..3; ALU0: 0..1; W: 0..1; ALU1: 0..1; A: 0..7;
 Z: 0..255; SFF: 0..63; H: 0..1; B: 0..1; JmpCnd: 0..255)
 end
TrapCodes [*AccentType*] type = (TrapInit, TrapReadFault,
 TrapWriteFault, TrapSend, TrapReceive,
 TrapSetPortsWaiting, TrapPortsWithMessages,
 TrapDebugWrite, TrapException, TrapNothing,
 TrapRectDrawLine, TrapRectRasterOp, TrapCharRead,
 TrapFull, TrapFlush, TrapMoveWords, TrapRectPutString,
 TrapError, TrapClockEnable, TrapGPRead, TrapGPWrite,
 TrapSoftEnable, TrapGetIOSleepID, TrapRectColor,
 TrapRectScroll, TrapLockPorts, TrapMessagesWaiting)
Trim [*Spice_String*] function(Str: PString): PString
TruncateSegment [*AccInt*] function(ServPort: port; Segment:
 SegID; NewSize: Integer): GeneralReturn
ts_bell [*TSDefs*] const = 0
ts_boln [*TSDefs*] const = 1
ts_clear [*TSDefs*] const = 1
ts_delchar [*TSDefs*] const = 0
ts_down [*TSDefs*] const = 4
ts_eoln [*TSDefs*] const = 2
ts_erboln [*TSDefs*] const = 2
ts_ereoln [*TSDefs*] const = 1
ts_home [*TSDefs*] const = 0

ts_left [TSDefs] const = 5
ts_redisplay [TSDefs] const = 2
ts_right [TSDefs] const = 6
ts_scroll [TSDefs] const = 3
ts_up [TSDefs] const = 3
TSAsynch [TS] function(InP: Pointer): boolean
TSCharArray [TSDefs] type = packed array[0..0] of char
TString255 [TSDefs] type = string[255]
TYPEBIT [AccentType] const = 0
TYPEBOOLEAN [AccentType] const = 0
TYPEBYTE [AccentType] const = 9
TYPECHAR [AccentType] const = 8
TYPEINT16 [AccentType] const = 1
TYPEINT32 [AccentType] const = 2
TYPEINT8 [AccentType] const = 9
TYPEPAGE [AccentType] const = 14
TYPEPSTAT [AccentType] const = 11
TYPEPT [AccentType] const = 6
TYPEPTALL [AccentType] const = 5
TYPEPTOWNERSHIP [AccentType] const = 3
TYPEPTRECEIVE [AccentType] const = 4
TYPEREAL [AccentType] const = 10
TypeScript [TSDefs] type = port
TypeScript_Version [TS] function(ServPort: Port): TString255
TypeScriptIndex [PascalInit] const = 3
TypeScriptPort [PascalInit] var: Port
TYPESEGID [AccentType] const = 13
TYPESTRING [AccentType] const = 12
TypeType [AccentType] type = packed record case integer of 1:
 (TypeName: Bit8; TypeSizeInBits: Bit8; NumObjects:
 Bit12; InLine: boolean; LongForm: boolean; Deallocate:
 boolean); 2: (LongInteger: long) end
TYPEUNSTRUCTURED [AccentType] const = 0
ULInitial [*Spice_String*] function(Str1, Str2: PString): boolean
ULPosString [*Spice_String*] function(Source, Mask: PString):

Integer
UncaughtException [*AccentType*] const = AccErr+44
UNCHANGED [*SapphDefs*] const = -32001
UndefinedGlobal [*CFileDefs*] const = UndefinedSymbol
UndefinedSymbol [*CFileDefs*] const = 7
UndfDevice [*Stream*] exception
UndfInt [*Except*] exception
UndfQcd [*Except*] exception
UndReal [*Except*] exception
UniqueWordIndex [*CommandParse*] function(table:
 pWord_Search_Table; ptrWordString: pWord_String; var
 WordText: Cmnd_String): integer
UnitIOError [*Stream*] exception(FileName: SName)
Unknown_User [*AuthDefs*] const = 1023
UnknownAction [*ProcMgrDefs*] const = ProcMgrBase+3
UnknownFile [*IFileDefs*] const = 0
UnknownPort [*ProcMgrDefs*] const = ProcMgrBase+7
UnknownProcess [*ProcMgrDefs*] const = ProcMgrBase+1
UnknownSignal [*ProcMgrDefs*] const = ProcMgrBase+2
UnknownWindow [*ProcMgrDefs*] const = ProcMgrBase+4
UnMapAddr [*DiskUtils*] function(Disk: integer; PhyAddr:
 DiskAddress): DiskAddress
UnrecognizedMsgType [*AccentType*] const = AccErr+15
UNSPECEXCEPTION [*AccentType*] const = 5
Up_one_Directory [*SesameDefs*] const = '.. /'
UpChar [*Spice_String*] function(C: Char): Char
UpEQU [*Spice_String*] function(Str1: PString; Str2: PString):
 boolean
User_ID [*AuthDefs*] type = No_User..Max_Users
User_Time [*TimeDefs*] type = packed record Date: Date_Fields;
 Time: Time_Fields; Zone: Zone_Info; end
UserCommand [*PascalInit*] var: CommandBlock
UserError [*ViewPt*] exception(s: String)
UserEventPort [*IODefs*] type = port
UserNameNotFound [*AuthDefs*] const = Auth_Error_Base+1

UserRecord [*AuthDefs*] type = record Name: Auth_Var; UserID: User_ID; EncryptPass: PassType; Profile: APath_Name; NameOfShell: APath_Name; End

UserTypescript [*PascalInit*] var: Port

UserWindow [*PascalInit*] var: Port

UserWindowShared [*PascalInit*] var: Boolean

Valid_Name_Chars [*SesameDefs*] const = '\$.-.+0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'

ValidateMemory [*AccInt*] function(ServPort: port; var Address: VirtualAddress; NumBytes: Long; CreateMask: Long): GeneralReturn

value_marker_char [*CommandParse*] const = '='

Var_CharAr [*SymDefs*] type = Packed Array[0..511] of Char

Var_IntAr [*SymDefs*] type = Array[0..255] of Integer

var_ptr [*SymDefs*] type = Record Case Integer Of 0: (P: POINTER); 1: (p: pDirBlk); 2: (dict: pVarDictEntry); 3: (chars: pVar_CharAr); 4: (int: pVar_IntAr); End

VarDescriptor [*SymDefs*] type = Packed Record mainType: BaseType; subType: BaseType; etc: 0..255; End

VarDictEntry [*SymDefs*] type = Record CompID: Internal_Time; Globals: Integer; Routines: Array[0..Max_SymRoutines] of Integer; End

VarNameArray [*SymDefs*] type = Record TotNumChars: integer; VarNames: Packed Array[0..0] of Char; End

Ver_Separator [*SesameDefs*] const = '#'

ViewChar [*ViewPt*] procedure(ServPort: Viewport; fontVP: Viewport; funct: RopFunct; var dx: integer; var dy: integer; ch: char)

ViewChArray [*ViewPt*] procedure(ServPort: Viewport; fontVP: Viewport; funct: RopFunct; var dx: integer; var dy: integer; chars: pVPCharArray; chars_Cnt: long; firstCh: integer; var lastch: integer)

ViewColorRect [*ViewPt*] procedure(ServPort: Viewport; funct: RectColorFunct; x: integer; y: integer; width: integer;

height: integer)

ViewLine [*ViewPt*] procedure(ServPort: Viewport; funct:
LineFunct; x1: integer; y1: integer; x2: integer; y2: integer)

Viewport [*SapphDefs*] type = Port

ViewportState [*ViewPt*] procedure(ServPort: Viewport; var curlx:
integer; var curty: integer; var curwidth: integer; var
curheight: integer; var curRank: integer; var memory:
boolean; var courteous: boolean; var transparent: boolean)

ViewPutChar [*ViewPt*] procedure(ServPort: Viewport; fontVP:
Viewport; funct: RopFunct; dx: integer; dy: integer; ch:
char)

ViewPutChArray [*ViewPt*] procedure(ServPort: Viewport;
fontVP: Viewport; funct: RopFunct; dx: integer; dy: integer;
chars: pVPCharArray; chars_Cnt: long; firstCh: integer;
lastch: integer)

ViewPutString [*ViewPt*] procedure(ServPort: Viewport; fontVP:
Viewport; funct: RopFunct; dx: integer; dy: integer; str:
VPStr255; firstCh: integer; lastch: integer)

ViewROP [*ViewPt*] procedure(ServPort: Viewport; funct:
RopFunct; dx: integer; dy: integer; width: integer; height:
integer; srcVP: Viewport; sx: integer; sy: integer)

ViewScroll [*ViewPt*] procedure(ServPort: Viewport; x: integer; y:
integer; width: integer; height: integer; Xamt: integer;
Yamt: integer)

ViewString [*ViewPt*] procedure(ServPort: Viewport; fontVP:
Viewport; funct: RopFunct; var dx: integer; var dy: integer;
str: VPStr255; firstCh: integer; var lastch: integer)

VirtualAddress [*AccentType*] type = long

VPChar [*ViewKern*] procedure(destvp, fontVP: Viewport; funct:
RopFunct; var dx, dy: Integer; ch: Char)

VPCharArray [*SapphDefs*] type = Packed Array[0..1] of Char

VPChArray [*ViewKern*] procedure(destvp, fontVP: Viewport;
funct: RopFunct; var dx, dy: Integer; chars: pVPCharArray;
arSize: Long; firstCh: Integer; var lastch: Integer)

VPColRect [*ViewKern*] procedure(vp: Viewport; funct:

RectColorFunct; x, y, width, height: Integer)
VPExclusionFailure [*AccentType*] const = AccErr+40
VPIntegerArray [*SapphDefs*] type = Array[0..0] of Integer
VPLine [*ViewKern*] procedure(destvp: Viewport; funct:
 LineFunct; x1, y1, x2, y2: Integer)
VPNIY [*ViewPt*] exception
VPPortArray [*SapphDefs*] type = Record num: Integer; ar:
 Array[0..0] of Port; end
VPPutChar [*ViewKern*] procedure(destvp, fontVP: Viewport;
 funct: RopFunct; dx: Integer; dy: Integer; ch: Char)
VPPutChArray [*ViewKern*] procedure(destvp, fontVP: Viewport;
 funct: RopFunct; dx, dy: Integer; chars: pVPCharArray;
 arSize: Long; firstCh, lastch: Integer)
VPPutString [*ViewKern*] procedure(destvp, fontVP: Viewport;
 funct: RopFunct; dx, dy: Integer; var str: VPStr255; firstCh,
 lastch: Integer)
VPREGION [*SapphDefs*] const = 1
VPROP [*ViewKern*] procedure(destvp: Viewport; funct:
 RopFunct; dx, dy, width, height: Integer; srcVP: Viewport;
 sx, sy: Integer)
VPSscroll [*ViewKern*] procedure(destvp: Viewport; x, y, width,
 height, Xamt, Yamt: Integer)
VPStr255 [*SapphDefs*] type = string[255]
VPString [*ViewKern*] procedure(destvp, fontVP: Viewport; funct:
 RopFunct; var dx, dy: Integer; var str: VPStr255; firstCh:
 Integer; var lastch: Integer)
VPtoScreenCoords [*ViewPt*] procedure(ServPort: Viewport; x:
 integer; y: integer; var scrX: integer; var scrY: integer)
WAIT [*AccentType*] const = 0
WaitEtherAddress [*EtherUser*] function: INTEGER
WaitEtherClear [*EtherUser*] function(Typ: INTEGER; Listener:
 Port): BOOLEAN
WaitEtherFilter [*EtherUser*] function(Typ: INTEGER; Listener:
 Port): BOOLEAN
WaitPupClear [*EtherUser*] function(VAR Socket: Long;

Listener: Port): BOOLEAN
WaitPupFilter [*EtherUser*] function(VAR Socket: Long;
Listener: Port): BOOLEAN
WCSSizeError [*ControlStore*] exception
Wild_APath_Name [*SesameDefs*] type =
string[Path_Name_Size]
Wild_Path_Name [*PathName*] type = string[Path_Name_Size]
WILDREGION [*KeyTranDefs*] const = 31
WillReply [*AccentType*] const = AccErr+4
Window [*SapphDefs*] type = Port
WindowInUse [*ProcMgrDefs*] const = ProcMgrBase+5
WindowViewport [*Sapph*] procedure(ServPort: Window; var vp:
Viewport; var vpWidth: integer; var vpHeight: integer)
WinForName [*Sapph*] function(ServPort: Window; name:
ProgStr): Window
WinForViewPort [*Sapph*] function(ServPort: Window; vp:
Viewport; var isouter: boolean): Window
WinNameArray [*SapphDefs*] type = Array[0..0] of ProgStr
Word_String [*CommandParse*] type = String[1]
Word_Type [*CommandParse*] type = (in_arg, out_arg,
switch_arg, switch_value, command_file)
WordifyPool [*CommandParse*] function(ChPool:
pCharacter_Pool; PoolLength: Char_Pool_Index; var
WordStruct: CommandBlock): GeneralReturn
World_NO_Read_Access [*IFileDefs*] const = #04
World_Write_Access [*IFileDefs*] const = #10
WriteBoolean [*Writer*] procedure(var F: FileType; X: Boolean;
Field: integer)
WriteCh [*Writer*] procedure(Var F: FileType; X: char; Field:
integer)
WriteChArray [*Writer*] procedure(var F: FileType; var X:
ChArray; Max, Field: integer)
WriteChars [*Stream*] procedure(VAR F: FileType; VAR S:
String)
WriteD [*PasLong*] procedure(var F: FileType; X: long; Field, B:

integer)
WriteFault [*AccentType*] const = AccErr+26
WriteFile [*PathName*] function(var *PathName*: Path_Name; Data:
 File_Data; ByteCount: long): GeneralReturn
WriteIdentifier [*Writer*] procedure(var F: FileType; X: integer;
 var IT: IdentTable; L, Field: integer)
WriteInteger [*Writer*] procedure(var F: FileType; X: integer;
 Field: integer)
WriteNChars [*Stream*] procedure(VAR F: FileType; c: char; N:
 Integer)
WriteProcessMemory [*AccInt*] function(ServPort: port; Address:
 VirtualAddress; NumBytes: Long; Data: Pointer; Data_Cnt:
 long): GeneralReturn
WriteR [*PasReal*] procedure(var F: FileType; e: real; TotalWidth:
 integer; FracDigits: integer; format: integer)
WriteSegment [*AccInt*] function(ServPort: port; Segment: SegID;
 Offset: Integer; Data: Pointer; Data_Cnt: long):
 GeneralReturn
WriteString [*Writer*] procedure(var F: FileType; var X: String;
 Field: integer)
WriteX [*Writer*] procedure(var F: FileType; X, Field, B: integer)
WRONGARGS [*AccentType*] const = 2
WrongEnvVarType [*EnvMgrDefs*] const = Env_Error_Base+2
WS_NotFound [*CommandParse*] const = -1
WS_NotUnique [*CommandParse*] const = -2
WSQMapInfoRecord [*QMapDefs*] const =
 WordSize(QMapInfoRecord)
WSSourceMapRecord [*QMapDefs*] const =
 WordSize(SourceMapRecord)
Zone_Info [*TimeDefs*] type = packed record TimeZone: integer;
 UseTimeZone: boolean; Daylight: boolean; UseDaylight:
 boolean; end

NOTES ON THE KRAUT DEBUGGER

August 1, 1984

Copyright © 1984 PERQ Systems Corporation
2600 Liberty Avenue
P. O. Box 2600
Pittsburgh, PA 15230
(412) 355-0900

Accent is a trademark of Carnegie-Mellon University.

Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its Spice Project.

The Kraut program was written by Bernd Bruegge.

This document is not to be reproduced in any form or transmitted in whole or in part without the prior written authorization of PERQ Systems Corporation.

The information in this document is subject to change without notice and should not be construed as a commitment by PERQ Systems Corporation. The company assumes no responsibility for any errors that may appear in this document.

PERQ Systems Corporation will make every effort to keep customers apprised of all documentation changes as quickly as possible. The Reader's Comments card is distributed with this document to request users' critical evaluation to assist us in preparing future documentation.

PERQ, PERQ2, LINQ, and Qnix are trademarks of PERQ Systems Corporation.

KRAUT: PERQ PASCAL REMOTE

SYMBOLIC DEBUGGER

We are enclosing the *User Manual for KRAUT—The Interim Spice Debugger* by Bernd Bruegge. This manual is distributed with the permission of Siemens RTL, Princeton, New Jersey.

Please note the following variances between the version of Kraut that PERQ Systems is distributing with the Accent operating system and the version of Kraut that is documented in the enclosed manual. The areas of major difference are filenames and the usage of process control signals.

1. File-names and Logical-names from the environment in the released version of Kraut are as described in the document "Basic Operations" in the *Accent User's Manual*. That is, an absolute file-name is a machine-name followed by a partition-name followed by a (possibly empty) sequence of directory-names ended with a simple-name. All of these name components of the absolute file-name are separated by a "/" character. An example:

/DistPerq/Accent/LibPascal/PascalInit.Pas

refers to the file PascallInit.Pas contained in the LibPascal sub-directory within the Accent partition on the machine DistPerq. Logical names for Environment searchlist variables also conform to the syntax described in the "Basic Operations" document, e.g. they must end with a colon (:). This differing file syntax, of course, affects any command which may have a file-name as a parameter. Most of these are the Editor commands, described on page 13 of the Kraut manual, and the Searchlist commands, described on page 10 of the Kraut manual. Note that the SetSearch searchlist command has also been modified to be more akin to the Shell's SetSearch command. Kraut's SetSearch takes only one parameter: either the file-name of the directory to be pushed onto the "default:" searchlist or the switch "-POP" to remove the top item from the "default:" searchlist.

2. The Target Process commands described on page 19 of the Kraut manual are significantly different in the released version of Kraut. The CTRL-DEL-t, CTRL-DEL-c, and CTRL-DEL-k commands are not currently implemented. The CTRL-DEL-d command must be entered to the TARGET's window, not Kraut's.
3. The update command described in Section 1.1, "Getting Started," in the Kraut manual applies only to users at

Carnegie-Mellon University.

4. The model commands described on page 26 of the Kraut manual are not currently implemented.
5. Finally, Kraut may be invoked from the Shell command through the use of the "[-DEBUG]" special Shell switch rather than the use of the "@" as described on page 2 of the Kraut manual.

USER MANUAL FOR KRAUT: INTERIM SPICE DEBUGGER

Carnegie-Mellon University

Department of Computer Science

Spice Project

Bernd Bruegge

10 February 1984

Copyright © Bernd Bruegge

Kraut is a contribution of Siemens RTL, Princeton, New Jersey, to the SPICE project. This document is a revised version of the technical report RTL-83TR-008 from Siemens RTL.

Accent is a trademark of Carnegie-Mellon University.

Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its Spice Project.

The Kraut program was written by Bernd Bruegge of Siemens RTL.

PERQ, PERQ2, LINQ, and Qnx are trademarks of PERQ Systems Corporation.

Table of Contents

	Page
<u>1. Introduction</u>	1
1.1. Getting Started	2
<u>2. Naming</u>	7
2.1. Constants	7
2.2. Types	8
2.3. Scope Rules	9
<u>3. Path Rules</u>	13
3.1. Generalized Path Expressions	13
3.2. Path Actions	16
3.3. Manipulating Path Rules	17
<u>4. Command Language</u>	19
4.1. Search List Commands	20
4.2. Trace Commands	21
4.3. Breakpoint Commands	22
4.4. Editor Commands	23
4.5. Definition Commands	24
4.6. Path Rule Commands	25

4.7. Variable Inspection Commands	27
4.8. Next Command	28
4.9. Runtime Stack Commands	29
4.10. Target Process Control Commands	30
4.11. Miscellaneous Commands	32
4.12. Window Commands	37
4.13. Model Commands	38
<u>5. Mace</u>	41
 5.1. Mace Commands	44
<u>6. Coping with Debugger Bugs</u>	49
<u>7. Current Restrictions and Known Bugs</u>	51
<u>Appendix A. Kraut Command Summary</u>	55
A.1. Breakpoints	55
A.2. Constant syntax	55
A.3. Declarations	56
A.4. Editor Commands	56
A.5. Identifier Syntax	56
A.6. Inspection Commands	56
A.7. Line Number Syntax.	57
A.8. Mace	57
A.9. Miscellaneous	57

A.10. Model Commands	58
A.11. Next Command	58
A.13. Reporting Bugs	59
A.14. Runtimestack Commands	60
A.15. Show Command	60
A.16. Searchlist Commands	61
A.17. Target Process	61
Commands	
A.18. Trace Commands	62
A.19. Type Qualifier	62
A.20. Window commands<	62
Notes	63
Index	65

1. Introduction

Kraut is a remote symbolic debugger for Perq Pascal running under the Accent operating system. One of the novel features of Kraut is that the user can specify the expected behaviour of a computation with path expressions. The debugger runs in its own address space and inspection and modification of the target process address space is done by means of Accent kernel calls. Currently both the debugger and the target process have to reside on the same Perq, but later it will be possible to debug processes which are located on a physically remote Perq.

The manual is organized as follows: Section 1.1 explains how to retrieve the newest version of the debugger, how to generate symbolic information and how to invoke the debugger. Pascal's scope rules have been extended in Kraut and are described in Section 2. The major design objective of Kraut was to test the viability of path expressions in the context of debugging. Path expressions are embedded in path rules which are introduced in Section 3. In addition to path rule commands, Kraut provides traditional debugging commands for setting breakpoints, inspecting and modifying variables in the user process address space, opening source files and searching for text strings. The complete command language of Kraut is described in Section 4. Kraut contains the low level debugger Mace, which allows the inspection and modification of the target process on qcode and (partially) on microcode level. Mace is intended

mainly for compiler writers and microprogrammers and is described in Section 5. Kraut is still actively being developed and internal debugger errors might show up while you are debugging your program. Section 6 shows you how to cope with these errors and Section 7 mentions the current restrictions and known bugs of the debugger.

1.1. Getting Started

If Kraut is not yet part of your Accent system or if you want to get a newer version, you have to retrieve it from the CFS Vax@CMU. Kraut can be retrieved using the update program with the following command:

```
update krautrun/test
```

The run file is debugger.run and it has to reside in the partition <boot>. Symbolic information for a module is produced by the compiler if the module is compiled with the /scrounge switch. For example

```
compile test.pas/scrounge
```

generates the symbolic information for test.pas. The /scrounge switch is on by default. If you want to suppress the generation of the symbolic information use noscrounge. The symbolic information is contained in two files with the extension sym and qmap¹. Kraut must have access to these files to allow symbolic debugging at other than a routine and module name level.

Kraut can be invoked in the following ways:

1. It can be called on the shell level by closing the Run command with the ^ character; for example *RUN test^*. This

allows the definition of path rules, debugger variables and breakpoints before the execution of the program.

2. Typing the Shell command Debug² will suspend the execution of a running process. In this case the Spice Process Manager reports a Trap 0 in the target process and invokes the debugger. This allows you to catch processes in dead loops or in I/O wait states.
3. Any uncaught exception in the target process will invoke Kraut.
4. Kraut can be invoked via Sapphire if the item Debug Process is selected from the Pop-Up Menu. The menu is displayed from a window or an icon in a manner that is consistent with regular Sapphire window management.

In all of the above methods of invoking Kraut, a bug symbol appears in the icon corresponding to the window running the process in question. The Sapphire cursor appears on the screen and prompts for the creation of a Debug Window.

Kraut tries to set the current directory to the directory where the main program was compiled. If that directory does not exist, a warning is given and the current directory is left as <current>.

Kraut does a consistency check between symbolic information and object files: if a symbol table was not produced at the same time as the object file or if the

source file is younger than the seg file a warning is issued. In this case the debugger should be used with caution, because any information from those modules could be wrong. Any information retrieved from inconsistent symbol tables is marked with a '?'.

Kraut has a transcript facility to document errors that cannot be reproduced and bugs in modules not written by yourself. If you encounter a bug you cannot cope with, you might want to send the transcript file to the maintainer of the module (see Maintainer and Report commands). Transcript files are generated by default, but you can turn the transcript switch on or off (see Script command). If the transcript switch is turned OFF in the default command file the generation of a transcript file is suppressed. A transcript file has the name "M.kscr.i", where M is the extensionless name of the file containing the main program, and where i = A,...,Z. If the name space of the transcript files is exhausted, you are prompted for a file name.

At the beginning Kraut prints out a stack trace of the last 4 routine calls. If the symbol table information files exist, calls are written out in terms of the source program. If no symbolic information is available, the low level debugger Mace writes out the routine call in terms of the target code. For example

```
| Uncaught exception: Division by zero
| (EXCEPT"RAISEP) Q321(19,4,2306,2306, ...)
| DODIV (23 test.pas;1) j : = a div parm;
| EXECUTE (17 sys > user > bob > bug.pas;1 ) Dodiv(a,parm);
| ? MAIN (34 test.pas;1) parm : = 0; Execute(parm);
```

shows the trace of a program that did not catch the exception Division by Zero. The source line for the

routine on top of the stack could not be found, because the defining module was compiled with the noscrounge switch. The next three lines are given in terms of the source text. The last source line is marked with a '?' indicating that the symbolic information was not generated at the same time as the seg file. If the top routine is a predefined exception, the current routine is automatically set to the routine below the top routine, otherwise the current routine is set to the top routine.

Kraut looks for 2 files³ whose names are derived from the extensionless file name M containing the main program declaration. The file M.switches defines certain user definable constants and switches such as the prompt sign and whether a transcript file of the debugging dialogue is to be generated. The M.kmd file must be a Kraut command file and its commands are executed before any other command can be typed in by the user. If M.switches does not exist the debugger assumes the following defaults: The prompt sign is "?" and the transcript switch is on. If M.kmd does not exist, control is immediately handed over to the user.

PERQ Systems Corporation
Accent Operating System

Kraut Debugger
Naming

2. Naming

Since Kraut is a symbolic debugger it should be able to access any object mentioned in the source program. However, in many cases the current symbol table format does not provide enough information to access the full name space or to enforce Pascal's type rules. This section describes several extensions made to the Pascal syntax to allow symbolic debugging despite these constraints.

2.1. Constants

A constant can be prefixed by a '#', a base indicator and a type indicator. The base is indicated by B, D, H or O (binary, decimal, hexadecimal or octal) and the type by L or I (long or integer). If the base or type indicator is omitted, the defaults D (decimal) or I (16 bit constant) are assumed. A normal Pascal constant in the range -32768 to + 32767 is of type integer, otherwise of type long.

Examples:

#OL4743337 32 bit octal long

#HLABF9DC93 32 bit hexadecimal long

#b010111010111100
16 bit binary

#3434 16 bit decimal integer

3434 16 bit decimal integer

986966 32 bit decimal long

2.2. Types

The debugger tries to do some type checking and issues a warning if it finds a violation of Pascal's type rules. However, only simple types are checked correctly. **Arrays** are of the same type if they have the same subtype. **Records** are considered equivalent if they have the same length. All **pointers** are of the same type.

The type of any variable or Pascal expression can be explicitly specified by one of the following type qualifiers:

<u>:array</u> [n]	Array
<u>:boolean</u>	Byte
<u>:char</u>	Char
<u>:integer</u>	Integer
<u>:long</u>	Long
<u>:pointer</u>	Pointer
<u>:record</u> [n]	Record
<u>:real</u>	Real
<u>:string</u>	String
<u>:set</u> [n]	Set

Each type qualifier is uniquely specified by the underlined characters. The optional expression [n] specifies the wordsize of the structure. If [n] is not given, the default [1] is assumed.

Examples:

```
Var
  a: array [ 0..10 ]  of integer;

  rec: record
    a, b,c : integer;
    d      : pointer;
  end;

  i, gi : integer;
```

.....
a[j] :long denotes a 32 bit variable.

(rec. 3^:long +5 * gi:integer):s is interpreted as string.

gi: int := # b1101011111 assigns bit pattern
0000001101011111 to gi.

i:set[1]= prints out the value of i as 16 bits ⁴.

2.3. Scope Rules

Kraut allows you to circumvent the scope rules of Perq Pascal. For example, variables from the **private** section of a module can be accessed, even if they are not visible. The Pascal notation for an identifier has been extended: To denote the variable Foo in routine P you can write P'Foo. And M''P'Foo denotes the variable Foo in routine P in module M. More generally, M''P₁' ...'P_{n-1}'P_n,i) denotes variable i in routine P_n nested in P_{n-1},..., nested in P₁, which is declared in module M⁵. One can also access local variables of routines currently allocated, but not visible. Local variables of activated routines that are outside the current static scope can be accessed by specifying the call chain in front of the variable name. A call chain consists of names of routines separated by a back slash ('\''). For example

M''P\MI''foo\MI''foo'i denotes the variable i in the second invocation of routine foo in module M1 called from routine P in module M.

Such a denotation may specify an object that is currently not in existence. If it is not part of a predicate (see section 3.1, the debugger prints an error message and returns to the user level. For predicates, a three-valued logic is assumed and predicates that cannot be evaluated will yield the predefined value —UNDEFINED. Any composite predicate containing undefined values will also yield —UNDEFINED.

Identifier search in Kraut follows the following algorithm:

1. If the name does not contain a module or routine qualifier, then it is parsed according to Perq Pascal's scope rules:
First the name is looked up in the runtime stack starting at the current routine. If this is not successful, then the name is looked up in the current module. The current routine and module can be set by the Current command.
2. If the name is of the form P'i, Kraut searches the runtime stack for each module mentioned in Kraut's search list for a routine with name P. If such a routine is found, Kraut looks for the local variable i. The symbol tables of the modules are searched in the order defined by Kraut's module search list (see section 4.1).

3. If the name is of the form M'i, Kraut looks for variable i in module M. Note that Kraut does not distinguish, whether i is in the private or in the exported section.

PERQ Systems Corporation
Accent Operating System

Kraut Debugger
Path Rules

3. Path Rules

Originally path expressions were developed for the synchronization of concurrent processes. In Kraut they are used as pattern recognizers to monitor the behaviour of a target process. Path expressions are embedded in the more general notion of a path rule. In this section we show how path rules are defined and give some examples for their use in debugging. A path rule consists of two parts: an event recognition part and an action part. The event recognition part consists of a generalized path expression which is discussed in section 3.1. The purpose of the action part is to describe what the debugger has to do in the case of a match or mismatch of the actual computation with the computation described by the recognition part. The action part is described in section 3.2.

3.1. Generalized Path Expressions

Generalized path expressions are basic path expressions extended by counters and predicates.

A basic path expression is a regular expression of operands connected with the operators repetition (*), sequencing (;) and exclusive selection (|). The operands are called path functions in the following. Any routine defined in the source program is a predefined path function. Other path functions can be defined during the debugging session with the Define command described in section 4.

If a path function R is used in a path expression it matches either the activation or termination of the execution of R in the target process. Path functions can be postfixed with an event qualifier to specify either the activation or termination of a routine: If R is a path function, then R!A denotes the activation of R and R!T denotes the termination of R. Thus R can be seen as a shorthand notation for the generalized path expression (R!A | R!T)).

A counter can be defined explicitly with the Counter command described in section 4. Furthermore, there are two predefined counters for each path function mentioned in a path expression: The __Act counter describes how many times the path function has been activated and the __Term counter describes how many times it has been completed⁶.

A predicate is a relational expression consisting of implicit or explicit counters and names from the name space of the program. Predicates have to be enclosed in curly brackets and can be associated with any path function⁷. There is a predefined predicate Always which is equivalent to True=True.

An example of a generalized path expression is

```
InitStack;  
(Push {(__ TERM(Push) - __ TERM(Pop)) < N} )  
(Pop | Top){(__ TERM(Push) - __ TERM(Pop) > 0} );
```

which states the operational constraints on a bounded stack of length N: First the routine InitStack has to be called. One of the following can then happen: Either Push can be called as long as the size of the stack is smaller than N, or Top or Pop can be called

if the size of the stack is larger than 0.

Generalized path expressions can be used in two different ways: If a generalized path expression is prefixed by the keyword Find, the specified execution sequence is matched against the observed execution sequence and the path action specifies what to do in the case of a match and a mismatch. Find expressions are useful in searching for the pattern of a particular execution sequence. For example,

```
" FindPath BeginLoop
  WhileLoop {__ ACT(Push) = N and __ ACT(Pop) = 1 }
```

looks for the activation of the while loop when the routine Push has been called N times and Pop once.

The other use of generalized path expressions is to enforce a particular execution sequence. If a generalized path expression is prefixed by the keyword Check, the specified execution sequence is matched against the observed execution sequence. The path action contains commands about what to do in the case of a violation or nonviolation. For example,

```
CheckPath Loop
  (WhileLoop {__ ACT(Whileloop) < 6 } | Pop)
```

says that the while loop should not be executed more than 6 times before a call to Pop occurs.

Any identifier that is used in a predicate of a generalized path expression is called a path variable. Predefined path variables are identifiers explicitly declared in the source program and accessible via the symbol table.

3.2. Path Actions

A path action is declared by the keyword Action followed by the identifier of the path expression with which the path action is to be associated. The keywords Match and NoMatch are followed by the actions to be taken in case of a match and mismatch, respectively. Any sequence of debugger commands not containing an assignment or routine call is a valid action. Assignments and routine calls have to be preceded by a Halt command. If an action contains more than one command, the commands have to be separated by ' ~ '. Either the Match or the Mismatch part or both can be omitted, in which case no action will be associated with the missing part. For example

```
Action Loop
    Match    => Writeln("I = ", I) >> LoopWindow
    NoMatch  => HALT
```

associates the debugger commands Halt and Writeln("I = ",I) >> LoopWindow) with the path expression Loop. Thus whenever the evaluation of Loop⁸ yields a violation of the specified ordering the computation is suspended and control is turned over to the debugger, otherwise the value of I is written into the window LoopWindow.

The association of a path action with a generalized path expression is dynamic. Thus you can remove a path action from a path rule and replace it by another path action. This can be done any time the target process is suspended: If the process is still running, you can suspend it with the characters \&DEL k) or \&DEL d) (see page 30).

3.3. Manipulating Path Rules

Path rules are automatically enabled when they are defined. They can be disabled and enabled again by the DisAble and Able commands described in section 4.6. Note that if a Match or a Mismatch occurs, the path rule is not automatically disabled. Disabling a path rule and successive enabling has to be done explicitly. Every time the Match part of a path rule is executed the path rule is set to its initial state, that is, is starting over again to look for the pattern specified in the generalized path expression.

PERQ Systems Corporation
Accent Operating System

Kraut Debugger
Command Language

4. Command Language

Any command can be abbreviated as long as the abbreviation is unique. For convenience, an attempt has been made to allow two letter abbreviations for nearly all commands. Because of this reason some of the command names might look a little bit strange. Commands and variable names are not case sensitive. For example, 'R' and 'r' mean the same command or identifier. If more than one command is entered per line, the commands have to be separated by the character "~~". A command or command sequence can be extended over several lines. Each of the lines of such a command has to be terminated with a `~` character. For example,

```
callstac \
k 1~ writ\
eln('l = ',l) ~\
go
```

is parsed as:

```
callstack 1~ writeln('l = ',l) ~ go
```

At any time the debugger maintains a current line number and a current source file which can be denoted by the dot character ("..").

4.1. Search List Commands

Kraut maintains a search list of open modules which determines the order in which symbol tables are searched for identifiers. The search order is the reverse of the order in which the modules are entered: The module opened latest will be searched first. By modifying the search list appropriately you can speed up the debugger significantly. The module search list is initially empty. Any time the target process is suspended, the modules of routines on the runtime stack not yet in the module search list are appended to the tail of the module search list. Modules are never implicitly removed from the list.

Kraut looks up the source, qmap and sym files of a module using the current file search list. The Setsearch command can be used to add directories to the file search list. This makes it possible to access remote files from inside the debugger. If the file search list did not contain the directory in which the source or symbolic files are located, when the module was Opened, Kraut will try to open the module without this information. If you want to add these files later, Close the module, call Setsearch with the appropriate (remote) directory and then call Open again.

ChDir D Set the current directory to D. If no argument is specified, show the current directory.

Open M Insert module M at the head of Kraut's module search list. If the module is already in the search list, it will be moved to the head of the list. The variable Current File is changed to the file name of the module opened. If M is a list of modules M_1 $M_2 \dots M_n$, then open each of the modules as described above. If M = *, then open all the

modules of the target process, except for the modules of the Pascal runtime system. The latter ones can be opened with the System command.

Close M Delete module M from Kraut's search list. If M is a list of modules $M_1 M_2 \dots M_n$, then delete each of the modules. If $M = *$, then close all the modules of the target process.

SetSearch F_1, \dots, F_n

If no arguments are given, then show Accent's file search list. Otherwise modify the file search list as follows: If F_i is a directory name, push it on the search list, if F_i is a "-" then pop the top entry from the search list.

Show Modules M

If M is not given, show all open modules from Kraut's module search list. Otherwise give some information about module M. This includes date of compilation, names of seg, qmap and sym files, name of imports, etc.

System

Initially the Pascal runtime system modules are not in the search list, even if they are explicitly imported by the user program. The Pascal runtime system consist of the module PascallInit and all its imports. System opens these modules and adds them to the search list.

4.2. Trace Commands

Traces are implemented internally as path rules. For each routine to be traced, two path rules are created. The first one has the name __AR and monitors the Activation of R. The second one has the name __TR and monitors the Termination of R. The Match part in the path action is preset to print whether the routine is entered or exited. Traces can be edited with the EditPathRule command (see page 26).

Trace [Before | After] R [UserActions]

R can be either a module or a routine. If R is a module tracing path rules are created for each routine

defined in the module. If it is a routine and neither Before nor After is specified, both path rules __AR and __TR are created. If Before is given, only the path rule __AR to trace the entry of R is created. If After is given, only the path rule __TR to trace the exit of R is created. If "UserActions" is specified, then "UserActions" are executed whenever one of the trace path rules fires.

DTrace [Before |After] R

If R is a routine delete the tracing path rules for R. If R is a module delete the traces for all routines of R. If Before (After) is given, delete the only the path rule for the entry (exit) of R.

4.3. Breakpoint Commands

Setting of breakpoints can be done either by specifying a source line or by a routine name. In the following, the symbol R denotes a routine. R can be of the form M''P or P. The symbol '#' denotes a source line in a source file in a certain directory. If the file name is omitted, the current file is assumed. The line number must correspond to a source line containing the beginning of a Perq Pascal statement, otherwise Kraut writes an error message. Breakpoints are implemented internally as path rules. The name of the path rule is of the form __B#^o, where # is in the range 1..150. The Match part in the path action is preset to the command sequence

Halt~Exception~Callstack 1. Breakpoints can be edited with the EditPathRule command (see page 26).

NOTE: The Halt command is part of the definition of a break point and cannot be removed from the Match part.

Break (after) R , Break #

Set Breakpoint at (after) routine R or at line #. For

example Break 4 main.pas means: Set a break point at source line 4 in file main.pas.

DBreak * Delete all Breakpoints.

DBreak (after) R, DBreak #

Delete Breakpoint at (after) routine R or line #.

Show Breaks

Show current breakpoint list. This command is the same as Show Pathrules if no other pathrules but breakpoints are active.

4.4. Editor Commands

The following commands allow you to inspect source file and search for string patterns. If a new file name is specified, the current file will be set to the new name.

Type i j F

Show source lines i to j in file F. The current source line is set to the last line displayed. F must be visible with Accent's file search list. There are several abbreviations of the type command: The keyword Type can be omitted. If the file name is also omitted, then the current file is assumed. Furthermore, the commands i F or i will display line i in file F or in the current file, respectively.

Around i F

Show 20 source lines around line i in file F. Line i is indicated by '***'. If F is omitted, the current file is assumed. If i and F are omitted the current source line is assumed.

Scroll i F Type 20 source lines starting at the current source line. The Next command can be used after the first scroll.

Search 'S'? F]

Search backward in file F for string S. If S is omitted take the string from the last search command. If S contains a "", then two "" have to be specified. Note that the search is not case sensitive. If F is omitted, the current file is assumed. If 'S' is

found, the current source line is set to the line containing 'S'.

Search 'S'! F

Search forward in file F for string S. The '!' can be omitted if S is given. If S is omitted take the string from the last search command. If S contains a "", then two "" have to be specified. Note that the search is not case sensitive. If F is omitted, the current file is assumed. If 'S' is found, the current source line is set to the line containing 'S'.

Grep 'S' F

Look for all occurrences of string S in file F. The search is case insensitive. If F is omitted than do string search starting at the current source line in the current source file. If 'S' is replaced by !, then look with the last string pattern specified. Currently neither S nor F can contain wildcards.

4.5. Definition Commands

Kraut allows the definition of new path functions, counters and path variables which can be used in generalized path expressions in the same way as that for predefined names.

Counter C := I

Declare a counter variable C and assign it the value I. If the assignment is omitted, the counter is initialized to 0. Counters are regarded as global variables defined in a static scope surrounding the main program. If there is a variable with the same name defined in the program, Kraut prefixes the counter variable with the escape character '__' and asks you for confirmation.

DCounter C

Delete the counter variable C.

Define Function P B E

Associate the path function P with a group of statements indicated by the source lines B and E in file F. B and E must belong to the same source file

and the source line denoted by B must be smaller than the source line denoted by E.

4.6. Path Rule Commands

The debugger maintains a list of path rules which can be defined and manipulated. In the following, R denotes the name of a path rule and can be any identifier.

Action R A

Bind action A to path rule R. The arguments R and A can be omitted, in which case you are prompted for them. R must have been defined before by a FindPath or CheckPath command. A is of the form:

Match Cmd1 NoMatch Cmd2

Cmd1 and Cmd2 can be any Kraut command or command sequence. The Match part has to be defined before the Nomatch part. Command sequences must be enclosed in parentheses. If the execution sequence matches the sequence specified by the generalized path expression, then Cmd1 is executed. If there is no such match, then Cmd2 is executed. For example

Action R1 Match
(calls ~ br firstcall ~ disable R1 ~ enable R2)

Executes the Calls command, sets a breakpoint at routine firstcall, disables itself and enables path rule R2.

CheckPath R G

Qualify R as a Check rule and add it to the active path rule list. G is a generalized path expression (see section 3.1). If the arguments R and G are omitted, you are prompted for them and for the associated path action.

Disable R Disable the currently active path rule R. This command takes the path rule R from the active path

rule list and appends it to the list of inactive path rules.

Editpathrule P

This command allows to change the components Generalized Path Expression, Match and NoMatch of path rule P. Furthermore the user is prompted for the setting of two switches: The Enabled switch enables P if set to true, otherwise disables it. If the Verbose switch is set to true, the interpretation of the path actions is done verbose, otherwise quiet. *Note that editing of path rules created by the Break command is restricted in the following way: It is not possible to change the whole Generalized Path Expression associated with the breakpoint, but only its predicate. Predicates can be entered when the Condition prompt is given.*

The following example sets a breakpoint at line 86 and changes the default action to do the following commands: Print out the callstack to depth 4, print the locals of the current routine, write the value of i and continue. The action will only occur if the target process executes line 86 and i has the value 1000.

```
BREAK 86 test3.pas
Set Breakpoint __ B1 at line 86 in
TEST.PAS: write('First;');

EDIT __B1
CONDITION () : i = 1000
MATCH () :
Callstack 4 ~ locals ~ writeln('i = ', i) ~ go
NOMATCH () :
ENABLE () : TRUE
VERBOSE () : TRUE
```

Enable R Enable the currently inactive path rule R. This command takes the path rule R from the inactive path rule list and appends it to the list of active path rules.

FindPath R G

Qualify R as a Check rule and add it to the active path rule list. G is a generalized path expression(see

section 3.1). If the arguments R and G are omitted, you are prompted for them and for the associated path action.

Show PathRules (enabled | disabled)

Show the active or inactive path rule list. If no argument is specified show the entire path rule list. (This command is equivalent to Show PathRules).

Show PathRules F

Show the components of path rule F.

RemovePath P]

Delete path rule P from the path rule list. If P is a list of path rules $P_1 P_2 \dots P_n$ then remove each of these path rules from the list.

Remove Action R

Delete the action bound to path rule R.

4.7. Variable Inspection Commands

In the following V and Id denote identifiers as defined in section 2, page 7, counters as defined in section 4.5, page 24 or history variables. Id can also be an arbitrary Pascal expression with the restrictions mentioned in section 7, page 51.

Globals M

Display all globals of module M. If M is not specified, display the locals of the current module.

Locals R Display all locals of routine R. If R is not specified, display the locals of the current routine.

Parameters R

Show the current values of the parameters of R. If R is not specified, then show the parameters of the current routine.

Radix N Set the current radix to N where N is a decimal number from the set { 2,8,10,16}. Initially the current radix is 10.

V = Get the value of V following the scope rules as described in section 2.3. If the name of the variable

is not also a Kraut command, then the equal sign (=) can be omitted. To avoid confusion arising from names used in more than one module, values are always written out in the form "value (Module)". Module is the name of the module containing V or it is the string Internal Counter if V has been defined as a counter variable. If the value was retrieved from an inconsistent symbol table, the module name is marked with a '?' sign.

V := Id Assign the value of Id to V following the scope rules described in section 2.3. V cannot be a history variable. The debugger tries to do some kind of type checking as described in section 2 and issues a warning if there is a violation.

Write(p1,p2,...,pn), Writeln(p1,p2,...,pn)

Write and Writeln are similar to Pascal's write and writeln. The only difference is that the output is always written in the current window. P₁,P₂...,P_n can be any Pascal expressions with the restrictions and extensions described in section 2, page 7. For example, if foo = 'df' and bas = 1, then Writeln('FOO = ', foo, 'BAS = ', bas) results in the output:

FOO = 'df'BAS = 1

4.8. Next Command

<CR> This command is interpreted in the context of the last command. The next command is implemented for the following commands:

\$ Mace command: Display the next location in the current mode and current radix.

Around Show the next 20 source lines in the current file.

Calls Display one more call. The current source line and file are updated to the line displayed, so typing Around after this command displays the context around the source line. Note: The current routine is not changed.

DownStack Move down one activation record in the dynamic call chain.

Forward Search

Continue the search with the previous argument starting at the current source line.

Scroll Show the next 20 source lines in the current file.

SingleStep Execute one more source line.

Type Show the next 20 source lines in the current file.

UpStack Move up one activation record in the dynamic call chain.

4.9. Runtime Stack Commands

BottomStack

Set the current routine to the routine described by the activation record at the bottom of the runtime stack.

Calls N Display the dynamic calling sequence of the target process. N specifies the number of calls to be printed. If N is omitted, then the whole sequence will be printed.

DownStack N

Move down one activation record towards the bottom of the runtime stack and update the current routine. If N is omitted, N=1 is assumed.

TopStack Set the current routine to the routine described by the activation record at the top of the runtime stack.

UpStack N

Move up N activation records towards the top of the runtime stack and update the current routine. If N is omitted, N=1 is assumed.

4.10. Target Process Control Commands

These commands allow to modify or observe the execution of the target process.

Process Control Characters

The process control characters \&DEL c , \&DEL d , \&DEL k and \&DEL t are intercepted by Kraut. Their interpretation depends on the context of the last command:

EXECUTE,GO

\&DEL t : Show the state of the target process.
(Not yet implemented)

\&DEL c , \&DEL d , \&DEL k : Suspend the target process and return to command level.

\&Del r : NOOP

All other commands

\&DEL t : Show the state of the target process.
(Not yet implemented)

$\text{\&DEL r}, \text{\&DEL d}$: Abort the current command.

\&DEL c : NOOP

Execute R(p_1, p_2, \dots, p_n)

Execute routine R with parameters p_1, p_2, \dots, p_n . R can be the main program, any global routine or a nested routine statically visible from the current point of execution. If R is the main program, you are prompted whether the run time stack shall be cleared or not. If R has been defined with parameters, they have to be passed, but the debugger does no type checking at all. If the closing parenthesis is omitted, the debugger writes out the type information for each parameter and asks for confirmation. The syntax for parameters is similar to Pascal's syntax except for the

following two cases:

1. Var parameters must be prefixed with a `^`.
2. Normal Pascal constants are treated as integer decimals if they are in the range -32768..32767, otherwise as long decimals. Other constants have to be prefixed by a '#', a base indicator and a type indicator as described in section 2.1. For example:

Execute Foo(^ Bar, 34, #H5FA4, # BL0111, Count)

calls procedure Foo with reference parameter Bar, decimal integer 34, hexadecimal integer 5FA4, binary long 0111 and value parameter Count. The variables Bar and Count are visible from the current point of execution.

If R is a function, more restrictions have to be observed, most of them due to the insufficient symbol table information provided by the compiler: Functions names must always be indicated by parentheses followed by a Type Qualifier. Parentheses are needed even if the function does not have any parameters, otherwise the result of the function is denoted. Functions can be used in arbitrary expressions, but cannot be passed as parameters.

WARNING: It is easy to do something wrong when calling a function from inside the debugger. The following examples show how to use function calls and how not.

```
Type REC = record i : integer; l : long end;
Var b : boolean; r : REC;
Function foo(i : integer): boolean;
Function foo1: boolean;
Function bla(s : string(80)): REC;
```

Legal calls:

```
b := foo(i):boolean
b := foo1()
foo1()           — displays result of function call on screen.
r := bla('sdfsfd':string [80]):record [3]
                           - assigns the value of bla to r
bla('sdfsfd': string [80]):record [3]
```

- types the value of bla on on the display.

Illegal calls:

b := foo(i) — missing type qualifier

Problematic calls:

b := foo1 — Missing parentheses:

The debugger takes the current return value of foo1 instead of calling foo1.

r := bla('sdfsfd'):record(3)

- takes default value of string length.

Go Resume the execution of the suspended target process.

Halt Suspend the execution of the target process.

Run T Delete the link to the current target process and start the execution of target process T. This command is useful for debugging a new target process, for example if the current program has been recompiled and relinked, without leaving the debugger. NOTE: If you just want to restart the current target process, use the Execute command with the name of the main program as parameter.

TerminateTarget YES|NO

Terminate or do not terminate the target process on Quit. Initially TerminateTarget is set to YES.

Unwind Remove all activation records from the runtimestack up to and including the last routine that was called with the Execute command. If there is no such routine the stack is not changed.

4.11. Miscellaneous Commands

@F Execute command file F. If F is not specified, the command file M.kmd is executed, where M is the extensionless file name of the main program¹⁰. Command files can be nested to an arbitrary depth. If F does not contain an extension and there is no file with the name F, then the extension .kmd is appended.

\$M Execute the Mace command M without leaving the

Kraut command interpreter. Mace commands are described in section 5.

-- Start of a comment. Any string following the two dashes up to the end of the line is regarded as comment.

& If, the result of the last debugging command was a numeric value, it can be denoted by the symbol &. & can be used in any Pascal expression. The symbol &^A interprets the value of the last command as a pointer. Thus it especially useful for pointer chains.

Address ID

Show the target address of ID. ID denotes an identifier as defined in section 2.3. If ID is a routine, then show the addresses of its entry and exit points. If ID is omitted, the identifier from the last debugger command is taken. The address command allows you to inspect ID's more closely with Mace if Kraut does not provide any help (for example for record fields).

Compute E

Compute expression E. The Compute command evaluates (nearly) any pascal expression. The keyword Compute can be omitted if the expression starts with a constant¹¹. If the type of a variable is not known or has to be recasted, use type qualifiers. For example, if I is an integer and W a function of type integer and the result is of type real, the following command computes the real value 400022 + 5*7-i+w(i):

Compute (400022 + 5*7-i + w(i):integer):real

Current M If M is a module, set the current module to M (This is equivalent to Open M). Otherwise, if M is an active routine, set the current routine to M. If M is not active, do not change the current routine. If no argument is specified the current module and routine are displayed.

Exception Show the current exception.

- Help Creates a help window and allows the user to interactively peruse the appendix of this manual.
- Indent N Concatenate N blanks with the prompt sign. NOTE: In the case of a command error or when `\DEL c`, `\DEL k` or `\DEL d` is typed, INDENT 0 is automatically executed.
- Maintainer M Print out the name of the maintainer of module M). This command prints out the string following the key word **Maintainer** in the comment switch in module M. For example, if module Test contains `{$Comment Maintainer bob@cmux x3828}` in the source file, then "bob@cmux x3828" is the maintainer of Test. If M is omitted in the command, then the maintainers of all maintained modules are printed out.
- Mace Enter the command interpreter of Mace, the low level Accent debugger. Mace commands are described in section 5. To return back to the Kraut command interpreter type the Mace quit command.
- News Print out news about undocumented features, new bugs and fixes of old bugs.
- Quit There are four possible courses of action when choosing to quit the debugger. They are as follows:
- Quit Exits the debugger.
- Continue Resumes the debugging interaction.
- Report Refer to the Report Command below.
- SaveScriptAndQuit Automatically saves a transcript of the current dialogue if the transcript switch was On at least once during the debugging session; then exits from the debugger.
- Report M Create the transcript of the debugging session to be deposited in the outgoing mail for the mailing address M when the QUIT command is typed. If M is omitted, the account Spice@spice is assumed as the receiver. When quitting you are asked to confirm the mailing address, the FROM field (which is taken from the first line of the file <boot>sysname) and

whether you want to provide an additional message. The message can be in a file or it can be typed in the window. A typed message has to be terminated with a single dot on a line. The Report command can be typed at any time of the debugging session. Note that Kraut is not doing the actual mailing. It only puts the transcript in a file <boot>perqout.R*, from where it has to be picked up by the mail system.

EXAMPLE (user input is underlined):

```
| Report bob, dzg
| Mail request to bob, dzg registered
| ...
| Quit
| Action (Quit, Continue, Report,
| SaveScriptAndQuit) [REPORT] ? <CR>
| Mailing transcript...
| TO: [ bob, dzg ] bob@cmu-cs-spice, hibbard @ cmua
| FROM [ bob@cmu-cs-spice ]: <CR>
| Subject [ Bug in test ]: <CR>
| Do you want to add an initial
| message? (Yes, No) [YES] Yes
| Enter file name or hit <RETURN>: <CR>
| Type message (terminate with a "." alone on a line):
| Hi, I found the following bug:
|
| Preparing mail...
| ...Mail deposited for bob @ cmu-cs-spice,
| hibbard @ cmua
```

Script on/off

Turn the transcript on or off. If Script off is contained in the default command file, no transcript file will be generated at all.

Silence on/off

If silence is on, then the commands executed in a command file will not be echoed on the current window. If silence is off, they are echoed. The default value is off.

Shell

Spawn another shell, inheriting the environment in which Kraut is currently running.

Show P Depending on the parameter P the following is displayed:

P=Breaks Show the current break point list.

P=Counters

 Show the currently defined counter variables.

P=Models Show the current models.

P=Modules M

 If M is omitted, then type Kraut's module list. Otherwise give some information about module M. This includes date of compilation, names of seg, qmap and sym files, name of imports, etc.

P=Pathfunctions M

 Show the user defined path functions for module M. If M is omitted show the user defined path functions of all modules.

P=Pathrules P1

 If P1='active' then display the list of enabled pathrules; if P1='inactive', then display the list of disabled path rules, otherwise display all path rules currently defined.

P=Routines M

 Show the routines and user defined path functions for module M. M must be in the module search list. If M is omitted show the routines and user defined path functions of all modules.

P=RunFile (R M)

 If R and M are omitted, print the core image of the current target process into the file M.map, where M is the extensionless filename of the main program. If R and M are given, then print the contents of run file R into file M. Note that in this case some fields in the file and segment entry descriptors are set to 0 and do not describe locations in

the address space!

P=SearchList

Display Accent's file search list.

P=Window Display the currently defined windows.

Version Type the version number of the debugger.

4.12. Window Commands

The following commands allow the user to define typescript windows, change from one window to another and display information in any defined window. There are two predefined windows, which are treated differently from user defined windows: The Debug window is the window allocated by Accent when the debugger is started up. The Debug window is treated in such a way that it allows the debugger to run even if the window manager is not running. The BarGraphWindow is a graphic window in which variables can be bound to bargraphs (see section 4.13).

There is always a current window, which is originally set to Debug. There may be a maximum of ten windows, each of which may be manipulated by all of the normal Sapphire commands (top, bottom, move, grow, etc.).

Window W LX TY W H

Creates window W with upper-left corner LX, TY and W characters per line and H lines. If these coordinates are omitted, default coordinates are used.

Window W

Creates window W. Typing Window W a second time brings window W to the top and makes it the listener. This command is used to move the listener from one window to another.

DWindow W

Deletes the window with the name W.

DWindow *

Deletes all windows but the current one.

C >> W Redirect the output of the Kraut command or command sequence C into window W but do not leave the current window. If W has not yet been defined, a window with default coordinates will be created.

4.13. Model Commands

Model commands allow you to display variables with a format different from the built-in display format. A model is a display template defined by a Model command. Bargraphs are predefined models. Models are instantiated by the Bind command and it is possible to bind several variables to the same model.

Model Bargraph B MAX S

Create or rescale a bargraph with the name B, where B can be any identifier. If B has not yet been defined, a template for B is created. The maximal value of the template is given by the integer MAX. S describes the scale and can be either LINEAR (linear scale) or LOG (logarithmic scale). MAX and S can be omitted in which case the defaults MAX = 32767 and S = LINEAR are assumed. If the bargraph B already exists, the template and all its instantiations are rescaled according to the new values MAX and S. Note that the keyword Model can be omitted in the command.

Model Routine M R

Create a model with the name M, where M can be any identifier, and bind the routine R to it. The keyword Model can be omitted in the command. R can be any procedure defined in the user program and may have parameters. All except for one of R's parameters have to be bound at the definition of the model. It is possible to mark one of the parameter's slots with a "\$" character and this parameter will be

bound by the Bind command. For example,

Model Routine M Foo(^gi, # l23, \$)

binds the routine Foo with gi, #l23 and \$ to the model M, and \$ will be provided by the Bind command.

Bind ID M

Bind the variable ID to a model with the name M. Currently two classes of bindings are possible:

1. If M is a bargraph, then ID can be a counter, a history variable (ACT or TERM) or a variable of type integer. The debugger allocates a slot in the window BargraphWindow and displays ID according to the attributes of M¹². For example

Bind __ ACT(Foo) M

binds the history variable __ACT(Foo) to the bargraph M. After a variable is bound to a bargraph, any Display command will display its value according to the scale and maximum value of the bargraph. If ID is a counter or a history variable, the bargraph is updated on any assignment. However, if ID is a program variable, the bargraph is not updated until the next splay command is issued.

2. If M is a routine, then ID can be a variable of any type. If M has been defined with parameter "\$", then ID is substituted for "\$" in M. For example, the commands

**Model Routine M Foo(^gi, # l23, ^\$)
Bind grec M**

define a model M for routine Foo and bind grec to it. Displaying grec will result in calling Foo(^gi, #l23, ^grec).

DModel M

Delete the model M.

DBind ID Delete ID's binding to a model and rebind it to text
format.

5. Mace

The command syntax for Mace is quite different from that for Kraut. Some Mace commands may contain Mace expressions and qualifiers to change the radix and mode of the entities being displayed. The radix or mode qualifier is optional and if omitted the current radix or mode is assumed, respectively. The radix qualifier controls the radix of numbers and can have one of the following values:

- # Octal radix
- . Decimal radix

The mode qualifier specifies the type of object(s) to be displayed and can have one of the following values:

- a 16 bit integer, 32 bit long, byte, character and string (max 15 characters)
- b Byte
- c Character
- i 16 bit integer
- l 32 bit long
- m IPC Message
- q Qcode
- r Floating point
- s Pascal string

A Mace expression is a parenthesized expression of primary values using one or more of the following

operators:

\wedge	Dereference (written <u>after</u> the value)	6
-	Negation	5
\sim	Logical inversion	4
*	Multiplication	3
/	Division	3
!	MOD	3
+	Addition	2
-	Subtraction	2
&	Logical And	1
%	Logical Or	1
<	Leftshift	1
>	RightShift	1

The operators are ordered in decreasing precedence.
Parentheses can be used to indicate different
precedences.

A primary value is one of the following constructs,
where $\langle M \rangle$, $\langle P \rangle$, $\langle N \rangle$ and $\langle E \rangle$ are Mace
expressions denoting addresses and numbers.

892	Any decimal constant
#377	Any octal constant
.	The last location referenced.
$\langle M \rangle$	Base address of the code segment of module M.
$\langle P \rangle$	Entry point of routine P.
$\langle M \rangle.G\langle E \rangle$	Offset E in the global frame of module M.
$\langle M \rangle.R\langle E \rangle$	The E'th word in the routine dictionary of module M.
$\langle M \rangle''\langle P \rangle$	Entry point of routine

P in module M.
 $<P>.Q<E>$ Offset E in routine P.
 $<P>.A<E>$ The E'th word of the ACB
for last call to P.
 $<P>.<N>A<E>$
The E'th word of
the ACB for
the N'th call to P.
 $<P>.F<E>$ The E'th local word of
last call to P.
 $<P>.R<E>$ The E'th word in P's
routine dict entry.

Examples for Mace expressions are

Test3.Proc1 + #557^A Procedure

Proc1 in module Test3
plus contents of address
octal 557.

Proc1.4A3

The third word in the ACB
for the fourth call of
procedure Proc1.

Foo.Q23 - .

Qcode offset 23 in routine Foo
minus last expression referred to.

The following example shows the order of evaluation
of Mace expressions:

Foo.Q5 * Test.G45 & ~3 - 5^A

is evaluated as

(Foo.Q5 * Test.G45) & ((~3) - (5^A)).

5.1. Mace Commands

In the following $<E>$ and $<V>$ denote Mace expressions, $<R>$ a radix qualifier, $<M>$ a mode qualifier and $<D>$ a decimal integer. Note that command names are case sensitive, whereas Mace expressions are not.

? , h, H, '?'

Type help information. The commands ?, h and H explain Mace expressions and the ".", "=" and ":" commands. The command '?' explains the "''' commands.

$<E>; <R> <M> <D>$

Display $<D>$ target memory locations starting at $<E>$ in the format specified by mode $<M>$ and radix $<R>$ in units appropriate for the given type. $<D>$ must be a decimal integer. $<E>$ can be any Mace expression. For example:

Foo. q5423

displays 23 locations starting at address Foo.q54 as qcode instructions with decimal arguments.

$<E> = <R> <M>$

Calculate the Mace expression $<E>$ in mode $<M>$ using radix $<R>$. For example:

87506 + 45 *7 = #i

displays the value of 87506 + 45*7) as an octal integer.

$<E> := <V>$

Store the value $<V>$ in the 16 bit location $<E>$. $<E>$ and $<V>$ can be arbitrary Mace expressions. For example:

Test. G456 := Foo.F4

stores the value of the 4'th local word of routine Foo into offset 456 in module Test.

$<D>'a$ If $<D> = 1,2,\dots,100$, then show the contents of the $<D>$ 'th activation record (ACB) on the stack. $<D>$

= 1 means the ACB of the top routine and <D> = n means the n'th ACB down towards the bottom of the stack. If <D> is larger than 100, Mace assumes you have typed the address of the ACB¹³. The ACB consists of the following information:

ACBSL:	Activation pointer (AP) of enclosing routine
ACBPL:	Local pointer (LP) of this routine
ACBDL:	Activation pointer of calling routine
ACBGL:	Global pointer (GP) of calling routine
ACBTL:	Top pointer (TP) of calling routine
ACBRS:	System Segment number (SSN) of calling routine
ACBRA:	Program counter (VPC) of calling routine
ACBRR:	Routine Number of calling routine
ACBEP:	Pointer to current exception enable record
ACBStackSize:	Size of EStack of calling routine
ACBStackSize + i:	Saved EStack values ,i = 1,...,ACBStackSize
<E>'b	Set a breakpoint at address <E>. If <E>=0, show the list of current breakpoints.
<E>'d	Delete a breakpoint at address <E>.
'e	Display expression stack (from the bottom towards the top).
<D>'E<V>	If <D>= -1, then assign <V> to the micro state register EStkCount, which holds the current size of the EStack. If <D> in [0..15] then assign <V> to expression stack register <D>. (Note: 0 is the bottom of the expression stack).
<E>'h	Display the history for message <E>. If <E>=0 then display the history of all messages. This command assumes that the program you are debugging imports the module AccCall from AccCall.pas. If this is not the case, you get the error message No module specified. AccCall must be specially compiled to enable the message history

mechanism. (The message history mechanism is currently disabled.)

<D>'m If <D>=0, show the qcode state of the current routine¹⁴. Otherwise show the qcode state for the <D>'th activated routine on the stack, counting downward towards the bottom of the stack. Thus, <D>= 1 means the routine on top of the stack, etc. The qcode state consists of the following information:

SB: Stack base
CB: Base address of code segment
GP: Global pointer
VPC: Program counter
RN: Routine number
LP: Local pointer
AP: Activation pointer (Base address of ACB)
TP: Top pointer

'M Displays the contents of registers 5 to 16 and 110 of Accent's micro context block. A typical output for the 'M command looks as follows:

Program counter	[R 5] : 172 [# 254]
Top pointer	[R 6] : 21290 [# 51452]
Activation Pointer	[R 7] : 21280 [# 51440]
Global Pointer	[R 8] : 2562 [# 5002]
Local Pointer	[R 9] : 21236 [# 51364]
Routine Number	[R 10] : 0 [# 0]
Code Segment	[R 11] : 152 [# 230]
Stack Segment	[R 12] : 1 [# 1]
Breakpoint register	[R 13] : -1 [# -1]
Exc handler code segment	[R 14] : 152 [# 230]
Exc handler global pointer	[R 15] : 2562 [# 5002]
Encode overlay # and entry	[R 16] : 3 [# 3]
VM status	[R 110] : 1280 [# 2400]

<E>'p If <E> is an address within the code of any routine, then display the routine dictionary of that routine. The routine dictionary has the following format:

RDPS: Size of parameters
RDRPS: Size of result + parameters
RDLTS: Size of locals + temporaries
RDLL: Lexcial Level

RDEntry: Entry address relative to segment
RDExit: Exit address relative to segment

<E>'P If <E>=0, then show the code segment base address (CB) and the global pointer (GP) of each module. Otherwise, show the segment header block for module <E>. The time stamp attached to a file name indicates the last time the file was updated. For example, the command Test3P might display the following information:

```
Program TEST3 [ Compiled at: 11 Apr 83 11:33:29 ]
Src file = <boot>user>bob> test3.PAS
           [ 20 Aug 82 10:53:01 ]
QVersion = 3
GDB size = 184 words
Version =
Comment =
QMapFile = <boot>bob> test3.QMAP
           [ 11 Apr 83 11:33:29 ]
SymFile = <boot>user>bob> test3.SYM
           [ 11 Apr 83 11:33:29 ]
Imports   3 files: (Import info at block 2)
          TEST3A from test3a.PAS
          WRITER from WRITER.PAS
          STREAM from STREAM.PAS
```

- 'q Terminate Mace and return to Kraut command level.
Note: This command should only be executed on Mace command level.
- 'r Resume execution of the target process.
- 's Suspend execution of the target process.
- <D>'t Show the last <D> routine calls starting at the top of the stack.
- <D>'u Show the contents of micro state register <D>.
- <D>'U <V> Assign the value <V> to micro state register <D>.
- 'x Toggle Mace debug flag (for debugging purposes only).

- 'X Toggle Expression trace flag (for debugging purposes only).
- '(a Low, low level debugging aids (type '(a? for help if you are really desperate).

6. Coping with Debugger Bugs

Kraut is still actively being developed and an internal error might show up while you are debugging your own program. If the error is an exception defined in "except.dfs", the debugger writes out an explanation such as DIVISION BY ZERO, otherwise the exception is written out as a segment and routine number-pair.

You have one of the following options:

- RECOVER (Default): Try to return to command level.
- DEBUG: Recursive debugging mode. You are prompted for a new window to debug the debugger.
- QUIT: Quit the session.

The following command is intended for debugging the debugger.

Verbose Set debugging switches interactively. The switches can also be set initially by a file M.switches, where M is the name of the file containing the main program.

If you run across an uncaught exception in the debugger, it would be helpful if you could do the following:

- Recover from the bug.
- Turn on the switches with the Verbose command.

- Repeat the command that caused the exception.
- Enter the recursive debugging mode and print the call stack of the current state of the debugger.
- Save both transcript files and send them to bob@CS-SPICE (Use the Maintainer and Report commands).

7. Current Restrictions and Known Bugs

- It is not possible to *Open* modules which are not imported by one of the modules of the program.
- *Call chains* are not yet implemented¹⁵.
- *Consistency checks* between source and object files are not implemented.
- The predicate `_ UNDEFINED` (see page 9) is not implemented.
- The *general notation* $M''P_1' \dots 'P_{n-1}'P_n,i$ denoting variable i in routine P_n nested in P_{n-1}, \dots , nested in P_1 , which is declared in module M has not yet been implemented. Only $P'Foo$ to denote the variable Foo in routine P and $M''P'Foo$ to denote the variable Foo in routine P in module M are implemented.
- Because of restrictions in the symbol table, it is not possible to access *record fields* by name. Word (= 16 bit) offsets relative to the base address of the record must be used instead. To determine word offsets, you have to know how the compiler allocates storage for record fields. For example, in the following record definition

```
Var      Rec1 : record
          Rec2: record
                  i : integer;
```

```
Rec3: record
      p : long;
      foo: ^integer;
    end;
    end;
  b: boolean;
end;
```

the field Rec1.Rec2.Rec3.foo^ is denoted by the expression Rec1.0.1.2^, where Rec1.0 is the offset of field Rec2) in record Rec1, etc. Note that the compiler allocates field list identifiers separated by commas in the reverse order in which they are mentioned in the declaration. For example, in the following definition

```
Var     Rec1 : record
        i, j,k,l,m,n : integer;
      end;
```

the expression Rec1.0 denotes the field Rec1.n and Rec1.4 denotes Rec1.k. The above examples are only valid for unpacked records. Ask a compiler hacker if you want to examine packed records!

- *Array elements* are not fully supported: Because there is no symbolic information available for the bounds of arrays, the debugger assumes 0 as the lower bound for array variables. Thus for arrays starting on a different lower bound, the lower bound has to be normalized to 0. Multidimensional arrays are not supported.
- *Named Constants* (defined with Const) are not supported.

- *Display of Records:* Records are displayed on a rather low level. Kraut calls Mace to successively interpret each word of the structure as an integer, a long (using two words), two bytes, two characters or the start of a string. The number of words to be displayed can be specified or a default value is assumed: For records the number of words necessary to store the record and for arrays the size of the subcomponent type. For example, given the program fragment

```
Var      grec : record
          int:  integer;
          bool: boolean;
        end;
...
grec. int:= 11111;
grec. bool := true;
```

grec can be displayed as follows:

```
|grec
| Record!    [TEST3 ]
| Display how many words? [2] 1_
| TEST3.G172:
|   Integer Long  LSB  MSB Char
|   11111. 76647. 103. 43. | g + |
|
|   String [ 15 ]
| [ 103 ]+##### R#R ##"
```

The next command (<CR>) can be used to display successive words:

```
|<CR>
| TEST3.G173:
| 1. 131073. 1. 0. | A A @ |
|
| [ 1 ]="#"
```

```
|<CR>
|TEST3.G174:
| 255. 65791. 255. 0. lxxA(w)|
| [ 255 ]##### R#R ##"
```

The pair XX is used to indicate a byte which cannot be interpreted as an Ascii character.

- Because there is no symbolic information for enumerated types, sets are displayed as bit sequences. The leftmost bit refers to the first literal, and the n'th bit from the left to the n'th literal of the set. For example, given the program fragment

```
type
  color = (red, green, brown, yellow, black, pink);
var
  cset: set of color;
  ...
  cset := (red, green, pink);
```

the variable cset will be displayed as

```
cset
TEST3.G200:
(bits 0..15) 1100010000000000
```

- *User defined enumerated types* must be denoted by their integer equivalent.
- *String search* does not work correctly for files that contain one or more Include files.

Appendix A.Kraut Command Summary

A.1. Breakpoints

See Page 22.

Break [after] Rl# : Set breakpoint at [after] routine R or line #.
DBreak Rl# : Delete breakpoint at routine R or line #.
DBreak * : Delete all breakpoints.

A.2. Constant syntax

See Page 7.

Any Pascal constant is a valid constant. DECIMAL Pascal constants in the range -32768..+32767 are of type integer (16 bits), otherwise of type long (32 bits). Constants in a different radix have to be prefixed by a #, a radix indicator (B,D,O,H) and a size indicator (I,L).

(B =binary,D=decimal,O=Octal,H=hexadecimal, I=integer,L=long). The default radix is D, the default size I.

Examples:

#	OL4712347	32 bit octal
#	LHABFD4CD9	32 bit hexadecimal
#	b10101011	16 bit binary (default: I)
#	bl10101011	32 bit binary
	3434	16 bit decimal
	9896966	32 bit decimal

A.3. Declarations

See Page 24.

Counter C (:= l) : Define Counter C and initialize it to 0 (to l).
DCounter C : Delete Counter C.
Define Function F : Define path function F.

A.4. Editor Commands

See Page 23.

Around (i F) : Show 20 source lines around current line.
(around line i in file F).
Grep 'S' (F) : Look for all occurrences of string S.
starting at current line
(starting at line 1 in F).
Scroll (i F) : Show next 20 source lines from current line
(from line i in file F).
(Type) i (j F) : Show source lines i (to j in file F).
'S'(l) F : Search forward in file F for string S.

A.5. Identifier Syntax

See Page 7.

Identifier Foo (following Pascal's
scope rules)
Proc'Foo Identifier Foo in routine Proc
Mod"Proc'Foo Identifier Foo in routine Proc in
module Mod
Identifier Foo into type T,
where T can be any type qualifier.

A.6. Inspection Commands

See Page 27.

& : Value of last command.
Address (ID) : Get address of last variable (of ID).
Compute E : Compute expression E.
Current R : Move down in the run time stack to R and
make it the current routine.
Globals (M) : Show all globals of current module

Locals (R) : Show all variables of current routine (of module M).
Parameters (R) : Show all parameters of current routine (of routine R).
Radix N : Set the current radix to N where N is a DECIMAL (!) number in { 2,8,10,16 }. Initially the current radix is 10.
V(=) : Type the value of V (' = ' is needed, if V is also a command).
VAR:= ID : Assign the value of ID to VAR.
Write(p1, ...,pn), Writeln(p1, ...,pn): Write the Pascal expressions p1, ...,pn (and CRLF).

A.7. Line Number Syntax.

. Current line in current source file.
24 Line 24 in current file.
24 test3 Line 24 in file test3.pas;1.
24 test3.pas;2 Line 24 in file test3.pas;2.
24 sys>accent> test3 Line 24 in file sys > accent> test3.pas;1.

A.8. Mace

See Page 41.

Type MACE to enter the Mace interpreter. Type ?? to Mace for further help. Mace commands can also be typed to Kraut: Type \$M to the Kraut interpreter to execute the mace command M.

A.9. Miscellaneous

See Page 32.

@ FILE.kmd : Execute default command file (FILE.kmd).
\$M : Execute Mace command M.
- - - : Start of a comment.
Current M : Set current module to M (same as OPEN M).
Exception : Show the current exception.
Expert (ON) OFF : Expert switch. (Default: OFF).

Indent N : Concatenate N blanks with the prompt sign.
Shell : Create another shell with current environment.

A.10. Model Commands

See Page 38.

(Model) Bargraph B MX S : Define (or rescale) bargraph B with maximum value MX and scale S (Linear or Log).
(Model) Routine M R : Define a model M and bind the routine R to it.
Bind ID M : Bind variable ID to model M.
DModel M : Delete model M.
DBind ID : Delete ID's binding to a model.

A.11. Next Command

See Page 28.

<CR> : Execute the previous command with new arguments. Implemented for \$, Around, Calls, DownStack, Search, SingleStep, Type, UpStack.

A.12. PathActions

A path action is declared by the keyword Action followed by the identifier of the path expression with which the path action is to be associated. The keywords Match and NoMatch are followed by the actions to be taken in case of a match and mismatch, respectively. If an action contains more than one command, the commands have to be separated by '~'. Either the Match or the MisMatch part or both can be omitted, in which case no action will be associated with the missing part.

Example: ACTION LOOP MATCH WRITELN("I = ", I) >> LoopWindow NOMATCH HALT

associates the debugger commands Halt and Writeln("I = ",I) >> LoopWindow with the path expression Loop: Whenever the evaluation of Loop yields a violation of the specified ordering the computation is suspended and control is turned over to the debugger, otherwise the value of I is written into the window LoopWindow.

NOTE: The generalized path expression Loop is only evaluated when one of its path functions is executed.
kp}

```
First;'); |EDIT __B1 | CONDITION [] : i = 1000 |
MATCH [] : Callstack 4~locals~writeln('i = ',
i)~go | NOMATCH [] : | ENABLE [] : TRUE |
VERBOSE [] : TRUE kp}
```

A.13. Reporting Bugs

Maintainer	(M) : Get name of maintainer of all modules (of module M).
News	: Print any news about new features, known bugs, etc that are not undocumented in the Spice manual.
Report A@	M : When QUITting the debugger, the script file will be given to the Spice Mail system to mail it to address A at machine M.
Script ON	OFF : Set transcript switch on or off (default: on).
Verbose	: Set various debugging switches (For internal debugging).
Version	: Get the current version of Kraut.

A.14. Runtimestack Commands

See Page 29.

BottomStack : Move to bottom of stack.
Calls (#) : Show current call sequence(# levels).
DownStack (#) : Move backward 1 (#) routine(s) in dynamic call chain.
TopStack : Move to top of stack.
Unwind : Remove last Executed routine from stack.
UpStack (#) : Move forward 1 (#) routine(s) in dynamic call chain.

A.15. Show Command

Show P : P can be of the following values:
P = Breaks : Show the current break point list.
P = Counters : Show the currently defined counter variables.
P = Models : Show the current models.
P = Modules : If M is omitted, then type KRAUT's module list. Otherwise give some information about module M. This includes date of compilation, names of seg, qmap and sym files, names of imports, etc.
P = Pathfunctions M: Show the user defined path functions for module M. If M is omitted, show them for all modules.
P = PathRules P1 : If P1 = 'active', then display the list of enabled pathrules, if P1 = 'inactive' then display the list of disabled path rules, other display all currently defined path rules.
P = Routines M : Show the routines and user defined path functions for module M. If M is omitted then show them for all modules.
P = Runfile R M : Dump the contents of run file R into file M.
P = Searchlist : Display Accent's file search list.
P = Windows : Display the currently defined

windows.

A.16. Searchlist Commands

See Page 20.

ChDir D : Set current directory to D. If D is missing,
 show current directory.
Close M : Close module M (* for all modules).
Open M : Open module M (* for all modules).
SetSearch L : Modify file search list : - (pop), ID (push).
System : Open Pascallnit and all its imports.

A.17. Target Process Commands

See Page 30.

If the target process is running, the following commands and control characters can be typed. Note that control characters have to be typed in the window of the debugger, not that of the target process:

^ DEL t : Show the state of the target process. (Not yet implemented)
^DEL c, ^DEL k, ^ DEL d : Return to command level.
Halt : Suspend execution of target process.

If the target process has stopped the following commands can be executed:

Execute P(p1, ...pn), P(p1,...pn) : Execute routine R with parameters p1, ...,pn. R can also be the main program. Omit closing ')' for parameter check.
Go : Resume execution of target process.
Run (A) : Start execution of target process (with arguments A).
TerminateTarget YES! NO : Do or do not terminate target process on QUIT.

A.18. Trace Commands

See Page 21.

Trace [Before | After] R [C] : Trace routine R (only its entry or exit) and perform commands C.
If R is a module set traces for all of the routines of R.

DTrace [Before | After] R : Delete trace of R (only entry or exit). If R is a module, delete all traces for module R.

A.19. Type Qualifier

See Page 8.

Array [n]	array (n words)		Pointer	pointer
Boolean	byte (8 bit)		Record [n]	record (n words)
Char	char		REal	real
Integer	integer (16 bit)		String	string
Long	long (32 bit)		SEt[n]	set (n words)

A.20. Window commands <

See Page 37.

Window W LX TY W H : Create window W with upper left corner LX,TY and W characters per line and H lines.

Window W : Move to W if it exists, otherwise create window with default coordinates.

DWindow W : Delete window W.

DWindow * : Delete all existing debug windows except for the current one.

C >> W : Redirect output for command C into window W. Window W is created with default coordinates, if it does not yet exist.

Notes

¹The sym file contains the names of the variables and routines declared in the program. The qmap file contains a mapping between qcode offsets in the seg file and the corresponding source statements.

²**DEBUG** expects the name or the number of the target process as parameter, which can be found with the shell command Sys.

³File lookup is always done with Accent's file search list.

⁴Note that the least significant bit is printed out first.

⁵A note concerning the double quote: It is not possible to use only one quote because of the possibility of name conflicts. For example, in the following program

```
program foo;
var i : integer;

procedure foo;
var i : integer;
...
```

the name foo'i could either mean the global i in module foo, or the local variable i in routine foo.

⁶If these counter names conflict with names of objects in the target process, they have to be prefixed with the escape character '—'.

⁷If a path function P is mentioned without counters or predicates the defaults $\text{__Act}(P) \geq 0$ and $\text{__Term}(P) \geq 0$ are assumed.

⁸Note that the generalized path expression Loop is only evaluated when one of its path functions is executed

⁹Note that the character “ ” is actually the character “ ” on your keyboard

¹⁰M.kmd is the default command file. If it can be found with the initial file search list, it is automatically executed at the beginning of the debugging session.

¹¹Note that any single digit will be interpreted as a source line number of the TYPE command.

¹²If BargraphWindow does not yet exist, you are prompted for the screen locations.

¹³The activation pointer (AP) of a routine points to the address of its ACB. See 'm command.

¹⁴The current routine can be changed with the Kraut runtime stack commands UpStack, DownStack, etc.

¹⁵However, one can access allocated but unvisible variables by positioning with TopStack, BottomStack, DownStack and UpStack followed by Locals.

Index

- \$ command 32
- & command 33
- 'S'? command 23
- command 33
- := command 28
- <CR> command 28
- = command 27
- (ω) command 32
- 'S'! command 24
- Abbreviations 19
- Act counter 14
- Action command 25
- Address command 33
- Around command 23
- Backward search command 23
- Bargraph 38
- Basic path expressions 13
- Bind command 39
- BottomStack command 29

Break command 23
Breakpoint commands 22
Bugs 1

Calls command 29
Changing directory 20
ChDir command 20
Check expressions 15
CheckPath command 25
Close command 21
Command language 19
Command sequence 19
CommandLanguage 1
Comment 33
Compute command 33
Copying 1
Copying with debugger bugs 49
Counter 14
Counter command 24
Create window command 37
Current command 33
Current restrictions 51

DBind command 40
DCounter command 24
Define Function command 24
Definition commands 24
Delete window command 37
Disable command 25
DModel command 40
DownStack command 29
DTrace command 22

Editor commands 23
EditPathrule command 26

Enable command 26
EXCEPTION command 33
Execute command 30

Find expressions 15
FindPath command 26
Forward search command 24

Generalized path expressions 13
Getting Started 1, 2
Globals command 27
Go command 32
Grep command 24

Halt command 32
Help command 34

Indent command 34
Introduction 1

Known bugs 51

Locals command 27

Mace 1, 41
Mace command 34
MaceCommands 44
Maintainer command 34
Manipulating path rules 17
Match 16
Mode qualifier 41
Model command 38
Model commands 38

Naming 1, 7

News command 34
Next command 28
NoMatch 16

Open command 20

Parameters command 27
Path actions 16
Path function 13
Path rule commands 25
Path rules 13
Path variable 15
PathRules 1
PathRules command 27
Precedence of operators 42
Predicate 14
Primary value 42
Process Control Characters 30

Quit command 34

Radix command 27
Radix qualifier 41
Redirection of output 38
Remove Action command 27
Remove command 27
Report command 34
Run command 32
Runtime stack commands 29

Script command 35
Scroll command 23
Search list commands 20
SetSearch command 21
Shell command 35

- Show command 36
- Show modules command 21
- Silence command 35
- System command 21

- Target process control 30
- Term counter 14
- TerminateTarget command 32
- TopStack command 29
- Trace command 21
- Trace commands 21
- Transcript of Debugging session 4
- Type command 23

- Unwind command 32
- UpStack command 29

- Variable inspection commands 27
- Version command 37

- Window commands 37

PERQ Systems Corporation
Accent Operating System

Kraut Debugger
Index

MATCHMAKER: THE ACCENT REMOTE CALL PROCEDURE LANGUAGE

December 7, 1984

Copyright © 1984 PERQ Systems Corporation
2600 Liberty Avenue
P.O. Box 2600
Pittsburgh, PA 15230
(412) 355-0900

Accent is a trademark of Carnegie-Mellon University. Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its Spice Project.

The system described in this document is based upon the Matchmaker program by Michael B. Jones of CMU and Joseph Ginder, Ellen Colwell, and Edward Pervin of PERQ Systems. This document is adapted from two papers by Michael B. Jones, *Matchmaker: A Remote Procedure Call Generator*, Carnegie-Mellon University, Pittsburgh, PA, 1983, and *Matchmaker: A Language for Remote Procedure Calls*, Carnegie-Mellon University, Pittsburgh, PA, 1984.

This document is not to be reproduced in any form or transmitted in whole or in part, without the prior written authorization of PERQ Systems Corporation. The information in this document is subject to change without notice and should not be construed as a commitment by PERQ Systems Corporation. The company assumes no responsibility for any errors that may appear in this document. PERQ Systems Corporation will make every effort to keep customers apprised of all documentation changes as quickly as possible. The Reader's Comments card is distributed with this document to request users' critical evaluation to assist us in preparing future documentation.

PERQ, PERQ2, LINQ, and Qnix are trademarks of PERQ Systems Corporation.

Table of Contents

	Page
1. Theory	MM-1
1.1. What is Matchmaker?	MM-1
1.2. What Does It Do?	MM-2
1.3. A Declarative Language for Data Structures	MM-2
1.4. A Declarative Language for Remote Procedure Calls	MM-2
1.5. Goals	MM-3
1.6. Example	MM-3
2. General Language Structure	MM-5
2.1. Lexical Structure	MM-5
2.2. Kinds of Matchmaker Specifications	MM-5
2.2.1. Types specifications	MM-6
2.2.2. Interface specifications	MM-6
2.3. Order of Declarations	MM-6
2.4. Expressions	MM-7
2.4.1. Primitive expressions	MM-7
2.4.2. Expression operators	MM-7
3. Data Declarations	MM-9
3.1. Constant Declarations	MM-9
3.2. Type Declarations	MM-9

3.2.1. Examples	MM-10
3.2.2. Built-in types	MM-10
3.3. Use Declarations	MM-11
 3.3.1. Syntax	MM-11
 3.3.2. Examples	MM-11
4. Defining New Data Types	MM-13
 4.1. Record Types	MM-13
 4.1.1. Syntax	MM-13
 4.1.2. Packed record types	MM-13
 4.1.3. Examples	MM-14
 4.1.4. Restrictions	MM-14
 4.2. Array Types	MM-14
 4.2.1. Syntax	MM-14
 4.2.2. Packed array type	MM-15
 4.2.3. Examples	MM-15
 4.2.4. Restrictions	MM-15
 4.3. Enumeration Type	MM-15
 4.3.1. Syntax	MM-16
 4.3.2. Examples	MM-16
 4.3.3. Restrictions	MM-16
 4.4. Pointer Types	MM-17
 4.4.1. Syntax	MM-17
 4.4.2. Examples	MM-17
 4.4.3. Restrictions	MM-17
 4.5. Union Types	MM-17
 4.5.1. Syntax	MM-18
 4.5.2. Examples	MM-18
 4.5.3. Restrictions	MM-18

5. Matchmaker Calls	MM-19
5.1. Classes of Matchmaker Calls	MM-19
5.1.1. Remote_Procedure calls	MM-19
5.1.2. Message calls	MM-20
5.1.3. Server_Message calls	MM-21
5.1.4. Alternate_Reply calls	MM-22
5.2. Call Arguments	MM-22
5.2.1. Data arguments	MM-23
5.2.1.1. Argument directions	MM-23
5.2.1.2. Normal argument types and parameters	MM-23
5.2.1.3. Variable-size array type parameters	MM-24
5.2.1.4. Union type parameters	MM-25
5.2.2. Special arguments	MM-26
5.3. Defaults and Options	MM-27
5.3.1. Modifying global call defaults	MM-27
5.3.2. Modifying local call defaults	MM-28
5.3.3. Listing of defaults	MM-28
5.3.4. Non-call argument options	MM-28
5.3.5. Examples	MM-30
5.4. Message ID Declarations	MM-30
5.4.1. Skip_ID declaration	MM-30
5.4.2. Next_ID declaration	MM-30
6. Using Matchmaker	MM-33
6.1. Using a Generated Interface	MM-33
6.1.1. Using Matchmaker user code	MM-33

6.1.2. Using Matchmaker server code	MM-34
6.2. Matchmaker File Names	MM-35
6.3. Matchmaker Command Line Syntax	MM-36
7. A BNF for the Matchmaker Language	MM-39
7.1. Interface and Options Definitions	MM-39
7.2. Datatype Definitions	MM-40
7.3. Message Definitions	MM-42
7.4. Expression Syntax	MM-43
7.5. Lexical Definitions	MM-44
8. Information Specific to Each Code Generator	MM-45
8.1. Pascal	MM-45
8.2. C	MM-45
8.3. Lisp	MM-46
8.4. Substitute-typedef	MM-47
9. Example Source and Matchmaker Files	MM-49
9.1. Source Files	MM-49
9.1.1. Typescript.mm	MM-49
9.1.2. TS.mm-pas	MM-51
9.2. Matchmaker-Produced Files	MM-51
9.2.1. TSDefs.pas	MM-51
9.2.2. TSUser.pas	MM-51
9.2.3. TSServer.pas	MM-74
9.2.4. TSDefs.slisp	MM-91
9.2.5. TSUser.slisp	MM-92
9.2.6. TSMsgDefs.slisp	MM-108

9.2.7. TSUInit.slisp
9.3. Server-Supplied Code

MM-122
MM-127

List of Tables

Page

Table 1: Special Arguments, Defaults and Legal Values	MM-29
--	--------------

1. Theory

1.1. What is Matchmaker?

Matchmaker provides a mechanism for declaratively defining a procedural interface for message-based communication between processes. This type of procedural interface to message-based communication is usually called a remote procedure call. Remote procedure calls in Accent can involve processes written in different languages; Matchmaker supports this capability. Once an interface has been declared, Matchmaker may be used to generate procedures for sending and receiving messages between processes written in any of the supported languages. Matchmaker does all the work of appropriately packing the procedure arguments into messages and extracting incoming procedure arguments from message fields.

Matchmaker allows a programmer to write a server process and declare the types of all data to be exchanged between the server and its client processes. Based on those types, procedural interfaces are declared for sending data between processes in messages. A client process would call a Matchmaker-generated procedure to make use of a server facility. This procedure would pack the procedural parameters into a message and send the message to the server process. The server process would receive the message, provide its service, and pack return data into a message to be sent back to the client process. The client process would return from the original procedure call with the data sent back from the server. Remote procedure call interfaces simplify and increase the reliability of writing message based code.

1.2. What Does It Do?

Matchmaker, a specialized compiler, accepts its own language and produces code for multiple target "machines." The target "machines" for Matchmaker code generators are standard programming languages; currently available languages are C, Lisp, and Pascal.

The result of a compilation is source code for each target language. This code implements the same message interface to the declared service, independent of the language with which it will be used.

1.3. A Declarative Language for Data Structures

Matchmaker defines a set of data type primitives and constructors for declaring types and values and a set of rules for representing these declared types. For any Matchmaker type, the Matchmaker code generator for each language can generate a native type declaration for the desired representation.

Any data types that are to be used by more than one language should be declared in the Matchmaker language. Then Matchmaker can be used to generate the appropriate types for each language.

1.4. A Declarative Language for Remote Procedure Calls

Matchmaker simplifies message passing by providing a set of message-based procedural linkages between processes. A number of classes of remote procedures can be declared. There are enough to cover almost all interactions between cooperating processes.

1.5. Goals

The main goals of Matchmaker are:

1. to provide a self-contained language for specifying message level interfaces between cooperating Accent processes;
2. to provide a language rich enough to express any data structure that can be both efficiently represented in a message and reasonably represented in all target languages; and
3. to generate efficient code.

1.6. Example

In PERQ Pascal, sending a message to a server containing a 32-bit integer and a port requires a hand coded program like this:

```
Function SendIt(ServPort : Port; I : Long;
                P : Port ) : GeneralReturn;
begin
  type
    MyMessage = record
      head      : Msg;
      IPCNam2  : TypeType;
      Arg2     : Long;
      IPCNam3  : TypeType;
      Arg3     : Port;
    end;

    var
      MyMsg      : MyMessage;

    begin
      with MyMsg.head do
        begin
          SimpleMsg := false;
          MsgSize := WordSize(MyMsg)*2;
          MsgType := NORMALMSG;
          RemotePort := ServPort;
          LocalPort := ReplyPort;
          ID := 10000;
        end;
      with MyMsg do
        begin
          IPCNam2.LongInteger := #2000220002;
          Arg2 := I;
          IPCNam3.LongInteger := #2000220006;
        end;
    end;
end.
```

```
    Arg3 := P;
  end;
  SendIt := Send(MyMsg.head,0,WAIT);
end;
```

An equivalent piece of code could be generated with these
Matchmaker declarations:

```
Interface foo = 10000;

Message SendIt( : port; I : Long; P : Port) : GR_Value;
```

2. General Language Structure

2.1. Lexical Structure

Identifiers

Identifiers are represented as a string of letters, digits, and underscores. The first character may not be a digit. Case of identifiers is preserved as in the declaration, but identifiers are matched with case folded.

Integers

Decimal integers are represented as a string of decimal digits. Octal integers are represented as a '#' character followed by a string of octal digits.

Strings

Character strings are enclosed in double quotes (" "). The literal double quote is represented by two double quotes. Strings cannot span lines.

Characters

Character constants are surrounded by single quotes (' '). The literal single quote is represented by a string of four single quotes. (Character constants are not implemented yet.)

Comments

Comments may be introduced at any lexical break with the exclamation point character (!) and continue to the end of the line.

2.2. Kinds of Matchmaker Specifications

There are two kinds of Matchmaker specification files: Types and Interface.

2.2.1. Types specifications

Types specifications contain data structure declarations, but no code declarations:

```
Types <SpecificationName>;
```

where `<SpecificationName>` is an identifier for the specification.
A Types specification must end with the keywords End Types.

2.2.2. Interface specifications

Interface specifications contain data structure declarations and code (call) declarations:

```
Interface <SpecificationName> = <MsgIDbase> ;
```

where:

`<SpecificationName>` is an identifier for the specification.

`<MsgIDbase>` is an expression that names the message ID that is to be allocated to the first call declared.

Interface specifications must end with the keywords End Interface.

2.3. Order of Declarations

The declarations in all Matchmaker source files (.mm) must be made in this order:

1. Declare Interfaces or Types: Give name of specification, and, for Interface declaration, message ID origin. *Required*.
2. Declare Options: Give values for various options. *Optional*.
3. Declare Data: Can be Constant, Type, or Use declarations. *Optional*.

4. Declare Calls: Used for declaring message-based calls. *Required* for Interface specifications; *not allowed* for Types specifications.
5. End Interface or End Types: End Interface for Interface specifications, or End Types for Types specifications. *Required*.

2.4. Expressions

Typed constant values are used in many different contexts throughout the Matchmaker language. Anywhere a literal constant (such as #400, "Carp", or 'a') can be used, a compile-time constant expression can also be used. Since all Matchmaker expressions are evaluated at the time the specification is compiled, compile-time constant expressions will be referred to as "expressions."

2.4.1. Primitive expressions

The primitive building blocks for Matchmaker expressions are:

Literals

Any integer, string, or character literal

Boolean Literals

The pre-declared identifiers "True" or "False"

Constant Names

The identifier for any value declared in a "Constant" declaration

Enumeration Names

The identifier for any element of an enumerated type

2.4.2. Expression operators

The operators for combining expressions, in decreasing order of precedence, are:

(...)	Specify evaluation order
* / Mod	Multiplication, Division, Modulus
+ -	Addition, Subtraction (Either may be unary)
= <>	Equal, Not equal

> >= <= < Order operators
Not Boolean negation
And Conjunction
Or Disjunction

3. Data Declarations

This chapter will cover the means of declaring typed values and data types in Matchmaker specifications.

3.1. Constant Declarations

Symbolic constant values may be declared to Matchmaker using the Constant keyword, followed by a list of declarations of the form:

Name = Expression ;

The two boolean values True and False are pre-declared in every Matchmaker specification. Their declarations are equivalent to the declarations:

Constant

True is the same as 0 = 0;
False is the same as 1 = 0;

3.2. Type Declarations

Named types may be declared to Matchmaker using the Type keyword, followed by a list of declarations of the form:

Name = Type Specification ;

Type Specification may either be a built-in data type, a previously declared type name, or a new type definition. For more information on new data type definitions, refer to Chapter 4.

3.2.1. Examples

Some examples of Type declarations are:

```
Type
  Integer      = Short;
  Bit23       = unsigned[23];

  Complex      =
    record
      Re         : Real;
      Im         : Real;
    end record;

  NewBoolean   = (No, Yes);

Type
  File_Data    = array [*] of Byte;
```

NOTE: * is the token for a variable-sized array.

3.2.2. Built-in types

Matchmaker provides several types and classes of types which are pre-declared for each specification. They are:

Boolean A one bit logical quantity.

Character An eight bit character type.

Signed[n] An n bit signed integer. n defaults to 16 if the [n] is not specified.

Unsigned[n] An n bit unsigned integer. n defaults to 16 if the [n] is not specified.

n .. m A subrange type of the integers n to m inclusive.

PERQ_String[n]

An n byte string prefixed by a length byte and padded to a 16-bit boundary. n defaults to 80 if [n] is omitted.

Port, Port_Send, Port_Receive, Port_Ownership, Port_All
Message communication ports, with associated port rights. Port and Port_Send are synonyms.

Real A 32 bit IEEE floating point numeric type.

Byte A pre-declared 8 bit unsigned integer type.

Short A pre-declared 16 bit signed integer type.

Long A pre-declared 32 bit signed integer type.

3.3. Use Declarations

The Use declaration allows one Matchmaker specification to use data structure declarations from another. Constant, Type, and Use declarations can appear in any order with respect to one another.

3.3.1. Syntax

The syntax for the Use declaration is the keyword Use followed by specifications of the form:

InterfaceName from "FileName" ;

where InterfaceName is the declared interface name in the specification to be used by the current specification, and FileName is the filename for that specification without the ".mm" extension. (Language-specific declaration file names will also be derived from FileName).

The "use" chain is recursively expanded. In other words, if an interface A uses declarations from interface B, and B uses interface C, then A may also use declarations from C.

3.3.2. Examples

Some examples of Use declarations are:

```
Use
  Foo from "Foo";
  Shoes from "Shoes";
Use
  AccInt from "Accent";
```


4. Defining New Data Types

Matchmaker allows construction of new data types from existing ones. Matchmaker is a language for specifying message-based interfaces between processes; therefore only those types that can be efficiently sent in messages are supported.

In some cases, types that are too complicated for Matchmaker to parse can still be defined using the command Substitute-typedef (see Section 8.4) in auxiliary files.

4.1. Record Types

A Record type is a group of one or more "fields" made up of an identifier and an associated data type.

4.1.1. Syntax

A Record type specification contains the keyword Record and one or more fields of the form:

FieldName : FieldType ;

followed by the keywords End Record. Each FieldName must be a valid identifier, and each FieldType must be a declared type name or built-in type.

4.1.2. Packed record types

"Packed" record types are like ordinary record types, except that attempts are made to "pack" several small fields together where possible, instead of always aligning them on 16-bit boundaries. The syntax for packed record types is the same as that for ordinary record types, except that the type keyword is "Packed Record."

4.1.3. Examples

Some examples of Record specifications are:

```
Type
  Couple =
    record
      Male    : Person;
      Female  : Person;
    end record;

  Time_Rec =
    packed record
      Hours   : 0 .. 23;
      Minutes : 0 .. 59;
      Seconds : 0 .. 59;
    end record;
```

4.1.4. Restrictions

Due to message passing restrictions, record field types must not be port types, pointer types, variable-sized array types, or union types.

4.2. Array Types

An Array type is a collection of some number of elements, all of the same data type. The number of elements may be fixed or may vary.

4.2.1. Syntax

An Array type specification is of the form:

Array [size] of ElementType

where ElementType is a declared type name or built-in type, and size is either the token * for a variable-sized array or an integer expression giving the number of elements for a fixed-size array.

4.2.2. Packed array type

The "packed" array type is like the ordinary array type, except that attempts are made to "pack" several small elements together where possible, instead of always aligning them on 16-bit boundaries. The syntax for the packed array type is the "Packed Array" keyword.

4.2.3. Examples

Some examples of Array specifications are:

```
Type      Triple      = array [3] of Note;  
  
Type      Byte_Array  = packed array [*] of Byte;
```

4.2.4. Restrictions

Array elements must not be of pointer, variable-size array, or union types. Variable-size arrays can be declared; however, only pointers to the arrays can be directly passed in messages, along with a size parameter giving the number of elements being passed. The number of elements must be positive (greater than zero) for fixed-size arrays.

4.3. Enumeration Type

The Enumeration type provides a means of naming a set of related values and grouping those values together into a new type. The value of each name, if not specified, is provided by Matchmaker, beginning with zero and incrementing by one for each successive name.

4.3.1. Syntax

An Enumerated Type specification consists of a pair of parentheses enclosing a comma-separated list of enumeration elements of either the form:

ElementName

or:

ElementName = ElementValue

ElementNames without specified values are assigned sequential element values. Both kinds of elements may be mixed, as long as no value or name is duplicated.

4.3.2. Examples

Some examples of Enumeration specifications are:

```
Type
    Tri_State = (Open, Closed, Floating);

    Return_Values =
    (
        OK = 1,
        Information,
        Take_Heed,
        Warning = 99,
        Error = 1000,
        Fatal_Error = 10000,
        Internal_Error = -1
    );
```

4.3.3. Restrictions

All enumeration type element values must be expressible as 16-bit signed integers.

4.4. Pointer Types

Pointer types allow Matchmaker to pass a message by reference instead of directly including it into the message body. This is useful when passing variable-size arrays between processes, and occasionally for passing large fixed-size structures. (Do not use pointers to small structures, since at least one page of data is always passed regardless of the actual size.)

4.4.1. Syntax

A Pointer type specification is of the form:

`^ BaseType`

where `BaseType` is a valid type specification.

4.4.2. Examples

Some examples of Pointer specifications are:

```
Type
  Bytes      = ^ Byte_Array;
  Ports     = ^ array [ * ] of Port;
  Block_Ptr = ^ Array [Block_Size] of Block_Elts;
```

4.4.3. Restrictions

Due to message passing restrictions, pointer base types must not be pointer types or union types.

4.5. Union Types

Union types provide a mechanism for passing data for which the IPC type number cannot be determined until run-time. This is useful only when the data is sometimes of a port type and sometimes a different port type, or is unstructured data. Only use a Union type when absolutely necessary.

4.5.1. Syntax

A Union type specification consists of a union head, of the form:

Union <TagType> of

followed by one or more fields of the form:

TagName : (FieldName : FieldType)

terminated by the keywords End Union. The TagType must be an integer, character, boolean, or enumeration type. Each TagValue must be an expression of type TagType. Each FieldName must be a valid identifier, and each FieldType must be a declared type name or built-in type.

4.5.2. Examples

Some examples of Union specifications are:

```
Type
  PortRights =
    union <integer> of
      TypePtOwnership : (PtO: Port_Ownership);
      TypePtReceive   : (PtR: Port_Receive);
      TypePtAll       : (PtA: Port_All);
      TypePt         : (Pt: Port);
    end union;

  Port_Or_Index =
    union <boolean> of
      False   : (Port_Index : long);
      True    : (Port_Value : port);
    end union;
```

4.5.3. Restrictions

Due to message passing restrictions, union field types must not be pointer types, variable-sized array types, or union types.

5. Matchmaker Calls

Matchmaker provides several different kinds of calls that can be made between processes. These calls vary in the way errors are handled and in the directions in which the messages are sent.

A Matchmaker call is declared in a manner similar to Pascal or Ada procedures and functions: the introductory keyword specifies the class of call being declared, the call name is given, and then a parameter list. The call is usually terminated by a type or keyword giving the return value for the call.

5.1. Classes of Matchmaker Calls

For each of the individual Matchmaker declarations, CallName is an identifier giving the name of the message-based call being declared, and ArgList refers to a semicolon-separated list of call arguments.

5.1.1. Remote_Procedure calls

Remote_Procedure calls generate code that allows a user process to send request parameters to a server and to receive reply parameters back from the server.

Remote_Procedure declarations are of the form:

`Remote_Procedure CallName (ArgList) : ValueType ;`

If ValueType is the keyword GR_Value then CallName is a function with result type GeneralReturn (signed[16]). This is a success/error code. If any errors occur sending or receiving the

messages for CallName, then CallName will return the send or receive error code.

If ValueType is the keyword No_Value then CallName is a procedure that does not return a value. If any errors occur sending or receiving the messages for CallName, then Matchmaker signals the send or receive error code in a language-dependent response.

If ValueType is neither GR_Value or No_Value then it must be a type name for a result type to be returned by CallName. In this case, CallName is a function returning a value of type ValueType. ValueType cannot be a variable-sized array or a union type. If any errors occur sending or receiving the messages for CallName, then Matchmaker signals the send or receive error code in a language-dependent response.

A RemotePort argument is required for all Remote_Procedure calls.

5.1.2. Message calls

Message calls generate code for a user process that sends a single message to a server without a reply.

Message declarations are of the form:

Message CallName (ArgList) : ValueType ;

If ValueType is the keyword GR_Value then CallName is a function with result type GeneralReturn (signed[16]) on the user side and a valueless procedure on the server side. When an error occurs sending the message for CallName, then CallName returns the send error code to the user. Otherwise it returns Success.

If ValueType is not GR_Value, then it must be the keyword No_Value. When ValueType is No_Value, then CallName is a procedure that does not return a value. If any errors occur sending the message for CallName, then Matchmaker signals the send error code in a language-dependent response.

A RemotePort argument is required for all Message calls. Only one-directional "In" parameters are allowed for Message calls (see Section 5.2).

5.1.3. Server_Message calls

Server_Message calls generate code for a server process that sends a single message to a user process.

Server_Message declarations are of the form:

`Server_Message CallName (ArgList) :ValueType ;`

If ValueType is the keyword GR_Value then CallName is a function with result type GeneralReturn (signed[16]) on the server side and a valueless procedure on the user side. If any errors occur sending the message for CallName, CallName returns the send error code to the server. Otherwise it returns Success.

If ValueType is not GR_Value, then it must be the keyword No_Value. When ValueType is No_Value, then CallName is a procedure that does not return a value. If any errors occur sending the message for CallName, then Matchmaker signals the send error code in a language-dependent response.

A RemotePort argument is required for all Server_Message calls. Only one-directional "In" parameters are allowed for Server_Message calls.

5.1.4. Alternate_Reply calls

Alternate_Reply calls generate code for a server process that sends a reply message back to a user process in response to a Remote_Procedure that was different than expected.

Alternate_Reply messages return values from exception conditions that occur during execution.

Alternate_Reply declarations forms are:

Alternate_Reply CallName (ArgList) ;

Alternate_Reply CallName ;

CallName is a language-dependent exception procedure that does not return a value.

A RemotePort argument is illegal for all Alternate_Reply calls. Only one-directional "In" parameters are allowed for Alternate_Reply calls.

5.2. Call Arguments

The Matchmaker call argument is:

ArgUsage ArgName : ArgType

where ArgUsage is a keyword specifying the argument to be used, ArgName is an identifier for the argument, and ArgType is the type of the argument.

ArgUsage can specify a direction for a data argument (for example, ArgUsage might be the keyword InOut), or it can be a keyword giving the special usage for the argument (such as the RemotePort keyword).

5.2.1. Data arguments

Most arguments to Matchmaker calls provide data that is passed between processes. Data arguments are of the form:

ArgDirection ArgParms : ArgType

5.2.1.1. Argument directions

ArgDirection specifies the direction(s) in which the data is sent. It may be one of In, Out, InOut, or may be omitted.

In Argument is only sent from the process initiating a call to the process handling the call.

Out Argument is only sent from the process handing a call back to the process that initiated a call.

InOut Argument is sent in both directions.

If ArgDirection is omitted, then In is assumed. Note: Some Matchmaker calls cannot accept Out or InOut arguments because they specify single-directional exchanges.

5.2.1.2. Normal argument types and parameters

ArgType is the name of a built-in or declared Matchmaker type appropriate for message passing. New types may not be implicitly declared when declaring calls.

When ArgType does not specify a variable-size array or a union type, then ArgParms is an identifier corresponding to that argument. This is used for the corresponding target language parameter name for languages that require them.

An example of simple call arguments is given by the declaration:

```
Remote_Procedure Example(
    RemotePort   Server : port;      ! (see 5.2.2)
    In          Put    : long;       ! In parameter
    Out         Get    : long;       ! Out parameter
    InOut        Both   : boolean;   ! InOut parameter
                           Implied : character ! In parameter
                           Gr_Value;
```

5.2.1.3. Variable-size array type parameters

When ArgType is the name for a pointer to a variable-size array (variable-size arrays cannot be passed; only pointers to them can be passed), then ArgParms must specify two parameter names. One is used to point to the array, and the other is used to dynamically indicate the number of elements in the array that is passed.

ArgParms for dynamic arrays can either be of the form:

PointerID [CountID]

or

[CountID] PointerID

The first is preferred. The only difference between the two forms is the order in which target language parameters for the pointer and count values are passed.

For example, the following Matchmaker declaration fragments:

```
type
  ByteVector = ` packed array [*] of byte;

Message Cattle(
  : port;          ! (see 5.2.2)
  in Bytes [Byte_Count] : ByteVector ! Dynamic array argument
  )
  : GR_Value;
```

would compile into the following C fragments:

```
typedef Byte ByteVector[];

GeneralReturn Cattle(ServPort, Bytes, Byte_Count);
  Port      ServPort;
  ByteVector Bytes;           /* Pointer to array */
  long      Byte_Count;       /* Element count */
  {
  ...
```

5.2.1.4. Union type parameters

When ArgType is the name for a union type, the ArgParms must specify two parameter names. One is the union tag (selector) value, and the other one passes the selected union field data.

Form of the ArgParms for union types:

DataID < TagID >

or

< TagID > DataID

The difference between the two forms is the order in which target language parameters for the field data and tag values are passed.

For example, the following Matchmaker declaration fragments:

```
type
  Port_Or_Index =
    union <boolean> of
      False   : (Port_Index : long);
      True    : (Port_Value : port);
    end union;

  Remote_Procedure Wow(
    : port;      ! (see 5.2.2)
    out PortOrIndex <WhichOne> : Port_Or_Index ! Union arg.
    )                      : No_Value;
```

would compile into the following Pascal fragments:

```
type
  Port_Or_Index =
    record case Boolean of
      False   : (Port_Index : Long);
      True    : (Port_Value : Port);
    end;

  procedure Wow(
    ServPort      : port;
    var PortOrIndex : Port_Or_Index;    { Data value }
    var WhichOne   : Boolean);          { Tag value }
```

5.2.2. Special arguments

Several Matchmaker arguments can dynamically provide information about how the messages implementing the call are sent and received. The syntax for each of them is of the form:

SpecialUsage ArgName : ArgType

The **ArgName** is an identifier for a parameter, and **ArgType** is the **type name** for that parameter. Specific **ArgType** values are required for each value of **SpecialUsage**.

SpecialUsage keywords:

RemotePort

Specify port to which request is sent. Required for all message classes except for **Alternate_Reply**. **ArgType** must be a port.

LocalPort

Specify port to which reply is sent. **ArgType** must be a port.

MsgType

Specify message type value for request message. Normal values are **NormalMsg** and **EmergMsg** (from **Accent.mm**). **ArgType** must be a 32-bit integer. The default is **Normal**.

ReplyType

Specify message type value for reply message (for **Remote_Procedure** declarations). **ReplyType** is implicitly an Out parameter on the server side of an interface. Normal values are **NormalMsg** and **EmergMsg** (from **Accent.mm**). **ArgType** must be a 32-bit integer. The default is **Normal**.

Send_Option

Specify the send option field for the request message. Normal values are **Wait**, **DontWait**, and **Reply** (from **Accent.mm**). **ArgType** must be a 16-bit integer. The default is **Wait**.

Send_Timeout

Specify the send timeout value for the request message. Values

are in milliseconds, and a zero value means to wait indefinitely. ArgType must be a signed 32-bit integer. The default is Wait_Forever (0).

Receive_Timeout

May be used to specify the receive timeout value for the reply message. Values are in milliseconds. A zero value means to wait indefinitely; -1 means return immediately if no message waiting. ArgType must be a signed 32-bit integer. The default is Wait_Forever (0).

A special abbreviation specifies the RemotePort argument for a Matchmaker call. If both SpecialUsage and ArgName are omitted (" : ArgType"), the call is interpreted the same as:

RemotePort ServPort : ArgType

5.3. Defaults and Options

Rather than passing special arguments to calls to specify every possible detail for every call, Matchmaker provides a set of defaults that cover most of the normal cases. These can generally be specified in one of two ways. Either the global default can be modified, or the default can be changed for a given message declaration.

5.3.1. Modifying global call defaults

The defaults for all messages in an interface can be changed using Options declarations. Options declarations must come before any data structure (Constant, Type, or Use) declarations.

An Options declaration consists of the Options keyword, followed by one or more declarations of the form:

OptionKey = OptionValue ;

Options declarations only affect the interface in which they occur. Another specification that references the interface containing the

Options declarations (for example, by a Use declaration) is not affected by the declarations.

5.3.2. Modifying local call defaults

The defaults for any particular call can be changed by inserting one or more clauses of the form:

, OptionKey = OptionValue

before the semicolon (;) terminating the call declaration. Such a clause is only effective for that specific call.

5.3.3. Listing of defaults

Many of the call defaults which can be changed are also call special arguments. Table 1 lists these keys, their defaults and legal values.

5.3.4. Non-call argument options

In addition to the call options listed in the previous section, three other global options are provided:

Protocol_Version

Declare version number of code generator algorithm to be used.
Currently only version 1 is supported.

Local_Ports Declare maximum number of ports that are allocated on the user side for outstanding calls to the server. A value of 0 requires all calls to specify a LocalPort special parameter. A value of * specifies no limit. Default is 1.

Ports_Backlog

Backlog value for any Local_Ports allocated. Default is 0 (requesting Accent default Backlog value).

Table 1: Special Arguments, Defaults and Legal Values

<u>Key</u>	<u>Default</u>	<u>Legal Values</u>
MsgType	NormalMsg (= 0)	NormalMsg, EmergencyMsg (0, 1)
ReplyType	NormalMsg (= 0)	NormalMsg, EmergencyMsg (0, 1)
Send_Option	Wait (= 0)	Wait,DontWait, Reply (0..2)
Send_Timeout	0 (Infinity)	32-bit signed integers (in milliseconds)
Receive_Timeout	0 (Infinity)	32-bit signed integers (in milliseconds)

5.3.5. Examples

```
Options
  Protocol_Version = 1;
  Local_Ports = 5;
  Receive_Timeout = 100;      ! Be impatient with it.
```

5.4. Message ID Declarations

Calls are normally assigned message ID numbers sequentially, in ascending order from the declared Interface ID number. Reply IDs for Remote_Procedure, Alternate_Reply, and Server_Message calls have 100 added to them, to distinguish them from request IDs.

There are instances, however, where it might be necessary to assign the IDs in another fashion. Matchmaker provides two declarations strictly for manipulating the message ID numbers that are assigned to declared calls.

5.4.1. Skip_ID declaration

The Skip_ID declaration increments the message ID counter exactly as a call declaration would. The primary purpose of the Skip_ID declaration is to replace an obsolete call declaration in an interface file.

The syntax for Skip_ID is:

```
Skip_ID ;
```

5.4.2. Next_ID declaration

The Next_ID declaration is used to directly set the message ID counter to a new value. The new value should be within 100 of the initial value.

The syntax for Next_ID is:

Next_ID = NewValue ;

where NewValue is an integer expression.

PERQ Systems Corporation
Accent Operating System

Matchmaker
Using Matchmaker

6. Using Matchmaker

This chapter describes how to use Matchmaker and code generated by Matchmaker.

6.1. Using a Generated Interface

Matchmaker only generates code that allows parameters for kinds of calls to be passed between User and Server processes. The code implementing the calls and any code that uses the calls must be hand-coded.

This section describes the specifics of writing code that is called from Matchmaker procedures and code that calls Matchmaker procedures. This section assumes that an interface called Prog has been declared.

For a detailed example of the files generated by Matchmaker, please refer to Chapter 9.

6.1.1. Using Matchmaker user code

The following applies to code written for the user side of a Matchmaker interface:

- The procedure InitProg must be called before any other procedure from module ProgUser (user side code) to initialize ProgUser.
- Any Message or Remote_Procedure can be called directly using appropriate target language calling conventions.
- Code must be written by the user for any Alternate_Reply calls declared to handle the

Alternate_Reply and take appropriate actions when they occur.

- The ProgDispatcher function may be called to decode parameters and to call the function implementing any Server_Message procedures declared in Prog. ProgDispatcher is passed a message. It returns true for messages it recognizes, and false for those it does not.
- Code must be written by the user for each declared Server_Message call that takes appropriate action for the Server_Message. These routines are called by the ProgDispatcher function for appropriate messages.
- An error-handling routine should be provided to catch any signaled errors.

6.1.2. Using Matchmaker server code

The following applies to code written for the server side of a Matchmaker interface:

- The implementor must write code to receive the message and then call the ProgServer function to decode message parameters and to call the function implementing any Message or Remote_Procedure procedures declared in Prog for received messages. ProgServer must be passed both the message received and a reply message. It returns a true for messages it recognizes and false for those it does not. The reply message is sent unless the ReturnValue field of the call is NOREPLY.
- The implementor must write code for each declared Message and the declared Remote_Procedure call that implements the call. These routines are called by the

ProgServer function for appropriate messages.

- An Alternate_Reply message can reply to a Remote_Procedure request by signaling the Alternate_Reply flag with appropriate parameters. Construct an alternate reply message for ProgServer to return.
- Any Server_Message may be directly called via appropriate target language calling conventions to pack and send the message.

6.2. Matchmaker File Names

The source file name for a Matchmaker specification named Prog should be:

Prog.mm

If a set of language-specific declarations for the specification are needed for a given language that normally uses the extension .ext, then those declarations should be in the file:

Prog.mm-ext

When compiled for a language using extension .ext, Matchmaker will then normally produce the files:

ProgDefs.ext (Data structures)
ProgUser.ext (User side code)
ProgServer.ext (Server side code)

For example, the Time interface uses the following files:

Time.mm Matchmaker source
TimeDefs.pas
Pascal types for time

TimeUUser.pas

Pascal user interface to time server

TimeServer.pas

Pascal server interface for time server (the Time server is
implemented in Pascal)

Time.h C types for time

TimeUser.c

C user interface to time server

TimeDefs.slisp

Accent Lisp types for time

TimeMsgDefs

Accent Lisp types for time messages

TimeUser.slisp

Accent Lisp user interface to time server

TimeUInit

Load-time initialization forms for the time interface

If language-specific declarations are needed, they are specified in
the files:

- Time.mm-pas (Pascal-specific declarations)
- Time.mm-c (C-specific declarations)
- Time.mm-slisp (Accent Lisp-specific declarations)

Information specific to each language is discussed in Chapter 8.

6.3. Matchmaker Command Line Syntax

Invoke Matchmaker with the command line:

Matchmaker FileName -Lang1=Opt1 ... -LangN=OptN

where FileName is the name of the specification file to be
compiled without the .mm extension. The Lang names are
currently members of the set Pascal, C, and Accent Lisp. The
Opt values must be members of the set:

All	Generate all files for Lang
Defs	Generate Defs file for Lang
User	Generate User file for Lang
Server	Generate Server file for Lang
NoUser	Generate all but User file for Lang
NoServer	Generate all but Server file for Lang

The order of switches and the filename does not matter. Unique abbreviations for both switches and options are accepted.

Other switches are:

Help	Display information about Matchmaker
Verbose	Display progress information
Quiet	Do not display progress information
Errorfile	Generate a file of error messages
NoErrorfile	Do not generate a file of error messages
Loop	Allow additional specification files to be processed before exiting

PERQ Systems Corporation
Accent Operating System

Matchmaker
BNF

7. A BNF for the Matchmaker Language

The following is a BNF description of the Matchmaker language.
Conventions used are as follows:

- Literal tokens are enclosed in double quotes (" ") .
- Optional productions are enclosed in square brackets ([]). .
- Three periods (...) denote optional repetition.
- A vertical bar (|) denotes choices between productions.
- Braces ({ }) enclose a group of required productions.
- Parens (()) enclose comments.

7.1. Interface and Options Definitions

```
Interface      ::= Interface_Decl
                  [Options_Decl]...
                  [Data_Decl]...
                  [Msg_Decl]...
                  End_Interface |
Types_Decl     ::= Types_Decl
                  [Options_Decl]...
                  [Data_Decl]...
                  End_Types

Interface_Decl ::= "Interface" Interface_Name "="
                  Msg_ID_Base ":""

Types_Decl     ::= "Types" Interface_Name ":""

Interface_Name ::= Identifier

Msg_ID_Base   ::= Integer_Constant

Reply_Base    ::= Integer_Constant

End_Interface ::= "End" "Interface"
```

```

End_Types           ::= "End" "Type;""
Options_Decl        ::= "Options" {Option_Decl ";"}...
                           (not yet implemented)
Option_Decl         ::= Msg_Options | Msg_Name_Options |
                           Protocol_Options
Msg_Name_Options   ::= Msg_Name_Key "=" Identifier
                           (not yet implemented)
Msg_Name_Key        ::= "Server_Prefix" | "User_Prefix"
Protocol_Options    ::= "Protocol_Version" = Integer_Constant
                           (not yet implemented)

```

7.2. Datatype Definitions

```

Data_Decl          ::= Use_Decl | Type_Decl |
                           Constant_Decl
Use_Decl          ::= "Use" Single_Use...
Single_Use         ::= Interface_Name "From" File_Name ";""
File_Name          ::= String_Constant
Constant_Decl      ::= "Constant" Single_Constant...
Single_Constant    ::= Constant_Name "=" Constant_Expr ";""
Constant_Name      ::= Identifier
Type_Decl          ::= "Type" Single_Type...
Single_Type         ::= Type_Name "=" Type_Specification
                           [".." Type_Option]... ";""
Type_Name          ::= Identifier
Type_Specification ::= Type_Name | Builtin_Type |
                           Array_Type | Record_Type |
                           Pointer_Type | Enumeration_Type |
                           Union_Type
Builtin_Type       ::= "Boolean" | "Character" | "Real" |
                           Integer_Type | String_Type |
                           Port_Type
Integer_Type        ::= "Unsigned" ["|" Integer_Constant "|"] |
                           "Signed" ["|" Integer_Constant "|"] |
                           Subrange_Type | "Long" | "Short" |
                           "Byte"

```

```
Subrange_Type      ::= Integer_Constant ".." Integer_Constant
Port_Type          ::= "Port" [".." "Port_Attribute" "="
                      Integer_Constant] |
                      "Port_Send" | "Port_Receive"
String_Type        ::= "PERQ_String" ["{" Integer_Constant "}" ]
Array_Type         ::= [Packing] "Array" "[" Array_Size "]"
                      "Of" Type_Specification
Array_Size         ::= Integer_Constant | "*"
Packing            ::= "Packed" | "Unpacked"
Record_Type        ::= [Packing] "Record"
                      Record_Component...
                      "End" "Record"
Record_Component   ::= Field_Identifier ":" Type_Specification ";"
Field_Identifier   ::= Identifier
Pointer_Type       ::= "*" Type_Specification
Enumeration_Type  ::= "(" Enum_List ")"
Enum_List          ::= Implicit_Enum_List |
                      Explicit_Enum_List
Implicit_Enum_List ::= Enum_Name ["," Enum_Name] ...
Explicit_Enum_List ::= Enum_Element ["," Enum_Element] ...
Enum_Element        ::= Enum_Name [= Integer_Constant]
Enum_Name           ::= Identifier
Union_Type          ::= "Union: "<" Union_Selector_Type ">" "
                      "Of" Union_Component... "End" "Union"
Union_Selector_Type ::= Type_Specification
Union_Component    ::= Union_Tag ":" "("
                      [Record_Component] ...
                      ")" ";" "
Union_Tag           ::= Constant_Expr | "Otherwise"
Type_Option         ::= "TypeType" "=" Integer_Constant |
                      "Deallocate" [= Boolean_Constant] |
                      "NoDeallocate" |
                      "Element_Size" "=" Integer_Constant |
```

```
"Element_Count" "=" Integer_Constant |  
"In_Translation_Fun" "=" Identifier |  
"Out_Translation_Fun" "=" Identifier
```

7.3. Message Definitions

```
Msg_Decl          ::= Msg_Code_Decl | Msg_ID_Decl  
  
Msg_Code_Decl    ::= Msg_Body [," Msg_Options]... ";"  
  
Msg_Body         ::= To_Server | From_Server |  
                    Remote_Procedure | Alternate_Reply  
  
Msg_Options      ::= Msg_Param_Key "="  
                    Integer_Constant |  
                    Msg_CodeGen_Key "="  
                    Boolean_Constant (NYI) |  
                    Msg_Doc_Key "="  
                    String_Constant  
  
To_Server        ::= "Message" Msg_Name Arg_List  
                    ":" Msg_Result  
  
Remote_Procedure ::= "Remote_Procedure" Msg_Name Arg_List  
                    ":" Msg_Result  
  
Alternate_Reply  ::= "Alternate_Reply" Msg_Name [Arg_List]  
  
From_Server      ::= "Server_Message" Msg_Name Arg_List  
                    ":" Msg_Result  
  
Msg_Name         ::= Identifier  
  
Msg_Result       ::= Special_Result | Arg_Type  
  
Special_Result   ::= "GR_Value" | "No_Value"  
  
Arg_List         ::= "(" Msg_Arg [":" Msg_Arg]... ")"  
  
Msg_Arg          ::= Data_Arg | Special_Arg  
  
Data_Arg         ::= [Arg_Direction] Data_Arg_Spec  
  
Arg_Direction    ::= "In" | "Out" | "InOut"  
  
Data_Arg_Spec    ::= Simple_Arg_Spec |  
                    Variable_Arg_Spec |  
                    Union_Arg_Spec  
  
Simple_Arg_Spec  ::= Arg_Name ":" Arg_Type  
  
Variable_Arg_Spec ::= Arg_Name "[" Arg_Cnt_Name "]" ":"  
                     Arg_Type | "[" Arg_Cnt_Name "]"  
                     Arg_Name ":" Arg_Type
```

```

Union_Arg_Spec      ::= Arg_Name "<" Selector_Name ">"  

                      ";" Arg_Type | "<" Selector_Name  

                      ">" Arg_Name ":" Arg_Type

Special_Arg        ::= Special_Usage Arg_Name ":"  

                      Arg_Type | ":" Arg_Type

Special_Usage       ::= Port_Usage_Key | Msg_Param_Key

Port_Usage_Key      ::= "RemotePort" | "LocalPort"

Msg_Param_Key       ::= "MsgType" | "Send_Option" |  

                      "Send_Timeout" | "Receive_Timeout"

Msg_Doc_Key         ::= Documentation |  

                      User_Documentation |  

                      Server_Documentation

Msg_CodeGen_Key    ::= Asynchronous |  

                      Asynchronous_User |  

                      Asynchronous_Server

Arg_Cnt_Name        ::= Arg_Name

Selector_Name        ::= Arg_Name

Arg_Name             ::= Identifier

Arg_Type             ::= Type_Name [ "," Type_Option]...

Msg_ID_Decl         ::= "Skip_ID" ";" |  

                      "Next_ID" "=" Msg_ID_Source ";"
```

7.4. Expression Syntax

```

Constant_Expr        ::= OR_CTCE (Valid types context  

                                dependent)  

Integer_Constant     ::= Adding_CTCE (Must be integer valued)  

Boolean_Constant     ::= Or_CTCE (Must be boolean valued)  

Character_Constant   ::= Primary_CTCE (Must be character  

                                valued)  

String_Constant      ::= Primary_CTCE (Must be string valued)  

Enumeration_Constant ::= Primary_CTCE (Must result in a  

                                declared Enum_Name identifier)  

Or_CTCE               ::= And_CTCE [ "Or" And_CTCE]...  

And_CTCE              ::= Not_CTCE [ "And" Not_CTCE]...  

Not_CTCE              ::= [ "Not" ] Relational_CTCE
```

```
Relational_CTCCE ::= Equality_CTCCE
                     [ {">" | ">=" | "<" | "<=" }
                      Equality_CTCCE ] ...

Equality_CTCCE ::= Adding_CTCCE [ {"=" | "<>" }
                                Adding_CTCCE] ...

Adding_CTCCE ::= [ {"+" | "-"} Multiplying_CTCCE
                     [ {"+" | "-"} Multiplying_CTCCE] ...

Multiplying_CTCCE ::= Primary_CTCCE
                     [ {"**" | "/" | "Mod"} Primary_CTCCE] ...

Primary_CTCCE ::= Identifier | Constant_Lexeme |
                  "(" Or_CTCCE ")"
```

7.5. Lexical Definitions

```
Constant_Lexeme ::= Octal_Literal | Decimal_Literal |
                     String_Literal | Character_Literal |
                     Boolean_Literal

Octal_Literal ::= "#" followed by a non-empty octal
                  digit string

Decimal_Literal ::= A non-empty decimal digit string

String_Literal ::= A character string enclosed in double
                  quotes. A double quote may be
                  represented by two.

Character_Literal ::= A character enclosed in single quotes.
                     A single quote may be represented
                     by two.

Boolean_Literal ::= "True" | "False"

Identifier ::= A string composed of letters, digits
              the underscore character, not starting
              with a digit. Identifiers are matched
              in a non-case-sensitive manner.
```

At any lexical break, comments can be inserted as:

"!" comment text <End_Of_Line>

8. Information Specific to Each Code Generator

8.1. Pascal

To import files into a Matchmaker-generated Pascal user or server file, create a file named

FOO.mm-pas

where FOO is the name of the interface declared at the beginning of the Matchmaker declaration file. The contents of FOO.mm-pas should look like:

```
(in-package "MM-PASCAL-GEN")
(simports "file1")
(simports "file2")
(uimports "file3")
(uimports "file4")
(imports "file5")
(imports "file6")
```

In this example the files file1.pas, file2.pas, file5.pas, and file6.pas will be imported into FOOserver.pas, and files number 3, 4, 5, and 6 will be imported into FOouser.pas.

The Matchmaker-generated file FOOserver.pas will import the file FOOProcs.pas. This file must contain the actual human-written routines that FOOserver.pas calls.

8.2. C

Using Matchmaker with the C language works the same as with Pascal except:

1. (In-package "MM-PASCAL-GEN") becomes
(in-package "MM-C-GEN").

2. Filenames have the extension .c instead of .pas.
3. If the module name and filename are not the same, you should state both, as in:

```
(imports "ModuleName" "FileName")
```

Otherwise, Matchmaker assumes the module and file have the same name.

The Matchmaker-generated file FOOserver.c will import the file FOOProcs.h. This file must contain the actual human-written routines that FOOserver.c calls.

If there are any asynchronous Server routine declarations in FOO.mm, FOOUser.c will import the file FOOUProcs.h, which must contain the asynchronous routines that are called in FOOUser.c. This file is not used for Pascal because, in Pascal, asynchronous server routines are treated as exceptions, and it is impossible to import exception handlers in Pascal.

8.3. Lisp

Using Matchmaker with the Accent Lisp language works the same as with Pascal except:

1. The source file in the above example would be named FOO.mm-slisp.
2. The contents of the source file would be:

```
(in-package "MM-LISP-GEN")
<-arbitrary Lisp expressions to be evaluated->)
```

Matchmaker does not import any Procs files for Lisp. The person writing the interface must include the human-written routines in the FOOUser and FOOserver packages using Common Lisp package mechanisms.

For more information, see the *Accent Lisp Manual*.

8.4. Substitute-typedef

The Substitute-typedef command can be used to create data types using the files FOO.mm-pas, FOO.mm-c and FOO.mm-lisp that are too complicated to be generated directly by Matchmaker.

For example, creating a variant record data type is usually not possible in Matchmaker. Instead, the following code can be placed in the file FOO.mm:

```
type pRecord = ^array [*] of byte;  
  
Remote_Procedure DoSomethingToRecord(  
    RemotePort FooPort : Port;  
    InOut TheRecord [Length_of_array] : pRecord  
): No_Value;
```

The actual desired definition of pRecord is a pointer to a variant record. The actual definition should be placed in the file FOO.mm-pas in the following form:

```
(in-package "MM-PASCAL-GEN")  
(substitute-typedef "pRecord"  
    "record case boolean of  
        True : (a : integer);  
        False : (b : array [32] of integer)  
    end")
```

When Matchmaker generates FOODefs.pas, the definition of pRecord in FOO.mm-pas will be substituted for the definition in FOO.mm. When DoSomethingToRecord is created, it will treat TheRecord as if it were a pointer to a variable length array type, and will even send a value for Length_of_array to the server procedure. Both the complex variant record (which Matchmaker cannot parse) and a variable length array (which Matchmaker CAN parse) require the same generated code to handle them; therefore the different type definitions for pRecord will be transparent.

Substitute-typedef is used analogously for C and Lisp.

Substitute-typedef is intended to be used to aid in the conversion from the past Pascal-only version of Matchmaker to the new multiple-language version. It is not intended for general use and should be used only when the cost of using a more general language-independent mechanism is too high. Avoid using Substitute-typedef whenever possible.

9. Example Source and Matchmaker Files

In the example given in this section, Matchmaker generates the Typescript client interface for Pascal and Lisp and the Typescript server interface for Pascal.

9.1. Source Files

9.1.1. Typescript.mm

This file is a language-independent interface specification.

```
interface TS=2800;
!*****!
! Author:      Pedro Szekely
! Abstract:    Matchmaker interface for Typescript
!
!-----
! Change log:
! Aug 27 1984 jrg   Oops! Had to change TSCharArray arg
!                      STSPutCharArray to a pTSCharArray.
!                      of course.
! Jul 31 1984 jrg   Changed ptschararray arg to
!                      STSPutCharArray to a TSCharArray
!                      with a char_count arg for new MM
!                      format. Also changed def of
!                      TSCharArray to be variable-sized
!                      for backwards compatibility.
! May 25 1984 JmL   Converted to new matchmaker format.
!-----
```

```
use Accent from "accent";
use Sapphdefs from "sapphdefs";

type
  Typescript = Port;
  TSString255 = Perg_String[255];
  TSCharArray = packed array[*] of Character;
  PTSCharArray = ^ TSCharArray.

Remote_Procedure STSOpen(: Port; vp: ViewPort; env: Port);
  Typescript;

Remote_Procedure STSOpenWindow( : Port,
```

```
w: Window;
env: Port
): Typescript;

Remote_Procedure STSFullOpen( : Port;
    vp : ViewPort;
    env : Port;
    fontName : TString255;
    doWrap : Boolean;
    dispPages : Integer
): Typescript;

Remote_Procedure STSFullOpenWindow( : Port;
    w : Window;
    env : Port;
    fontName : TString255;
    doWrap : Boolean;
    dispPages : Integer
): Typescript;

Remote_Procedure STSFullLine(): Boolean;

Remote_Procedure STSGetChar(): Character;

Remote_Procedure STSGetString(): TString255;

Message STSPutChar(: Typescript; ch: Character): No_Value;

Message STSPutString(: Typescript; s: TString255): No_Value;

Message STSFlushInput(): Typescript): No_Value;

Remote_Procedure STSFlushOutput(): Typescript): No_Value,
    !use for synchronization

Message STSChangeEnv(: Typescript; env: Port): No_Value;

Remote_Procedure STSGrabWindow(): Typescript; kPort: Port);
    Window;

Message STSPutCharArray(: Typescript;
    chars [char_count];
    pTSCcharArray;
    firstCh: Integer;
    lastCh: Integer
): No_Value;

end interface
```

9.1.2. TS.mm-pas

This file contains Pascal-specific declarations.

```
(in-package "MM-PASCAL-CEN")
(simports "TSTypes")
(simports "STypescript")
```

9.2. Matchmaker-Produced Files

9.2.1. TSDefs.pas

This file contains Pascal types.

```
Module TSDefs;

Exports

Imports BuiltinDefs from BuiltinDefs;
Imports AccIntDefs from AccIntDefs;
Imports SdefsDefs from SdefsDefs;

type
  Typescript = port;
  TSString255 = string[255];
  TSCharArray = packed array [0 .. 0] of char;
  pTSCharArray = ^TSCharArray;

Private

Procedure Bug;
begin end.
```

9.2.2. TSUser.pas

This file is the Pascal user interface.

```
Module TS;

{ -----> } Exports { <----- }

Imports AccInt from AccIntUser;
Imports TSDefs from TSDefs;

Procedure InitTS(Rport : port);

Function STSOpen(
  ServPort : Port;
  vp : Viewport;
  env : port): Typescript;

Function STSOpenWindow(
  ServPort : Port;
  w : Window;
```

```
    env : port): Typescript;

Function STSFullOpen(
    ServPort : Port;
    vp : Viewport;
    env : port;
    fontName : TSString255;
    doWrap : boolean;
    dispPages : Integer): Typescript;

Function STSFullOpenWindow(
    ServPort : Port;
    w : Window;
    env : port;
    fontName : TSString255;
    doWrap : boolean;
    dispPages : Integer): Typescript;

Function STSFullLine(
    ServPort : Port): boolean;

Function STSGetChar(
    ServPort : Port): char;

Function STSGetString(
    ServPort : Port): TSString255;

Procedure STSPutChar(
    ServPort : Port;
    ch : char);

Procedure STSPutString(
    ServPort : Port;
    s : TSString255);

Procedure STSFlushInput(
    ServPort : Port);

Procedure STSFlushOutput(
    ServPort : Port);

Procedure STSChangeEnv(
    ServPort : Port;
    env : port);

Function STSGrabWindow(
    ServPort : Port;
    kPort : port): Window;

Procedure STSPutCharArray(
    ServPort : Port;
    chars : pTSCharArray;
    char_count : long;
    firstCh : Integer;
    lastCh : Integer).
```

```
Function TSAsynch(InP: Pointer): boolean;
{ ----- } Private { ----- }

Imports AccCall from AccCall;
Imports PascalInit from PascalInit;
type
  DumMsg = record
    head : Msg;
    body: array [0..1023] of integer
  end;
  ptrDumMsg = ^DumMsg;

var
  ReplyPort : Port;
  GR        : GeneralReturn;
  ReP : ptrDumMsg;

Procedure TSEExceptions;
  forward;

Procedure InitTS(Rport : port);
begin
  if RPort = NullPort then
  begin
    GR:=AllocatePort(KernelPort,ReplyPort,5);
    if GR<>Success then ReplyPort:=DataPort
  end
  else ReplyPort:=RPort;
  new(ReP)
end;

Function STSOpen(
  ServPort : Port;
  vp : Viewport;
  env : port): Typescript;

type
  MyMessage = record
    head : Msg;
    IPCNam2 : TypeType;
    Arg2 : Viewport;
    IPCNam3 : TypeType;
    Arg3 : port
  end;

type
  RepMessage = record
    head : Msg;
    RetCodeType : TypeType;
    RetCode : GeneralReturn;
    IPCNam4 : TypeType;
    Arg4 : Typescript
  end;
  ptrRepMsg = ^RepMessage;
```

```
var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg);
  with MyMsg.head do
    begin
      Simplemsg := FALSE;
      MsgSize:= 38;
      MsgType:= NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2800
    end;
  with MyMsg do
    begin
      IPCNam2.InLine := TRUE;
      IPCNam2.Deallocate := FALSE;
      IPCNam2.LongForm := FALSE;
      IPCNam2.TypeName := 6;
      IPCNam2.TypeSizeInBits := 32;
      IPCNam2.NumObjects := 1;
      Arg2 := vp;
      IPCNam3.InLine := TRUE;
      IPCNam3.Deallocate := FALSE;
      IPCNam3.LongForm := FALSE;
      IPCNam3.TypeName := 6;
      IPCNam3.TypeSizeInBits := 32;
      IPCNam3.NumObjects := 1;
      Arg3 := env;
    end;
  with RepMsgP^.head do
    begin
      MsgSize := WordSize(DumMsg)*2;
      LocalPort := ReplyPort
    end;
  GR := Send(MyMsg.head,0,WAIT);
  if GR <> Success then
    begin
      Raise GRError(GR);
      exit(STSOpen)
    end;
  GR := Receive(RepMsgP^.head,0,LOCALPT,RECEIVEIT);
  If GR <> Success then
    begin
      Raise GRError(GR);
      exit(STSOpen)
    end;
  with RepMsgP^ do
    begin
      If head.ID <> 2900 then
        begin
          TSEExceptions;
          exit(STSOpen)
        end;
    end;
```

```
if RetCodeType.TypeName => TYPEINT16 then
begin
  Raise GRError(BADREPLY);
  exit(STSOpen)
end;
if (RetCode <> Success) then
begin
  Raise GRError(BadReply);
  Exit(STSOpen)
end;
if IPCNam4.TypeName <> 6 then
begin
  Raise GRError(BadReply);
  exit(STSOpen)
end;
STSOpen := Arg4;
end;
end;

Function STSOpenWindow(
  ServPort : Port;
  w : Window;
  env : port): Typescript;

type
  MyMessage = record
    head : Msg;
    IPCNam2 : TypeType;
    Arg2 : Window;
    IPCNam3 : TypeType;
    Arg3 : port
  end;

type
  RepMessage = record
    head : Msg;
    RetCodeType : TypeType;
    RetCode : GeneralReturn;
    IPCNam4 : TypeType;
    Arg4 : Typescript
  end;
  ptrRepMsg = ^RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg);
  with MyMsg.head do
  begin
    Simplemsg := FALSE;
    MsgSize:= 38;
    MsgType:= NormalMsg;
```

```
RemotePort := ServPort;
LocalPort := ReplyPort;
ID := 2801
end;
with MyMsg do
begin
  IPCNam2.InLine := TRUE;
  IPCNam2.Deallocate := FALSE;
  IPCNam2.LongForm := FALSE;
  IPCNam2.TypeName := 6;
  IPCNam2.TypeSizeInBits := 32;
  IPCNam2.NumObjects := 1;
  Arg2 := w;
  IPCNam3.InLine := TRUE;
  IPCNam3.Deallocate := FALSE;
  IPCNam3.LongForm := FALSE;
  IPCNam3.TypeName := 6;
  IPCNam3.TypeSizeInBits := 32;
  IPCNam3.NumObjects := 1;
  Arg3 := env;
end;
with RepMsgP^.head do
begin
  MsgSize := WordSize(DumMsg)*2;
  LocalPort := ReplyPort
end;
GR := Send(MyMsg.head,0,WAIT);
if GR <> Success then
begin
  Raise GRError(GR);
  exit(STSOpenWindow)
end;
GR := Receive(RepMsgP^.head,0,LOCALPT,RECEIVEIT);
If GR <> Success then
begin
  Raise GRError(GR);
  exit(STSOpenWindow)
end;
with RepMsgP^ do
begin
  If head.ID <> 2901 then
  begin
    TSExceptions;
    exit(STSOpenWindow)
  end;
  if RetCodeType.TypeName <> TYPEINT16 then
  begin
    Raise GRError(BADREPLY);
    exit(STSOpenWindow)
  end;
  if (RetCode <> Success) then
  begin
    Raise GRError(BadReply);
    Exit(STSOpenWindow)
  end;
  if IPCNam4.TypeName <> 6 then
```

```
begin
  Raise GSError(BadReply);
  exit(STSOpenWindow)
end;
STSOpenWindow := Arg4;
end;
end;

Function STSFullOpen(
  ServPort : Port;
  vp : Viewport;
  env : port;
  fontName : TSString255;
  doWrap : boolean;
  dispPages : Integer): Typescript;

type
  MyMessage = record
    head      : Msg;
    IPCNam2  : TypeType;
    Arg2     : Viewport;
    IPCNam3  : TypeType;
    Arg3     : port;
    IPCNam4  : TypeType;
    Arg4     : TSString255;
    IPCNam5  : TypeType;
    Arg5     : boolean;
    IPCNam6  : TypeType;
    Arg6     : Integer
  end;

  type
    RepMessage = record
      head      : Msg;
      RetCodeType : TypeType;
      RetCode   : GeneralReturn;
      IPCNam7  : TypeType;
      Arg7     : Typescript
    end;
    ptrRepMsg  = ^RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg);
  with MyMsg.head do
  begin
    Simplemsg := FALSE;
    MsgSize:= 310;
    MsgType:= NormalMsg;
    RemotePort := ServPort;
    LocalPort := ReplyPort;
```

```
ID := 2802
end;
with MyMsg do
begin
  IPCNam2.InLine := TRUE;
  IPCNam2.Deallocate := FALSE;
  IPCNam2.LongForm := FALSE;
  IPCNam2.TypeName := 6;
  IPCNam2.TypeSizeInBits := 32;
  IPCNam2.NumObjects := 1;
  Arg2 := vp;
  IPCNam3.InLine := TRUE;
  IPCNam3.Deallocate := FALSE;
  IPCNam3.LongForm := FALSE;
  IPCNam3.TypeName := 6;
  IPCNam3.TypeSizeInBits := 32;
  IPCNam3.NumObjects := 1;
  Arg3 := env;
  IPCNam4.InLine := TRUE;
  IPCNam4.Deallocate := FALSE;
  IPCNam4.LongForm := FALSE;
  IPCNam4.TypeName := 0;
  IPCNam4.TypeSizeInBits := 8;
  IPCNam4.NumObjects := 256;
  Arg4 := fontName;
  IPCNam5.InLine := TRUE;
  IPCNam5.Deallocate := FALSE;
  IPCNam5.LongForm := FALSE;
  IPCNam5.TypeName := 0;
  IPCNam5.TypeSizeInBits := 1;
  IPCNam5.NumObjects := 1;
  Arg5 := doWrap;
  IPCNam6.InLine := TRUE;
  IPCNam6.Deallocate := FALSE;
  IPCNam6.LongForm := FALSE;
  IPCNam6.TypeName := 1;
  IPCNam6.TypeSizeInBits := 16;
  IPCNam6.NumObjects := 1;
  Arg6 := dispPages;
end;
with RepMsgP^.head do
begin
  MsgSize := WordSize(DumMsg)*2;
  LocalPort := ReplyPort
end;
GR := Send(MyMsg^.head,0,WAIT);
if GR <> Success then
begin
  Raise GRError(GR);
  exit(STSFull0pen)
end;
GR := Receive(RepMsgP^.head,0,LOCALPT,RECEIVEIT);
If GR <> Success then
begin
  Raise GRError(GR);
  exit(STSFull0pen)
```

```
    end;
  with RepMsgP^ do
  begin
    If head.ID <> 2902 then
      begin
        TSExceptions;
        exit(STSFullOpen)
      end;
    if RetCodeType.TypeName <> TYPEINT16 then
      begin
        Raise GRError(BADREPLY);
        exit(STSFullOpen)
      end;
    if (RetCode <> Success) then
      begin
        Raise GRError(BadReply);
        Exit(STSFullOpen)
      end;
    if IPCNam7.TypeName <> 6 then
      begin
        Raise GRError(BadReply);
        exit(STSFullOpen)
      end;
    STSFullOpen := Arg7;
  end;
end;

Function STSFullOpenWindow(
  ServPort : Port;
  w : Window;
  env : port;
  fontName : TSString255;
  doWrap : boolean;
  dispPages : Integer): Typescript;

type
  MyMessage = record
    head : Msg;
    IPCNam2 : TypeType;
    Arg2 : Window;
    IPCNam3 : TypeType;
    Arg3 : port;
    IPCNam4 : TypeType;
    Arg4 : TSString255;
    IPCNam5 : TypeType;
    Arg5 : boolean;
    IPCNam6 : TypeType;
    Arg6 : Integer
  end;

type
  RepMessage = record
    head : Msg;
    RetCodeType : TypeType;
    RetCode : GeneralReturn;
```

```
IPCNam7 : TypeType;
Arg7    : Typescript
end;
ptrRepMsg = `RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg);
  with MyMsg.head do
    begin
      Simplemsg := FALSE;
      MsgSize:= 310;
      MsgType:= NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2803
    end;
  with MyMsg do
    begin
      IPCNam2.InLine := TRUE;
      IPCNam2.Deallocate := FALSE;
      IPCNam2.LongForm := FALSE;
      IPCNam2.TypeName := 6;
      IPCNam2.TypeSizeInBits := 32;
      IPCNam2.NumObjects := 1;
      Arg2 := w;
      IPCNam3.InLine := TRUE;
      IPCNam3.Deallocate := FALSE;
      IPCNam3.LongForm := FALSE;
      IPCNam3.TypeName := 6;
      IPCNam3.TypeSizeInBits := 32;
      IPCNam3.NumObjects := 1;
      Arg3 := env;
      IPCNam4.InLine := TRUE;
      IPCNam4.Deallocate := FALSE;
      IPCNam4.LongForm := FALSE;
      IPCNam4.TypeName := 0;
      IPCNam4.TypeSizeInBits := 8;
      IPCNam4.NumObjects := 256;
      Arg4 := fontName;
      IPCNam5.InLine := TRUE;
      IPCNam5.Deallocate := FALSE;
      IPCNam5.LongForm := FALSE;
      IPCNam5.TypeName := 0;
      IPCNam5.TypeSizeInBits := 1;
      IPCNam5.NumObjects := 1;
      Arg5 := doWrap;
      IPCNam6.InLine := TRUE;
      IPCNam6.Deallocate := FALSE;
      IPCNam6.LongForm := FALSE;
      IPCNam6.TypeName := 1;
      IPCNam6.TypeSizeInBits := 16;
      IPCNam6.NumObjects := 1;
```

```
    Arg6 := dispPages;
end;
with RepMsgP^.head do
begin
  MsgSize := WordSize(DumMsg)*2;
  LocalPort := ReplyPort
end;
GR := Send(MyMsg^.head,0,WAIT);
if GR <> Success then
begin
  Raise GRError(GR);
  exit(STSFullOpenWindow)
end;
GR := Receive(RepMsgP^.head,0,LOCALPT,RECEIVEIT);
If GR <> Success then
begin
  Raise GRError(GR);
  exit(STSFullOpenWindow)
end;
with RepMsgP^ do
begin
  If head.ID <> 2903 then
  begin
    TSExceptions;
    exit(STSFullOpenWindow)
  end;
  if RetCodeType.TypeName <> TYPEINT16 then
  begin
    Raise GRError(BADREPLY);
    exit(STSFullOpenWindow)
  end;
  if (RetCode <> Success) then
  begin
    Raise GRError(BadReply);
    Exit(STSFullOpenWindow)
  end;
  if IPCNam7.TypeName <> 6 then
  begin
    Raise GRError(BadReply);
    exit(STSFullOpenWindow)
  end;
  STSFullOpenWindow := Arg7;
end;
end;

Function STSFullLine(
  ServPort : Port): boolean;

type
  MyMessage = record
    head      : Msg
  end;

type
  RepMessage = record
```

```
head      : Msg;
RetCodeType : TypeType;
RetCode    : GeneralReturn;
IPCNam2  : TypeType;
Arg2      : boolean
end;
ptrRepMsg = `RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg).
  with MyMsg.head do
    begin
      Simplemsg := TRUE.
      MsgSize:= 22;
      MsgType:= NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2804
    end;
  with MyMsg do
    begin
    end;
  with RepMsgP^.head do
    begin
      MsgSize := WordSize(DumMsg)*2;
      LocalPort := ReplyPort
    end;
  GR := Send(MyMsg.head,0,WAIT);
  if GR <> Success then
    begin
      Raise GSError(GR);
      exit(STSFullLine)
    end;
  GR := Receive(RepMsgP^.head,0,LOCALPT,RECEIVEIT);
  If GR <> Success then
    begin
      Raise GSError(GR);
      exit(STSFullLine)
    end;
  with RepMsgP do
    begin
      If head.ID <> 2904 then
        begin
          TSExceptions;
          exit(STSFullLine)
        end;
      if RetCodeType.TypeName <> TYPEINT16 then
        begin
          Raise GSError(BADREPLY);
          exit(STSFullLine)
        end;
      if (RetCode <> Success) then
```

```
begin
  Raise GRError(BadReply);
  Exit(STSFullLine)
end;
if IPCNam2.TypeName <> 0 then
begin
  Raise GRError(BadReply);
  exit(STSFullLine)
end;
STSFullLine := Arg2;
end;
end;

Function STSGetChar(
  ServPort : Port): char;

type
  MyMessage = record
    head      : Msg
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode     : GeneralReturn;
    IPCNam2   : TypeType;
    Arg2       : char
  end;
  ptrRepMsg  = ^RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg);
  with MyMsg.head do
  begin
    Simplemsg := TRUE;
    MsgSize:= 22;
    MsgType:= NormalMsg;
    RemotePort := ServPort;
    LocalPort := ReplyPort;
    ID := 2805
  end;
  with MyMsg do
  begin
  end;
  with RepMsgP^.head do
  begin
    MsgSize := WordSize(DumMsg)*2;
    LocalPort := ReplyPort
  end;
  GR := Send(MyMsg.head,0,WAIT);
```

```
if GR <> Success then
begin
  Raise GRError(GR);
  exit(STSGetChar)
end;
GR := Receive(RepMsgP^.head,0,LOCALPT,RECEIVEIT);
If GR <> Success then
begin
  Raise GRError(GR);
  exit(STSGetChar)
end;
with RepMsgP^ do
begin
  If head.ID <> 2905 then
  begin
    TSExceptions;
    exit(STSGetChar)
  end;
  if RetCodeType.TypeName <> TYPEINT16 then
  begin
    Raise GRError(BADREPLY);
    exit(STSGetChar)
  end;
  if (RetCode <> Success) then
  begin
    Raise GRError(BadReply);
    Exit(STSGetChar)
  end;
  if IPCNam2.TypeName <> 8 then
  begin
    Raise GRError(BadReply);
    exit(STSGetChar)
  end;
  STSGetChar := Arg2;
end;
end;

Function STSGetString(
  ServPort : Port): TSString255;

type
  MyMessage = record
    head      : Msg
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode   : GeneralReturn;
    IPCNam2  : TypeType;
    Arg2     : TSString255
  end;
  ptrRepMsg = ^RepMessage;
```

```
var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg);
  with MyMsg.head do
    begin
      Simplemsg := TRUE;
      MsgSize := 22;
      MsgType := NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2806
    end;
  with MyMsg do
    begin
    end;
  with RepMsgP^.head do
    begin
      MsgSize := WordSize(DumMsg)*2;
      LocalPort := ReplyPort
    end;
  GR := Send(MyMsg.head,0,WAIT);
  if GR <> Success then
    begin
      Raise GRError(GR);
      exit(STSGetString)
    end;
  GR := Receive(RepMsgP^.head,0,LOCALPT,RECEIVEIT);
  If GR <> Success then
    begin
      Raise GRError(GR);
      exit(STSGetString)
    end;
  with RepMsgP^ do
    begin
      If head.ID <> 2906 then
        begin
          TSExceptions;
          exit(STSGetString)
        end;
      if RetCodeType.TypeName <> TYPEINT16 then
        begin
          Raise GRError(BADREPLY);
          exit(STSGetString)
        end;
      if (RetCode <> Success) then
        begin
          Raise GRError(BadReply);
          Exit(STSGetString)
        end;
      if IPCNam2.TypeName <> 0 then
        begin
          Raise GRError(BadReply);
          exit(STSGetString)
        end;
    end;
end.
```

```
    end;
    STSGetString := Arg2;
  end;
end;

Procedure STSPutChar(
  ServPort : Port;
  ch : char);

type
  MyMessage = record
    head : Msg;
    IPCNam2 : TypeType;
    Arg2 : char;
  end;

type
  RepMessage = record
    head : Msg;
    RetCodeType : TypeType;
    RetCode : GeneralReturn;
  end;
  ptrRepMsg = ^RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg);
  with MyMsg.head do
    begin
      Simplemsg := TRUE;
      MsgSize := 28;
      MsgType := NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2807;
    end;
  with MyMsg do
    begin
      IPCNam2.InLine := TRUE;
      IPCNam2.Deallocate := FALSE;
      IPCNam2.LongForm := FALSE;
      IPCNam2.TypeName := 8;
      IPCNam2.TypeSizeInBits := 8;
      IPCNam2.NumObjects := 1;
      Arg2 := ch;
    end;
  GR := Send(MyMsg.head,0,WAIT);
  if GR <> Success then
    begin
      Raise GRError(GR);
      exit(STSPutChar)
    end;
end;
```

```
end;

Procedure STSPutString(
  ServPort : Port;
  s : TSString255);

type
  MyMessage = record
    head      : Msg;
    IPCNam2  : TypeType;
    Arg2     : TSString255
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode   : GeneralReturn
  end;
  ptrRepMsg  = ^RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP,ptrRepMsg);
  with MyMsg.head do
    begin
      Simplemsg := TRUE;
      MsgSize:= 282;
      MsgType:= NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2808
    end;
  with MyMsg do
    begin
      IPCNam2.InLine := TRUE;
      IPCNam2.Deallocate := FALSE;
      IPCNam2.LongForm := FALSE;
      IPCNam2.TypeName := 0;
      IPCNam2.SizeTypeInBits := 8;
      IPCNam2.NumObjects := 256;
      Arg2 := s;
    end;
  GR := Send(MyMsg.head,0,WAIT);
  if GR <> Success then
    begin
      Raise GRError(GR);
      exit(STSPutString)
    end;
end;
```

```
Procedure STSFlushInput(
    ServPort : Port);

type
  MyMessage = record
    head      : Msg
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode    : GeneralReturn
  end;
  ptrRepMsg = ^RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg);
  with MyMsg.head do
    begin
      Simplemsg := TRUE;
      MsgSize:= 22;
      MsgType:= NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2809
    end;
  with MyMsg do
    begin
    end;
  GR := Send(MyMsg.head,0,WAIT);
  if GR <> Success then
    begin
      Raise GSError(GR);
      exit(STSFlushInput)
    end;
end;

Procedure STSFlushOutput(
    ServPort : Port);

type
  MyMessage = record
    head      : Msg
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode    : GeneralReturn
```

```
end;
ptrRepMsg = `RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg);
  with MyMsg.head do
    begin
      Simplemsg := TRUE;
      MsgSize := 22;
      MsgType := NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2810
    end;
  with MyMsg do
    begin
      end;
  with RepMsgP^.head do
    begin
      MsgSize := WordSize(DumMsg)*2;
      LocalPort := ReplyPort
    end;
  GR := Send(MyMsg.head,0,WAIT);
  if GR <> Success then
    begin
      Raise GSError(GR);
      exit(STSFlushOutput)
    end;
  GR := Receive(RepMsgP^.head,0,LOCALPT,RECEIVEIT);
  If GR <> Success then
    begin
      Raise GSError(GR);
      exit(STSFlushOutput)
    end;
  with RepMsgP do
    begin
      If head.ID <> 2910 then
        begin
          TSExceptions;
          exit(STSFlushOutput)
        end;
      if RetCodeType.TypeName <> TYPEINT16 then
        begin
          Raise GSError(BADREPLY);
          exit(STSFlushOutput)
        end;
      if (RetCode <> Success) then
        begin
          Raise GSError(BadReply);
          Exit(STSFlushOutput)
        end;
    end;
end;
```

```
end.

Procedure STSChangeEnv(
  ServPort : Port;
  env : port);

type
  MyMessage = record
    head      : Msg;
    IPCNam2  : TypeType;
    Arg2     : port
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode   : GeneralReturn
  end;
  ptrRepMsg  = ^RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP.ptrRepMsg);
  with MyMsg.head do
    begin
      Simplemsg := FALSE;
      MsgSize:= 30;
      MsgType:= NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2811
    end;
  with MyMsg do
    begin
      IPCNam2.InLine := TRUE;
      IPCNam2.Deallocate := FALSE;
      IPCNam2.LongForm := FALSE;
      IPCNam2.TypeName := 6;
      IPCNam2.TypeSizeInBits := 32;
      IPCNam2.NumObjects := 1;
      Arg2 := env;
    end;
  GR := Send(MyMsg.head,0,WAIT);
  if GR <> Success then
    begin
      Raise GRError(GR);
      exit(STSChangeEnv)
    end;
end;
```

```
Function STSGrabWindow(
    ServPort : Port;
    kPort : port): Window;

type
  MyMessage = record
    head : Msg;
    IPCNam2 : TypeType;
    Arg2 : port
  end;

type
  RepMessage = record
    head : Msg;
    RetCodeType : TypeType;
    RetCode : GeneralReturn;
    IPCNam3 : TypeType;
    Arg3 : Window
  end;
  ptrRepMsg = ^RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP,ptrRepMsg);
  with MyMsg.head do
    begin
      Simplemsg := FALSE;
      MsgSize := 30;
      MsgType := NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2812
    end;
  with MyMsg do
    begin
      IPCNam2.InLine := TRUE;
      IPCNam2.Deallocate := FALSE;
      IPCNam2.LongForm := FALSE;
      IPCNam2.TypeName := 6;
      IPCNam2.TypeSizeInBits := 32;
      IPCNam2.NumObjects := 1;
      Arg2 := kPort;
    end;
  with RepMsgP^.head do
    begin
      MsgSize := WordSize(DumMsg)*2;
      LocalPort := ReplyPort
    end;
  GR := Send(MyMsg.head,0,WAIT);
  if GR <> Success then
    begin
      Raise GRError(GR);
      exit(STSGrabWindow)
```

```
    end;
    GR := Receive(RepMsgP^.head, 0, LOCALPT, RECEIVEIT);
    If GR <> Success then
        begin
            Raise GRError(GR);
            exit(STSGrabWindow)
        end;
    with RepMsgP^ do
        begin
            If head.ID <> 2912 then
                begin
                    TSExceptions;
                    exit(STSGrabWindow)
                end;
            if RetCodeType.TypeName <> TYPEINT16 then
                begin
                    Raise GRError(BADREPLY);
                    exit(STSGrabWindow)
                end;
            if (RetCode <> Success) then
                begin
                    Raise GRError(BadReply);
                    Exit(STSGrabWindow)
                end;
            if IPCNam3.TypeName <> 6 then
                begin
                    Raise GRError(BadReply);
                    exit(STSGrabWindow)
                end;
            STSGrabWindow := Arg3;
        end;
    end;

Procedure STSPutCharArray(
    ServPort : Port;
    chars : pTCharArray;
    char_count : long;
    firstCh : Integer;
    lastCh : Integer);

type
  MyMessage = record
    head      : Msg;
    IPCNam2  : TypeType;
    TName2   : integer;
    TSize2   : integer;
    NumElts2: long;
    Arg2     : pTCharArray;
    IPCNam3  : TypeType;
    Arg3     : Integer;
    IPCNam4  : TypeType;
    Arg4     : Integer
  end;

type
```

```
RepMessage = record
  head      : Msg;
  RetCodeType : TypeType;
  RetCode    : GeneralReturn
end;
ptrRepMsg  = ^RepMessage;

var
  MyMsg : MyMessage;
  RepMsgP : ptrRepMsg;

begin
  RepMsgP := RECAST(ReP,ptrRepMsg);
  with MyMsg.head do
    begin
      Simplemsg := FALSE;
      MsgSize:= 50;
      MsgType:= NormalMsg;
      RemotePort := ServPort;
      LocalPort := ReplyPort;
      ID := 2813
    end;
  with MyMsg do
    begin
      IPCNam2.InLine := FALSE;
      IPCNam2.Deallocate := FALSE;
      IPCNam2.LongForm := TRUE;
      TName2 := 8;
      TSize2 := 8;
      NumElts2 := char_count;
      Arg2 := chars;
      IPCNam3.InLine := TRUE;
      IPCNam3.Deallocate := FALSE;
      IPCNam3.LongForm := FALSE;
      IPCNam3.TypeName := 1;
      IPCNam3.TypeSizeInBits := 16;
      IPCNam3.NumObjects := 1;
      Arg3 := firstCh;
      IPCNam4.InLine := TRUE;
      IPCNam4.Deallocate := FALSE;
      IPCNam4.LongForm := FALSE;
      IPCNam4.TypeName := 1;
      IPCNam4.TypeSizeInBits := 16;
      IPCNam4.NumObjects := 1;
      Arg4 := lastCh;
    end;
  GR := Send(MyMsg.head,0,WAIT);
  if GR <> Success then
    begin
      Raise GSError(GR);
      exit(STSPutCharArray)
    end;
end;

Function TSAsynch(InP : POINTER): boolean;
```

```
var
  InMsgP : ptrMsg;

begin
  InMsgP := RECAST(InP.ptrMsg);
  with InMsgP do
    begin
      TSAsynch := True;
      case shrink(ID) of
        Otherwise: TSAsynch := False
      end
    end
  end;

Procedure TSExceptions;
begin
  with ReP^.head do
    case shrink(ID) of
      Otherwise: raise GRError(BADREPLY)
    end
end.
```

9.2.3. TSServer.pas

This file is the Pascal server interface.

```
Module TSServer;

{ -----> } Exports { <----- }

Imports AccInt from AccIntUser;
Imports TSDefs from TSDefs;
Imports STypescript from STypescript;
Imports TSTypes from TSTypes;

Function TSServer(InP,RepP : POINTER) : boolean;

{ -----> } Private { <----- }

Imports AccCall from AccCall;
Imports PascalInit from PascalInit;

var
  ReplyPort: port;
  GR       : GeneralReturn;
  ReP      : Pointer;

Procedure XSTSOpen(InP, ReP : POINTER);

type
  MyMessage = record
    head   : Msg;
    IPCNam2 : TypeType;
```

```
Arg2      : Viewport;
IPCNam3  : TypeType;
Arg3      : port
end;

type
RepMessage = record
  head      : Msg;
  RetCodeType : TypeType;
  RetCode   : GeneralReturn;
  IPCNam4  : TypeType;
  Arg4     : Typescript
end;
type
ptrMyMsg  = ^MyMessage;
ptrRepMsg = ^RepMessage;
var
MyMsgP   : ptrMyMsg;
RepMsgP  : ptrRepMsg;
ServPort : Port;
vp       : Viewport;
env     : port;

begin
MyMsgP := RECAST(InP.ptrMyMsg);
RepMsgP := RECAST(ReP.ptrRepMsg);
RepMsgP^.RetCode := Success;
with RepMsgP^.head do
begin
  Simplemsg := FALSE;
  MsgSize := 36;
  MsgType := NormalMsg;
  ID := 2900
end;
with MyMsgP^ do
begin
  ServPort := head.LocalPort;
  if IPCNam2.TypeName <> 6 then
    begin
      RepMsgP^.RetCode := WRONGARGS;
      exit(XSTSOpen)
    end;
  vp := Arg2;
  if IPCNam3.TypeName <> 6 then
    begin
      RepMsgP^.RetCode := WRONGARGS;
      exit(XSTSOpen)
    end;
  env := Arg3;
end;
RepMsgP^.Arg4 := STSOpen(
  ServPort,
  vp,
  env);
with RepMsgP^ do
begin
```

```
IPCNam4.InLine := TRUE;
IPCNam4.Deallocate := FALSE;
IPCNam4.LongForm := FALSE;
IPCNam4.TypeName := 6;
IPCNam4.TypeSizeInBits := 32;
IPCNam4.NumObjects := 1;
end
end;

Procedure XSTSOpenWindow(InP, ReP : POINTER);

type
  MyMessage = record
    head      : Msg;
    IPCNam2 : TypeType;
    Arg2     : Window;
    IPCNam3 : TypeType;
    Arg3     : port
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode    : GeneralReturn;
    IPCNam4 : TypeType;
    Arg4     : Typescript
  end;
type
  ptrMyMsg  = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP   : ptrMyMsg;
  RepMsgP  : ptrRepMsg;
  ServPort : Port;
  w        : Window;
  env      : port;

begin
  MyMsgP := RECAST(InP,ptrMyMsg);
  RepMsgP := RECAST(ReP,ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with RepMsgP^.head do
    begin
      Simplemsg := FALSE;
      MsgSize := 36;
      MsgType := NormalMsg;
      ID := 2901
    end;
  with MyMsgP^ do
    begin
      ServPort := head.LocalPort;
      if IPCNam2.TypeName <> 6 then
        begin
          RepMsgP^.RetCode := WRONGARGS;
        end;
    end;
end;
```

```
        exit(XSTSOpenWindow)
      end;
      w := Arg2;
      if IPCNam3.TypeName <> 6 then
        begin
          RepMsgP^.RetCode := WRONGARGS;
          exit(XSTSOpenWindow)
        end;
      env := Arg3;
      end;
      RepMsgP^.Arg4 := STSOpenWindow(
        ServPort,
        w,
        env);
      with RepMsgP^ do
        begin
          IPCNam4.InLine := TRUE;
          IPCNam4.Deallocate := FALSE;
          IPCNam4.LongForm := FALSE;
          IPCNam4.TypeName := 6;
          IPCNam4.TypeSizeInBits := 32;
          IPCNam4.NumObjects := 1;
        end
      end;

Procedure XSTSFullOpen(InP, ReP : POINTER);

type
  MyMessage = record
    head : Msg;
    IPCNam2 : TypeType;
    Arg2 : Viewport;
    IPCNam3 : TypeType;
    Arg3 : port;
    IPCNam4 : TypeType;
    Arg4 : TSString255;
    IPCNam5 : TypeType;
    Arg5 : boolean;
    IPCNam6 : TypeType;
    Arg6 : Integer
  end;

type
  RepMessage = record
    head : Msg;
    RetCodeType : TypeType;
    RetCode : GeneralReturn;
    IPCNam7 : TypeType;
    Arg7 : Typescript
  end;
type
  ptrMyMsg = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP : ptrMyMsg;
```

```
RepMsgP : ptrRepMsg;
ServPort : Port;
vp : Viewport;
env : port;
fontName : TSString255;
doWrap : boolean;
dispPages : Integer;

begin
  MyMsgP := RECAST(InP.ptrMyMsg);
  RepMsgP := RECAST(ReP.ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with RepMsgP^.head do
    begin
      Simplemsg := FALSE;
      MsgSize := 36;
      MsgType := NormalMsg;
      ID := 2902;
    end;
  with MyMsgP do
    begin
      ServPort := head.LocalPort;
      if IPCNam2.TypeName <> 6 then
        begin
          RepMsgP^.RetCode := WRONGARGS;
          exit(XSTSFullOpen)
        end;
      vp := Arg2;
      if IPCNam3.TypeName <> 6 then
        begin
          RepMsgP^.RetCode := WRONGARGS;
          exit(XSTSFullOpen)
        end;
      env := Arg3;
      if IPCNam4.TypeName <> 0 then
        begin
          RepMsgP^.RetCode := WRONGARGS;
          exit(XSTSFullOpen)
        end;
      fontName := Arg4;
      if IPCNam5.TypeName <> 0 then
        begin
          RepMsgP^.RetCode := WRONGARGS;
          exit(XSTSFullOpen)
        end;
      doWrap := Arg5;
      if IPCNam6.TypeName <> 1 then
        begin
          RepMsgP^.RetCode := WRONGARGS;
          exit(XSTSFullOpen)
        end;
      dispPages := Arg6;
    end;
  RepMsgP^.Arg7 := STSFullOpen(
    ServPort,
    vp,
```

```
    env,
    fontName,
    doWrap,
    dispPages);
with RepMsgP^ do
begin
  IPCNam7.InLine := TRUE;
  IPCNam7.Deallocate := FALSE;
  IPCNam7.LongForm := FALSE;
  IPCNam7.TypeName := 6;
  IPCNam7.TypeSizeInBits := 32;
  IPCNam7.NumObjects := 1;
end
end;

Procedure XSTSFullOpenWindow(InP, ReP : POINTER);

type
  MyMessage = record
    head : Msg;
    IPCNam2 : TypeType;
    Arg2 : Window;
    IPCNam3 : TypeType;
    Arg3 : port;
    IPCNam4 : TypeType;
    Arg4 : TSString255;
    IPCNam5 : TypeType;
    Arg5 : boolean;
    IPCNam6 : TypeType;
    Arg6 : Integer
  end;

type
  RepMessage = record
    head : Msg;
    RetCodeType : TypeType;
    RetCode : GeneralReturn;
    IPCNam7 : TypeType;
    Arg7 : Typescript
  end;
type
  ptrMyMsg = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP : ptrMyMsg;
  RepMsgP : ptrRepMsg;
  ServPort : Port;
  w : Window;
  env : port;
  fontName : TSString255;
  doWrap : boolean;
  dispPages : Integer;

begin
  MyMsgP := RECAST(InP,ptrMyMsg);
```

```
RepMsgP := RECAST(ReP.ptrRepMsg);
RepMsgP^.RetCode := Success;
with RepMsgP^.head do
begin
  Simplemsg := FALSE;
  MsgSize := 36;
  MsgType := NormalMsg;
  ID := 2903
end;
with MyMsgP^ do
begin
  ServPort := head.LocalPort;
  if IPCNam2.TypeName <> 6 then
    begin
      RepMsgP^.RetCode := WRONGARGS;
      exit(XSTSFullOpenWindow)
    end;
  w := Arg2;
  if IPCNam3.TypeName <> 6 then
    begin
      RepMsgP^.RetCode := WRONGARGS;
      exit(XSTSFullOpenWindow)
    end;
  env := Arg3;
  if IPCNam4.TypeName <> 0 then
    begin
      RepMsgP^.RetCode := WRONGARGS;
      exit(XSTSFullOpenWindow)
    end;
  fontName := Arg4;
  if IPCNam5.TypeName <> 0 then
    begin
      RepMsgP^.RetCode := WRONGARGS;
      exit(XSTSFullOpenWindow)
    end;
  doWrap := Arg5;
  if IPCNam6.TypeName <> 1 then
    begin
      RepMsgP^.RetCode := WRONGARGS;
      exit(XSTSFullOpenWindow)
    end;
  dispPages := Arg6;
end;
RepMsgP^.Arg7 := STSFullOpenWindow(
  ServPort,
  w,
  env,
  fontName,
  doWrap,
  dispPages);
with RepMsgP^ do
begin
  IPCNam7.InLine := TRUE;
  IPCNam7.Deallocate := FALSE;
  IPCNam7.LongForm := FALSE;
  IPCNam7.TypeName := 6;
```

```
IPCNam7.TypeSizeInBits := 32;
IPCNam7.NumObjects := 1;
end
end;

Procedure XSTSFullLine(InP, ReP : POINTER);

type
  MyMessage = record
    head : Msg
  end;

type
  RepMessage = record
    head : Msg;
    RetCodeType : TypeType;
    RetCode : GeneralReturn;
    IPCNam2 : TypeType;
    Arg2 : boolean
  end;
type
  ptrMyMsg = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP : ptrMyMsg;
  RepMsgP : ptrRepMsg;
  ServPort : Port;

begin
  MyMsgP := RECAST(InP.ptrMyMsg);
  RepMsgP := RECAST(ReP.ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with RepMsgP^.head do
  begin
    Simplemsg := TRUE;
    MsgSize := 34;
    MsgType := NormalMsg;
    ID := 2904
  end;
  with MyMsgP^ do
  begin
    ServPort := head.LocalPort;
  end;
  RepMsgP^.Arg2 := STSFullLine(
    ServPort);
  with RepMsgP^ do
  begin
    IPCNam2.InLine := TRUE;
    IPCNam2.Deallocate := FALSE;
    IPCNam2.LongForm := FALSE;
    IPCNam2.TypeName := 0;
    IPCNam2.TypeSizeInBits := 1;
    IPCNam2.NumObjects := 1;
  end
end;
```

```
Procedure XSTSGetChar(InP, ReP : POINTER);

type
  MyMessage = record
    head      : Msg
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode   : GeneralReturn;
    IPCNam2 : TypeType;
    Arg2     : char
  end;
type
  ptrMyMsg  = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP : ptrMyMsg;
  RepMsgP : ptrRepMsg;
  ServPort : Port;

begin
  MyMsgP := RECAST(InP.ptrMyMsg);
  RepMsgP := RECAST(ReP.ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with RepMsgP^.head do
    begin
      Simplemsg := TRUE;
      MsgSize := 34;
      MsgType := NormalMsg;
      ID := 2905
    end;
  with MyMsgP^ do
    begin
      ServPort := head.LocalPort;
    end;
  RepMsgP^.Arg2 := STSGetChar(
    ServPort);
  with RepMsgP^ do
    begin
      IPCNam2.InLine := TRUE;
      IPCNam2.Deallocate := FALSE;
      IPCNam2.LongForm := FALSE;
      IPCNam2.TypeName := 8;
      IPCNam2.TypeSizeInBits := 8;
      IPCNam2.NumObjects := 1;
    end
end;

Procedure XSTSGetString(InP, ReP : POINTER);

type
  MyMessage = record
```

```
head    : Msg;
end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode    : GeneralReturn;
    IPCNam2   : TypeType;
    Arg2      : TSString255
  end;
type
  ptrMyMsg  = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP : ptrMyMsg;
  RepMsgP : ptrRepMsg;
  ServPort : Port;

begin
  MyMsgP := RECAST(InP,ptrMyMsg);
  RepMsgP := RECAST(ReP,ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with RepMsgP^.head do
    begin
      Simplemsg := TRUE;
      MsgSize := 288;
      MsgType := NormalMsg;
      ID := 2906
    end;
  with MyMsgP^ do
    begin
      ServPort := head.LocalPort;
    end;
  RepMsgP^.Arg2 := STSGetString(
    ServPort);
  with RepMsgP^ do
    begin
      IPCNam2.InLine := TRUE;
      IPCNam2.Deallocate := FALSE;
      IPCNam2.LongForm := FALSE;
      IPCNam2.TypeName := 0;
      IPCNam2.SizeTypeInBits := 8;
      IPCNam2.NumObjects := 256;
    end
  end;

Procedure XSTSPutChar(InP, ReP : POINTER);

type
  MyMessage = record
    head    : Msg;
    IPCNam2 : TypeType;
    Arg2    : char
  end;
```

```
type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode    : GeneralReturn
  end;
type
  ptrMyMsg  = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP   : ptrMyMsg;
  RepMsgP  : ptrRepMsg;
  ServPort : Port;
  ch       : char;

begin
  MyMsgP := RECAST(InP.ptrMyMsg);
  RepMsgP := RECAST(ReP.ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with MyMsgP do
    begin
      ServPort := head.LocalPort;
      if IPCNam2.TypeName <> 8 then
        begin
          RepMsgP^.RetCode := WRONGARGS;
          exit(XSTSPutChar)
        end;
      ch := Arg2;
    end;
  STSPutChar(
    ServPort,
    ch);
  RepMsgP^.RetCode := NoReply
end;

Procedure XSTSPutString(InP, ReP : POINTER);

type
  MyMessage = record
    head      : Msg;
    IPCNam2 : TypeType;
    Arg2     : TString255
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode    : GeneralReturn
  end;
type
  ptrMyMsg  = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP   : ptrMyMsg;
```

```
RepMsgP : ptrRepMsg;
ServPort : Port;
s : TSString255;

begin
  MyMsgP := RECAST(InP.ptrMyMsg);
  RepMsgP := RECAST(ReP.ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with MyMsgP^ do
    begin
      ServPort := head.LocalPort;
      if IPCNam2.TypeName <> 0 then
        begin
          RepMsgP^.RetCode := WRONGARGS;
          exit(XSTSPutString)
        end;
      s := Arg2;
    end;
  STSPutString(
    ServPort.
    s);
  RepMsgP^.RetCode := NoReply
end;

Procedure XSTSFlushInput(InP, ReP : POINTER);

type
  MyMessage = record
    head : Msg
  end;

type
  RepMessage = record
    head : Msg;
    RetCodeType : TypeType;
    RetCode : GeneralReturn
  end;
type
  ptrMyMsg = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP : ptrMyMsg;
  RepMsgP : ptrRepMsg;
  ServPort : Port;

begin
  MyMsgP := RECAST(InP.ptrMyMsg);
  RepMsgP := RECAST(ReP.ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with MyMsgP^ do
    begin
      ServPort := head.LocalPort;
    end;
  STSFlushInput(
    ServPort);
```

```
RepMsgP^.RetCode := NoReply
end;

Procedure XSTSFlushOutput(InP, ReP : POINTER);

type
  MyMessage = record
    head      : Msg;
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode     : GeneralReturn;
  end;
type
  ptrMyMsg  = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP   : ptrMyMsg;
  RepMsgP : ptrRepMsg;
  ServPort : Port;

begin
  MyMsgP := RECAST(InP,ptrMyMsg);
  RepMsgP := RECAST(ReP.ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with RepMsgP^.head do
    begin
      Simplemsg := TRUE;
      MsgSize := 28;
      MsgType := NormalMsg;
      ID := 2910;
    end;
  with MyMsgP^ do
    begin
      ServPort := head.LocalPort;
    end;
  STSFlushOutput(
    ServPort);
  with RepMsgP^ do
    begin
    end
end;

Procedure XSTSChangeEnv(InP, ReP : POINTER);

type
  MyMessage = record
    head      : Msg;
    IPCNam2  : TypeType;
    Arg2      : port;
  end;
```

```
type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode    : GeneralReturn
  end;
type
  ptrMyMsg  = ^MyMessage;
  ptrRepMsg = ^RepMessage;
var
  MyMsgP   : ptrMyMsg;
  RepMsgP : ptrRepMsg;
  ServPort : Port;
  env : port;

begin
  MyMsgP := RECAST(InP,ptrMyMsg);
  RepMsgP := RECAST(ReP,ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with MyMsgP^ do
    begin
      begin
        ServPort := head.LocalPort;
        if IPCNam2.TypeName <> 6 then
          begin
            RepMsgP^.RetCode := WRONGARGS;
            exit(XSTSChangeEnv)
          end;
        env := Arg2;
      end;
      STSChangeEnv(
        ServPort,
        env);
    RepMsgP^.RetCode := NoReply
  end;

Procedure XSTSGrabWindow(InP, ReP : POINTER);

type
  MyMessage = record
    head      : Msg;
    IPCNam2 : TypeType;
    Arg2     : port
  end;

type
  RepMessage = record
    head      : Msg;
    RetCodeType : TypeType;
    RetCode    : GeneralReturn;
    IPCNam3 : TypeType;
    Arg3     : Window
  end;
type
  ptrMyMsg  = ^MyMessage;
  ptrRepMsg = ^RepMessage;
```

```
var
  MyMsgP : ptrMyMsg;
  RepMsgP : ptrRepMsg;
  ServPort : Port;
  kPort : port;

begin
  MyMsgP := RECAST(InP.ptrMyMsg);
  RepMsgP := RECAST(ReP.ptrRepMsg);
  RepMsgP^.RetCode := Success;
  with RepMsgP^.head do
    begin
      Simplemsg := FALSE;
      MsgSize := 36;
      MsgType := NormalMsg;
      ID := 2912;
    end;
  with MyMsgP^ do
    begin
      ServPort := head.LocalPort;
      if IPCNam2.TypeName <> 6 then
        begin
          RepMsgP^.RetCode := WRONGARGS;
          exit(XSTSGrabWindow)
        end;
      kPort := Arg2;
    end;
  RepMsgP^.Arg3 := STSGrabWindow(
    ServPort,
    kPort);
  with RepMsgP^ do
    begin
      IPCNam3.InLine := TRUE;
      IPCNam3.Deallocate := FALSE;
      IPCNam3.LongForm := FALSE;
      IPCNam3.TypeName := 6;
      IPCNam3.TypeSizeInBits := 32;
      IPCNam3.NumObjects := 1;
    end
end;

Procedure XSTSPutCharArray(InP, ReP : POINTER);

type
  MyMessage = record
    head : Msg;
    IPCNam2 : TypeType;
    TName2 : integer;
    TSize2 : integer;
    NumElts2: long;
    Arg2 : pTSCharArray;
    IPCNam3 : TypeType;
    Arg3 : Integer;
    IPCNam4 : TypeType;
    Arg4 : Integer
```

```
    end;

    type
      RepMessage = record
        head       : Msg;
        RetCodeType : TypeType;
        RetCode     : GeneralReturn
      end;
    type
      ptrMyMsg  = ^MyMessage;
      ptrRepMsg = ^RepMessage;
  var
    MyMsgP   : ptrMyMsg;
    RepMsgP  : ptrRepMsg;
    ServPort : Port;
    chars : pTSCharArray;
    char_count : long;
    firstCh : Integer;
    lastCh : Integer;

  begin
    MyMsgP := RECAST(InP.ptrMyMsg);
    RepMsgP := RECAST(ReP.ptrRepMsg);
    RepMsgP^.RetCode := Success;
    with MyMsgP^ do
      begin
        ServPort := head.LocalPort;
        if TName2 <> 8 then
          begin
            RepMsgP^.RetCode := WRONGARGS;
            exit(XSTSPutCharArray)
          end;
        char_count := NumElts2;
        chars := Arg2;
        if IPCNam3.TypeName <> 1 then
          begin
            RepMsgP^.RetCode := WRONGARGS;
            exit(XSTSPutCharArray)
          end;
        firstCh := Arg3;
        if IPCNam4.TypeName <> 1 then
          begin
            RepMsgP^.RetCode := WRONGARGS;
            exit(XSTSPutCharArray)
          end;
        lastCh := Arg4;
      end;
    STSPutCharArray(
      ServPort,
      chars,
      char_count,
      firstCh,
      lastCh);
    RepMsgP^.RetCode := NoReply
  end;
```

```
Function TSServer(InP, RepP : POINTER): boolean;
type
  RepMessage = record
    head        : Msg;
    RetCodeType : TypeType;
    RetCode     : GeneralReturn;
  end;
  ptrRepMessage = ^RepMessage;
var
  InMsgP   : ptrMsg;
  RepMsgP : ptrRepMessage;

begin
  InMsgP := RECAST(InP,ptrMsg);
  RepMsgP := RECAST(RepP,ptrRepMessage);
  with RepMsgP^.Head do
    begin
      LocalPort := InMsgP^.LocalPort;
      RemotePort := InMsgP^.RemotePort
    end;
  with RepMsgP^.RetCodeType do
    begin
      TypeName := TYPEINT16;
      TypeSizeInBits := 16;
      NumObjects := 1;
      InLine := true;
      LongForm := false;
      Deallocate := false
    end;
  TSServer := True;
  with InMsgP^ do
    case shrink(ID) of
      2800: XSTSOpen(InP,RepP);
      2801: XSTSOpenWindow(InP,RepP);
      2802: XSTSFULLOpen(InP,RepP);
      2803: XSTSFULLOpenWindow(InP,RepP);
      2804: XSTSFullLine(InP,RepP);
      2805: XSTSGetChar(InP,RepP);
      2806: XSTSGetString(InP,RepP);
      2807: XSTSPutChar(InP,RepP);
      2808: XSTSPutString(InP,RepP);
      2809: XSTSFlushInput(InP,RepP);
      2810: XSTSFlushOutput(InP,RepP);
      2811: XSTSChangeEnv(InP,RepP);
      2812: XSTSGrabWindow(InP,RepP);
      2813: XSTSPutCharArray(InP,RepP);
      otherwise: begin
        TSServer := False;
        RepMsgP^.RetCode := BADMSGID
      end
    end
  end.
end.
```

9.2.4. TSDefs.lisp

This file contains Accent Lisp types.

```
;::: -*-Lisp-*-  
;:::  
;::: Matchmaker generated types and constants definitions  
;::: file for TS  
;:::  
;:::  
;::: THIS FILE SHOULD NOT BE HAND EDITED  
;:::  
;::: To change this interface, edit the interface matchmaker  
;::: file and run it through matchmaker to generate this file.  
;:::  
  
(in-package "TSDEFS")  
(use-package "MMINTERNALDEFS")  
(use-package "BUILTINDEFS")  
  
(use-package "ACCINTDEFS")  
  
(use-package "SAPPHDEFSDDEFS")  
  
(defoperator (Typescript-op port) ((alien port)) ` (alien-value .  
alien))  
  
(defmacro access-Typescript (alien)  
  (alien-access (Typescript-op , alien) port))  
  
(defoperator (TSSString255-op simple-string) ((alien (perq-string  
255))) ` (alien-value , alien))  
  
(defmacro access-TSSString255 (alien)  
  (alien-access (TSSString255-op , alien) simple-string))  
  
(defmacro access-TSCharArray (a i)  
  (alien-access (index-TSCharArray , a , i) string-char))  
  
(defoperator (index-TSCharArray string-char) ((a TSCharArray) i))`  
  (alien-index (alien-value , a) (* . i 16) 8))  
  
(defoperator (deref-pTSCharArray TSCharArray)  
  ((p (ref TSCharArray)) size-in-bits)`  
  (alien-indirect (alien-value , p) . size-in-bits))
```

```
(defmacro address-pTCharArray (alien)
  (alien-access , alien system-area-pointer))

(export
  '(pTCharArray address-pTCharArray deref-pTCharArray
    TCharArray index-TCharArray access-TCharArray
    TSString255 access-TSString255 TSString255-op Typescript
    access-Typescript Typescript-op))
```

9.2.5. TSUser.slisp

This file is the Accent Lisp user interface.

```
;;;; -*-Lisp-*-
;;;; Matchmaker generated user interface file for TS
;;;;;
;;;; THIS FILE SHOULD NOT BE HAND EDITED
;;;;;
;;;; To change this interface, edit the interface matchmaker
;;;; file and run it through matchmaker to generate this file.
;;;;
;;;

(in-package "TSUSER")
(use-package "TSDEFS")
(use-package "BUILTINDEFS")
(use-package "MMINTERNALDEFS")

;
(use-package "ACCINTUSER")
(defvar *receiveport* NIL)
(defconstant interface-backlog 0)

(use-package "SAPPHDEFSDDEFS")

(use-package "ACCINTDEFS")

;;
;; TS-Send -- internal
;;
;; Send macro for TS
;;
(defmacro TS-Send (msg argblock wait-time send-option)
  "User send macro for interface TS"
  "
  (cond ((fixnump *receiveport*)
          (alien-store (msg-localport-op . msg) port *receiveport*)
          (send , argblock , wait-time , send-option))
        (t
         notaport)))
```

```
;;;  
;; User side interface initialization function.  
;;;  
(defun TS-Init (user-port)  
  "The user side initialization function"  
  (cond ((eql user-port nullport)  
         (multiple-value-bind (gr port) (allocateport kernelport  
                                         interface-backlog)  
             (if (eql gr success) (setq *receiveport* port)  
                 (setq *receiveport* dataport))  
             gr))  
        (t  
         (setq *receiveport* user-port)  
         success)))  
  
;;; STSOpen -- public  
;;;  
;;; User-side remote procedure call interface.  
;;;  
(defun STSOpen (Remote_Port vp env)  
  "The user side of remote procedure STSOpen"  
  (prog (R_e_s_u_l_t  
            (send-waittime 0)  
            (send-option 0)  
            (receive-waittime 0)  
            gr)  
    (alien-bind ((to to-STSOpen to-STSOpen))  
      (alien-store  
        (msg-remoteport-op (Msgsize-to-STSOpen-op  
                           (alien-value to)) port Remote_Port)  
        (alien-store (to-STSOpen-vp-op (alien-value to))  
                    port vp)  
        (alien-store (to-STSOpen-env-op (alien-value to))  
                    port env)  
        (setq gr  
              (TS-send (Msgsize-to-STSOpen-op  
                     (alien-value to)) to-STSOpen  
                     send-waittime send-option)))  
    (alien-bind ((from from-STSOpen from-STSOpen))  
      (cond ((eql gr success)  
             (alien-store  
               (msg-msgsize-op  
                 (Msgsize-from-STSOpen-op (alien-value  
                                           from)))  
               (signed-byte 32) 36)  
             (alien-store  
               (msg-localport-op  
                 (Msgsize-from-STSOpen-op (alien-value  
                                           from)))  
               port *ReceivePort*)  
             (setq gr  
                   (receive from-STSOpen receive-waittime localpt  
                           receiveit))  
             (cond ((not (eql gr success))
```

```
      NIL
      (return gr))
(=access-msg-id
 (Msgsize-from-STSOopen-op (alien-value from)))
 2900)
 (cond ((=access-typepeI-typename
        (msg-retcode-typepeI-op
         (Msgsize-from-STSOopen-op
          (alien-value from))))
 1)
 (setq gr
       (access-msg-retcode
        (Msgsize-from-STSOopen-op
         (alien-value from))))
 (t
  NIL
  (return badreply)))
 (cond ((eql
        (access-typepeI-typename
         (from-STSOopen-R_e_s_u_l_t-typepeI-op
          (alien-value from)))
 6)
 (setq R_e_s_u_l_t
       (access-from-STSOopen-R_e_s_u_l_t
        (alien-value from))))
 (t
  NIL
  (return BADREPLY)))
 NIL
  (return (values R_e_s_u_l_t)))
 (t
  NIL
  (return
   (user-alternate-return
    (Msgsize-from-STSOopen-op (alien-value from)))))))
 (t
  NIL
  (return gr)))))

;;: STSOopenWindow -- public
;;:
;;: User-side remote procedure call interface.
;;:
(defun STSOopenWindow (Remote_Port w env)
  "The user side of remote procedure STSOopenWindow"
  (prog (R_e_s_u_l_t
           (send-waittime 0)
           (send-option 0)
           (receive-waittime 0)
           gr)
        (alien-bind ((to to-STSOopenWindow to-STSOopenWindow))
                   (alien-store
```

```
(msg-remoteport-op (Msgsize-to-STSSopenWindow-op
                     (alien-value to)))
  port Remote_Port)
(alien-store (to-STSSopenWindow-w-op
             (alien-value to)) port w)
(alien-store (to-STSSopenWindow-env-op
             (alien-value to)) port env)
(setq gr
      (TS-send (Msgsize-to-STSSopenWindow-op
                  (alien-value to))
               to-STSSopenWindow send-waittime
               send-option)))
(alien-bind ((from from-STSSopenWindow
                    from-STSSopenWindow))
            (cond ((eql gr success)
                   (alien-store
                     (msg-msgsize-op
                       (Msgsize-from-STSSopenWindow-op
                         (alien-value from)))
                     (signed-byte 32) 36)
                   (alien-store
                     (msg-localport-op
                       (Msgsize-from-STSSopenWindow-op (alien-value from)))
                     port *ReceivePort*)
                   (setq gr
                         (receive from-STSSopenWindow receive-waittime localpt
                                 receiveit))
                   (cond ((not (eql gr success))
                           NIL
                           (return gr))
                         ((=
                           (access-msg-id
                             (Msgsize-from-STSSopenWindow-op
                               (alien-value from)))
                           2901)
                           (cond ((=
                                 (access-typetypei-typename
                                   (msg-retcode-typetypei-op
                                     (Msgsize-from-STSSopenWindow-op
                                       (alien-value from))))
                                 1)
                           (setq gr
                                 (access-msg-retcode
                                   (Msgsize-from-STSSopenWindow-op
                                     (alien-value from)))))

                           (t
                             NIL
                             (return badreply)))
                         (cond ((eql
                                 (access-typetypei-typename
                                   (from-STSSopenWindow-R_e_s_u_l_t-typetypeI-op
                                     (alien-value from)))
                                 6)
                           (setq R_e_s_u_l_t
                                 (access-from-STSSopenWindow-R_e_s_u_l_t
                                   (alien-value from))))
```

```
(t
  NIL
  (return BADREPLY)))
NIL
(return (values R_e_s_u_l_t)))
(t
  NIL
  (return
    (user-alternate-return
      (Msgsize-from-STSSOpenWindow-op (alien-value from))))))
(t
  NIL
  (return gr)))))

;;: STSFullOpen -- public
;;:
;;: User-side remote procedure call interface.
;;:
(defun STSFullOpen (Remote_Port vp env fontName doWrap
  dispPages)
  "The user side of remote procedure STSFullOpen"
  (prog (R_e_s_u_l_t
    (send-waittime 0)
    (send-option 0)
    (receive-waittime 0)
    gr)
    (alien-bind ((to to-STSSFullOpen to-STSSFullOpen))
      (alien-store
        (msg-remoteport-op (Msgsize-to-STSSFullOpen-op
          (alien-value to)) port Remote_Port)
        (alien-store (to-STSSFullOpen-vp-op (alien-value to)) port vp)
        (alien-store (to-STSSFullOpen-env-op (alien-value to))
          port env)
        (alien-store (to-STSSFullOpen-fontName-op (alien-value to))
          simple-string fontName)
        (alien-store (to-STSSFullOpen-doWrap-op (alien-value to))
          boolean doWrap)
        (alien-store (to-STSSFullOpen-dispPages-op (alien-value to))
          (signed-byte 16) dispPages)
        (setq gr
          (TS-send (Msgsize-to-STSSFullOpen-op (alien-value to))
            to-STSSFullOpen send-waittime send-option)))
      (alien-bind ((from from-STSSFullOpen from-STSSFullOpen))
        (cond ((eql gr success)
          (alien-store
            (msg-msgsize-op
              (Msgsize-from-STSSFullOpen-op (alien-value from)))
            (signed-byte 32) 36)
          (alien-store
            (msg-localport-op
              (Msgsize-from-STSSFullOpen-op (alien-value from)))
            port *ReceivePort*)
          (setq gr
            (receive from-STSSFullOpen receive-waittime localpt)
```

```
receiveit))
  (cond ((not (eql gr success)))
        (NIL
         (return gr))
        ((=
          (access-msg-id
           (Msgsize-from-STSFULLOpen-op (alien-value from)))
          2902)
         (cond ((=
                  (access-typepei-typename
                   (msg-retcode-typepei-op
                    (Msgsize-from-STSFULLOpen-op
                     (alien-value from))))
                  1)
                  (setq gr
                        (access-msg-retcode
                         (Msgsize-from-STSFULLOpen-op
                          (alien-value from)))))
          (t
           NIL
           (return badreply)))
        (cond ((=1
                  (access-typepei-typename
                   (from-STSFULLOpen-R_e_s_u_l_t-typepei-op
                    (alien-value from)))
                  6)
                  (setq R_e_s_u_l_t
                        (access-from-STSFULLOpen-R_e_s_u_l_t
                         (alien-value from)))
          (t
           NIL
           (return BADREPLY)))
        NIL
        (return (values R_e_s_u_l_t)))
      (t
       NIL
       (return
        (user-alternate-return
         (Msgsize-from-STSFULLOpen-op (alien-value from)))))))
    (t
     NIL
     (return gr)))))

::: STSFULLOpenWindow -- public
:::
::: User-side remote procedure call interface.
:::
(defun STSFULLOpenWindow (Remote_Port w env fontName
                           doWrap dispPages)
  "The user side of remote procedure STSFULLOpenWindow"
  (prog (R_e_s_u_l_t
          (send-waittime 0)
          (send-option 0)
          (receive-waittime 0)
```

```
        gr)
(alien-bind ((to to-STSFULLOpenWindow to-STSFULLOpenWindow))
  (alien-store
    (msg-remoteport-op
      (Msgsize-to-STSFULLOpenWindow-op (alien-value to)))
    port Remote_Port)
  (alien-store (to-STSFULLOpenWindow-w-op (alien-value to))
    port w)
  (alien-store (to-STSFULLOpenWindow-env-op (alien-value to))
    port env)
  (alien-store (to-STSFULLOpenWindow-fontName-op (alien-value
    to)) simple-string fontName)
  (alien-store (to-STSFULLOpenWindow-dowrap-op (alien-value to))
    boolean dowrap)
  (alien-store (to-STSFULLOpenWindow-disppages-op (alien-value
    to)) (signed-byte 16) dispPages)
  (setq gr
    (TS-send (Msgsize-to-STSFULLOpenWindow-op (alien-value to))
      to-STSFULLOpenWindow send-waittime send-option)))
(alien-bind ((from from-STSFULLOpenWindow
  from-STSFULLOpenWindow))
  (cond ((eql gr success)
    (alien-store
      (msg-msgsize-op
        (Msgsize-from-STSFULLOpenWindow-op (alien-value from)))
      (signed-byte 32) 36)
    (alien-store
      (msg-localport-op
        (Msgsize-from-STSFULLOpenWindow-op (alien-value from)))
      port *ReceivePort*)
    (setq gr
      (receive from-STSFULLOpenWindow receive-waittime
        localpt receiveit))
    (cond ((not (eql gr success))
      NIL
      (return gr))
      ((=
        (access-msg-id
          (Msgsize-from-STSFULLOpenWindow-op
            (alien-value from)))
        2903)
      (cond ((=
        (access-typepei-typename
          (msg-retcode-typepei-op
            (Msgsize-from-STSFULLOpenWindow-op
              (alien-value from))))
        1)
      (setq gr
        (access-msg-retcode
          (Msgsize-from-STSFULLOpenWindow-op
            (alien-value from)))))
      (t
        NIL
        (return badreply)))
      (cond ((eql
        (access-typepei-typename
```

```
(from-STSFULLopenWindow-R_e_s_u_l_t--typetype1--op
  (alien-value from)))
 6)
(setq R_e_s_u_l_t
  (access-from-STSFULLopenWindow-R_e_s_u_l_t
  (alien-value from)))
(t
 NIL
 (return BADREPLY)))
NIL
(return (values R_e_s_u_l_t)))
(t
 NIL
 (return
  (user-alternate-return
   (Msgsize-from-STSFULLopenWindow-op
   (alien-value from))))))
(t
 NIL
 (return gr)))))

;;: STSFULLLine -- public
;;:
;;: User-side remote procedure call interface.
;;
(defun STSFULLLine (Remote_Port)
  "The user side of remote procedure STSFULLLine"
  (prog (R_e_s_u_l_t
    (send-waittime 0)
    (send-option 0)
    (receive-waittime 0)
    gr)
  (alien-bind ((to to-STSFULLLine to-STSFULLLine))
  (alien-store
   (msg-remoteport-op (Msgsize-to-STSFULLLine-op
   (alien-value to)))
   port Remote_Port)
  (setq gr
    (TS-send (Msgsize-to-STSFULLLine-op (alien-value to))
    to-STSFULLLine send-waittime send-option)))
  (alien-bind ((from from-STSFULLLine from-STSFULLLine))
  (cond ((eql gr success)
  (alien-store
   (msg-msgsize-op
   (Msgsize-from-STSFULLLine-op (alien-value from)))
   (signed-byte 32) 34)
  (alien-store
   (msg-localport-op
   (Msgsize-from-STSFULLLine-op (alien-value from)))
   port *ReceivePort*)
  (setq gr
    (receive from-STSFULLLine receive-waittime localpt
    receiveit)))
  (cond ((not (eql gr success))
```

```
      NIL
      (return gr))
((=
  (access-msg-id
    (Msgsize-from-STSFULLLine-op (alien-value from)))
  2904)
(cond ((=
  (access-typepei-typename
    (msg-retcode-typepei-op
      (Msgsize-from-STSFULLLine-op
        (alien-value from))))
  1)
  (setq gr
    (access-msg-retcode
      (Msgsize-from-STSFULLLine-op
        (alien-value from)))))
(t
  NIL
  (return badreply)))
(cond ((eql
  (access-typepei-typename
    (from-STSFULLLine-R_e_s_u_l_t-typepei-op
      (alien-value from)))
  0)
  (setq R_e_s_u_l_t
    (access-from-STSFULLLine-R_e_s_u_l_t
      (alien-value from))))
(t
  NIL
  (return BADREPLY)))
  NIL
  Return (values R_e_s_u_l_t)))
(t
  NIL
  (return
    (user-alternate-return
      (Msgsize-from-STSFULLLine-op (alien-value from))))))
(t
  NIL
  (return gr)))))

;;: STSGetChar -- public
;;
;;: User-side remote procedure call interface.
;;
(defun STSGetChar (Remote_Port)
  "The user side of remote procedure STSGetChar"
  (prog (R_e_s_u_l_t
    (send-waittime 0)
    (send-option 0)
    (receive-waittime 0)
    gr)
  (alien-bind ((to to-STSGetChar to-STSGetChar)))
  (alien-store
```

```
(msg-remoteport-op (Mysize-to-STSGetChar-op
    (alien-value to)))
port Remote_Port)
(setq gr
    (TS-send (Mysize-to-STSGetChar-op (alien-value to))
        to-STSGetChar send-waittime send-option)))
(alien-bind ((from from-STSGetChar from-STSGetChar)))
(cond ((eql gr success)
    (alien-store
        (msg-msgsize-op
            (Mysize-from-STSGetChar-op (alien-value from)))
        (signed-byte 32) 34)
    (alien-store
        (msg-localport-op
            (Mysize-from-STSGetChar-op (alien-value from)))
        port *ReceivePort*)
    (setq gr
        (receive from-STSGetChar receive-waittime localpt
            receiveit)))
    (cond ((not (eql gr success))
        NIL
        (return gr))
    (t=
        (access-msg-id
            (Mysize-from-STSGetChar-op (alien-value from)))
        2905)
        (cond ((=
            (access-typepei-typename
                (msg-retcode-typepei-op
                    (Mysize-from-STSGetChar-op
                        (alien-value from))))
            1)
        (setq gr
            (access-msg-retcode
                (Mysize-from-STSGetChar-op
                    (alien-value from)))))

        (t
            NIL
            (return badreply)))
    (cond ((eql
        (access-typepei-typename
            (from-STSGetChar-R_e_s_u_l_t-typepei-op
                (alien-value from)))
        8)
        (setq R_e_s_u_l_t
            (access-from-STSGetChar-R_e_s_u_l_t
                (alien-value from))))
    (t
        NIL
        (return BADREPLY)))
    NIL
    (return (values R_e_s_u_l_t)))
(t
    NIL
    (return
        (user-alternate-return
```

```
(Msgsize-from-STSGetChar op (alien-value from))))))
(t
 NIL
 (return gr)))))

;;; STSGetString -- public
;;
;;; User-side remote procedure call interface.
;;
(defun STSGetString (Remote_Port)
  "The user side of remote procedure STSGetString"
  (prog (R_e_s_u_l_t
          (send-waittime 0)
          (send-option 0)
          (receive-waittime 0)
          gr)
        (alien-bind ((to to-STSGetString to-STSGetString))
          (alien-store
            (msg-remoteport-op (Msgsize-to-STSGetString-op
              (alien-value to)))
            port Remote_Port)
          (setq gr
            (TS-send (Msgsize-to-STSGetString-op (alien-value to))
              to-STSGetString send-waittime send-option)))
        (alien-bind ((from from-STSGetString from-STSGetString))
          (cond ((eql gr success)
            (alien-store
              (msg-msgsize-op
                (Msgsize-from-STSGetString-op (alien-value from)))
              (signed-byte 32) 288)
            (alien-store
              (msg-localport-op
                (Msgsize-from-STSGetString-op (alien-value from)))
              port *ReceivePort*)
            (setq gr
              (receive from-STSGetString receive-waittime localpt
                receiveit))
            (cond ((not (eql gr success))
              NIL
              (return gr))
              ((=
                (access-msg-id
                  (Msgsize-from-STSGetString-op (alien-value from)))
                2906)
              (cond ((=
                (access-typepei-typename
                  (msg-retcode-typepei-op
                    (Msgsize-from-STSGetString-op
                      (alien-value from)))))
                1)
                (setq gr
                  (access-msg-retcode
                    (Msgsize-from-STSGetString-op
                      (alien-value from)))))))
```

```
(t
  NIL
  (return badreply)))
(cond ((eql
         (access-type:typei->typename
           (from-STSGetString-R_e_s_u_l_t->type:typei->op
             (alien-value from)))
         0)
      (setq R_e_s_u_l_t
            (access->from-STSGetString-R_e_s_u_l_t
              (alien-value from))))
      (t
        NIL
        (return BADREPLY)))
  NIL
  (return (values R_e_s_u_l_t)))
(t
  NIL
  (return
    (user-alternate-return
      (Msgsize->from-STSGetString-op (alien-value from))))))
(t
  NIL
  (return gr)))))

;;: STSPutChar -- public
;;
;;: User-side message interface.
;;
(defun STSPutChar (Remote_Port ch)
  "The user side of message STSPutChar"
  (let ((send-maxwait 0)
        (send-option 0))
    (alien-bind ((to to-STSPutChar to-STSPutChar))
      (alien-store
        (msg-remoteport-op (Msgsize->to-STSPutChar-op
          (alien-value to)))
        port Remote_Port)
      (alien-store (to-STSPutChar-ch-op (alien-value to))
        string-char ch))
    (send to-STSPutChar send-maxwait send-option)))

;;: STSPutString -- public
;;
;;: User-side message interface.
;;
(defun STSPutString (Remote_Port s)
  "The user side of message STSPutString"
  (let ((send-maxwait 0)
        (send-option 0))
    (alien-bind ((to to-STSPutString to-STSPutString))
      (alien-store
```

```
(msg-remoteport-op (Msgsize-to-STSPutString-op
  (alien-value to)))
  port Remote_Port)
(alien-store (to-STSPutString-s-op (alien-value to))
  simple-string s))
(send to-STSPutString send-maxwait send-option))

;;: STSFlushInput -- public
;;
;;: User-side message interface.
;;
(defun STSFlushInput (Remote_Port)
  "The user side of message STSFlushInput"
  (let ((send-maxwait 0)
        (send-option 0))
    (alien-bind ((to to-STSFlushInput to-STSFlushInput))
      (alien-store
        (msg-remoteport-op (Msgsize-to-STSFlushInput-op
          (alien-value to)))
        port Remote_Port))
    (send to-STSFlushInput send-maxwait send-option)))

;;: STSFlushOutput -- public
;;
;;: User-side remote procedure call interface.
;;
(defun STSFlushOutput (Remote_Port)
  "The user side of remote procedure STSFlushOutput"
  (prog ((send-waittime 0)
         (send-option 0)
         (receive-waittime 0)
         gr)
    (alien-bind ((to to-STSFlushOutput to-STSFlushOutput))
      (alien-store
        (msg-remoteport-op
          (Msgsize-to-STSFlushOutput-op (alien-value to)))
        port Remote_Port)
      (setq gr
        (TS-send (Msgsize-to-STSFlushOutput-op (alien-value to))
          to-STSFlushOutput send-waittime send-option)))
    (alien-bind ((from from-STSFlushOutput from-STSFlushOutput))
      (cond ((eql gr success)
             (alien-store
               (msg-msgsize-op
                 (Msgsize-from-STSFlushOutput-op (alien-value from)))
               (signed-byte 32) 28)
             (alien-store
               (msg-localport-op
                 (Msgsize-from-STSFlushOutput-op (alien-value from)))
               port *ReceivePort*)
             (setq gr
               (receive from-STSFlushOutput receive-waittime
```

```
localpt receiveit))
  (cond ((not (eql gr success))
         NIL
         (return gr))
        ( (= (access-msg-id
              (Msgsize-from-STFlushOutput-op (alien-value from)))
              2910)
            (cond ((=
                    (access-typepei-typename
                     (msg-retcode-typepei-op
                      (Msgsize-from-STFlushOutput-op
                       (alien-value from))))
                    1)
           (setq gr
                 (access-msg-retcode
                  (Msgsize-from-STFlushOutput-op
                   (alien-value from)))))

           (t
            NIL
            (return badreply)))
          NIL
          (return (values)))
         (t
          NIL
          (return
           (user-alternate-return
            (Msgsize-from-STFlushOutput-op (alien-value from)))))))
        (t
         NIL
         (return gr)))))

;;: STSChangeEnv -- public
;;
;;: User-side message interface.
;;
(defun STSChangeEnv (Remote_Port env)
  "The user side of message STSChangeEnv"
  (let ((send-maxwait 0)
        (send-option 0))
    (alien-bind ((to to-STSChangeEnv to-STSChangeEnv))
      (alien-store
       (msg-remoteport-op (Msgsize-to-STSSChangeEnv-op
                           (alien-value to))
                           port Remote_Port)
       (alien-store (to-STSSChangeEnv-env-op (alien-value to))
                           port env))
      (send to-STSChangeEnv send-maxwait send-option)))

;;: STSGrabWindow -- public
;;
;;: User-side remote procedure call interface.
```

```
;;
(defun STSGrabWindow (Remote_Port kPort)
  "The user side of remote procedure STSGrabWindow"
  (prog (R_e_s_u_l_t
          (send-waittime 0)
          (send-option 0)
          (receive-waittime 0)
          gr)
    (alien-bind ((to to-STSGrabWindow to-STSGrabWindow))
      (alien-store
        (msg-remoteport-op (Msgsize-to-STSGrabWindow-op
          (alien-value to)))
        port Remote_Port)
      (alien-store (to-STSGrabWindow-kPort-op (alien-value to))
        port kPort)
      (setq gr
        (TS-send (Msgsize-to-STSGrabWindow-op (alien-value to))
          to-STSGrabWindow send-waittime send-option)))
    (alien-bind ((from from-STSGrabWindow from-STSGrabWindow))
      (cond ((eql gr success)
        (alien-store
          (msg-msgsize-op
            (Msgsize-from-STSGrabWindow-op (alien-value from)))
          (signed-byte 32) 36)
        (alien-store
          (msg-localport-op
            (Msgsize-from-STSGrabWindow-op (alien-value from)))
          port *ReceivePort*)
        (setq gr
          (receive from-STSGrabWindow receive-waittime localpt
            receiveit)))
      (cond ((not (eql gr success)))
        NIL
        (return gr))
      (= (access-msg-id
          (Msgsize-from-STSGrabWindow-op (alien-value from)))
          2912)
        (cond ((=
          (access-typetypei-typename
            (msg-retcode-typetypei-op
              (Msgsize-from-STSGrabWindow-op
                (alien-value from))))
          1)
        (setq gr
          (access-msg-retcode
            (Msgsize-from-STSGrabWindow-op
              (alien-value from)))))
      (t
        NIL
        (return badreply)))
      (cond ((eql
          (access-typetypei-typename
            (from-STSGrabWindow-R_e_s_u_l_t-typetypeI-op
              (alien-value from)))
          6)
```

```
(setq R_e_s_u_l_t
      (access-from-STSGrabWindow-R_e_s_u_l_t
        (alien-value from)))
  (t
    NIL
    (return BADREPLY)))
  NIL
  (return (values R_e_s_u_l_t)))
(t
 NIL
 (return
  (user-alternate-return
   (Msgsize-from-STSGrabWindow-op (alien-value from))))))
(t
 NIL
 (return gr)))))

;;: STSPutCharArray -- public
;;
;;: User-side message interface.
;;
(defun STSPutCharArray (Remote_Port chars char_count firstCh
  lastCh)
  "The user side of message STSPutCharArray"
  (let ((send-maxwait 0)
        (send-option 0))
    (alien-bind ((to to-STSPutCharArray to-STSPutCharArray))
      (alien-store
       (msg-remoteport-op
        (Msgsize-to-STSPutCharArray-op (alien-value to)))
       port Remote_Port)
      (alien-store
       (to-STSPutCharArray-chars-TypeNumObjects-Op
        (alien-value to))
       (signed-byte 32) char_count)
      (alien-store (to-STSPutCharArray-chars-op
        (alien-value to))
       system-area-pointer chars)
      (alien-store (to-STSPutCharArray-firstCh-op
        (alien-value to))
       (signed-byte 16) firstCh)
      (alien-store (to-STSPutCharArray-lastCh-op
        (alien-value to))
       (signed-byte 16) lastCh))
    (send to-STSPutCharArray send-maxwait send-option)))

;;: TSAsynch -- internal
;;
;;: User-side asynchronous message dispatching function.
;;
(defun TSAsynch (msg)
  "User-side asynchronous message dispatching function."
```

```
(alien-bind ((alien-msg msg msg))
  (case (access-msg-id (alien-value alien-msg))
    (t badmsgid)))

;; User-Alternate-Return -- internal
;;
;; User-side alternate return dispatching function.
;;
(defun user-alternate-return (msg)
  "User-side alternate return dispatching function."
  (alien-bind ((alien-msg msg msg))
    (case (access-msg-id (alien-value alien-msg))
      (t badreply)))

(export
  '(TS-Init TSAsynch STSOpen STSOpenWindow STSFULLOpen
  STSFULLOpenWindow STSFULLLine STSGetChar STSGetString
  STSPutChar STSPutString STSFlushInput STSFlushOutput
  STSChangeEnv STSGrabWindow STSPutCharArray))
```

9.2.6. TSMsgDefs.slisp

This file contains the Accent Lisp message definitions and is for compilation only.

```
;;;; -*-Lisp-*-
;;;; Matchmaker generated msg definitions file for TS
;;;;;
;;;;;
;;;; THIS FILE SHOULD NOT BE HAND EDITED
;;;;;
;;;; To change this interface, edit the interface matchmaker
;;;; file and run it through matchmaker to generate this file.
;;;;;

(in-package "TSDEFS")
(use-package "BUILTINDEFS")
(use-package "MMINTERNALDEFS")

(use-package "SAPPHDEFSDEFS")
(use-package "ACCINTDEFS")

(defoperator (Msgsize-to-STSOpen-op msg) ((specific to-STSOpen))
  (alien-index (alien-value , specific) (bits 0) (bits 304)))

(defoperator (to-STSOpen-ize-op to-STSOpen) ((generic msg))
  (alien-index (alien-value . generic) (bits 0) (bits 304)))
```

```
(defoperator (to-STSOpen-vp-op port) ((f to-STSOpen))
  (alien-index (alien-value . f) (bits 208) (bits 32)))

(defmacro Access-to-STSOpen-vp (r)
  (alien-access (to-STSOpen-vp-op , r) port))

(defoperator (to-STSOpen-vp-TypeTypeI-op (signed-byte 32))
  ((f to-STSOpen)) ` (alien-index (alien-value , f)
  (bits 176) (bits 32)))

(defoperator (to-STSOpen-env-op port) ((f to-STSOpen))
  (alien-index (alien-value , f) (bits 272) (bits 32)))

(defmacro Access-to-STSOpen-env (r)
  (alien-access (to-STSOpen-env-op , r) port))

(defoperator (to-STSOpen-env-TypeTypeI-op (signed-byte 32))
  ((f to-STSOpen)) ` (alien-index (alien-value , f)
  (bits 240) (bits 32)))

(defoperator (Msgsize-from-STSOpen-op msg)
  ((specific from-STSOpen)) ` (alien-index (alien-value ,
  specific) (bits 0) (bits 288)))

(defoperator (from-STSOpen-ize-op from-STSOpen) ((generic msg))
  (alien-index (alien-value , generic) (bits 0) (bits 288)))

(defoperator (from-STSOpen-R_e_s_u_l_t-op port)
  ((f from-STSOpen)) ` (alien-index (alien-value ,
  f) (bits 256) (bits 32)))

(defmacro Access-from-STSOpen-R_e_s_u_l_t (r)
  (alien-access (from-STSOpen-R_e_s_u_l_t-op , r) port))

(defoperator (from-STSOpen-R_e_s_u_l_t-TypeTypeI-op
  (signed-byte 32)) ((f from-STSOpen))
  (alien-index (alien-value , f) (bits 224) (bits 32)))

(defoperator (Msgsize-to-STSOpenWindow-op msg)
  ((specific to-STSOpenWindow)) ` (alien-index
  (alien-value , specific) (bits 0) (bits 304)))

(defoperator (to-STSOpenWindow-ize-op to-STSOpenWindow)
  ((generic msg)) ` (alien-index (alien-value , generic)
  (bits 0) (bits 304)))

(defoperator (to-STSOpenWindow-w-op port)
```

```
((f to-STSSOpenWindow)) ` (alien-index
(alien-value . f) (bits 208) (bits 32))

(defmacro Access-to-STSSOpenWindow-w (r)
  .
  (alien-access (to-STSSOpenWindow-w-op . r) port))

(defoperator (to-STSSOpenWindow-w-TypeTypeI-op (signed-byte 32))
  ((f to-STSSOpenWindow))
  (alien-index (alien-value . f) (bits 176) (bits 32)))

(defoperator (to-STSSOpenWindow-env-op port)
  ((f to-STSSOpenWindow)) ` (alien-index
(alien-value . f) (bits 272) (bits 32)))

(defmacro Access-to-STSSOpenWindow-env (r)
  .
  (alien-access (to-STSSOpenWindow-env-op . r) port))

(defoperator (to-STSSOpenWindow-env-TypeTypeI-op (signed-byte 32))
  ((f to-STSSOpenWindow))
  (alien-index (alien-value . f) (bits 240) (bits 32)))

(defoperator (Msgsize-from-STSSOpenWindow-op msg)
  ((specific from-STSSOpenWindow)) ` (alien-index
(alien-value . specific) (bits 0) (bits 288)))

(defoperator (from-STSSOpenWindow-ize-op from-STSSOpenWindow)
  ((generic msg)) ` (alien-index (alien-value . generic)
(bits 0) (bits 288)))

(defoperator (from-STSSOpenWindow-R_e_s_u_l_t-op port)
  ((f from-STSSOpenWindow)) ` (alien-index (alien-value . f)
(bits 256) (bits 32)))

(defmacro Access-from-STSSOpenWindow-R_e_s_u_l_t (r)
  .
  (alien-access (from-STSSOpenWindow-R_e_s_u_l_t-op . r)
port))

(defoperator (from-STSSOpenWindow-R_e_s_u_l_t-TypeTypeI-op
(signed-byte 32)) ((f from-STSSOpenWindow))
  (alien-index (alien-value . f) (bits 224) (bits 32)))

(defoperator (Msgsize-to-STSFullop-open-op msg)
  ((specific to-STSFullop)) ` (alien-index
(alien-value . specific) (bits 0) (bits 2480)))

(defoperator (to-STSFullop-ize-op to-STSFullop)
  ((generic msg)) ` (alien-index (alien-value . generic)
(bits 0) (bits 2480)))
```

```
(defoperator (to-STSFULLOpen-vp-op port) ((f to-STSFULLOpen))
  (alien-index (alien-value . f) (bits 208) (bits 32)))

(defmacro Access-to-STSFULLOpen-vp (r)
  (alien-access (to-STSFULLOpen-vp-op . r) port))

(defoperator (to-STSFULLOpen-vp-TypeTypeI-op (signed-byte 32))
  ((f to-STSFULLOpen))
  (alien-index (alien-value . f) (bits 176) (bits 32)))

(defoperator (to-STSFULLOpen-env-op port) ((f to-STSFULLOpen))
  (alien-index (alien-value . f) (bits 272) (bits 32)))

(defmacro Access-to-STSFULLOpen-env (r)
  (alien-access (to-STSFULLOpen-env-op . r) port))

(defoperator (to-STSFULLOpen-env-TypeTypeI-op (signed-byte 32))
  ((f to-STSFULLOpen))
  (alien-index (alien-value . f) (bits 240) (bits 32)))

(defoperator (to-STSFULLOpen-fontName-op (perq-string 255))
  ((f to-STSFULLOpen)) ` (alien-index (alien-value . f)
  (bits 336) (bits 2048)))

(defmacro Access-to-STSFULLOpen-fontName (r)
  (alien-access (to-STSFULLOpen-fontName-op . r)
  simple-string))

(defoperator (to-STSFULLOpen-fontName-TypeTypeI-op
  (signed-byte 32)) ((f to-STSFULLOpen))
  (alien-index (alien-value . f) (bits 304) (bits 32)))

(defoperator (to-STSFULLOpen-doWrap-op boolean)
  ((f to-STSFULLOpen)) ` (alien-index (alien-value . f)
  (bits 2416) (bits 1)))

(defmacro Access-to-STSFULLOpen-doWrap (r)
  (alien-access (to-STSFULLOpen-doWrap-op . r) boolean))

(defoperator (to-STSFULLOpen-doWrap-TypeTypeI-op (signed-byte 32))
  ((f to-STSFULLOpen)) ` (alien-index (alien-value . f)
  (bits 2384) (bits 32)))

(defoperator (to-STSFULLOpen-dispPages-op (signed-byte 16))
  ((f to-STSFULLOpen)) ` (alien-index (alien-value . f)
  (bits 2464) (bits 16)))

(defmacro Access-to-STSFULLOpen-dispPages (r)
```

```
(alien-access (to-STSFULLOpen-dispPages-op . r)
  (signed-byte 16)))

(defoperator (to-STSFULLOpen-dispPages-TypeTypeI-op
  (signed-byte 32)) ((f to-STSFULLOpen))
  (alien-index (alien-value . f) (bits 2432) (bits 32)))

(defoperator (Msgsize-from-STSFULLOpen-op msg) ((specific
  from-STSFULLOpen)) ` (alien-index (alien-value . specific)
  (bits 0) (bits 288)))

(defoperator (from-STSFULLOpen-ize-op from-STSFULLOpen)
  ((generic msg)) ` (alien-index (alien-value . generic)
  (bits 0) (bits 288)))

(defoperator (from-STSFULLOpen-R_e_s_u_l_t-op port)
  ((f from-STSFULLOpen)) ` (alien-index (alien-value . f)
  (bits 256) (bits 32!))

(defmacro Access-from-STSFULLOpen-R_e_s_u_l_t (r)
  (alien-access (from-STSFULLOpen-R_e_s_u_l_t-op . r) port))

(defoperator (from-STSFULLOpen-R_e_s_u_l_t-TypeTypeI-op
  (signed-byte 32)) ((f from-STSFULLOpen))
  (alien-index (alien-value . f) (bits 224) (bits 32!)))

(defoperator (Msgsize-to-STSFULLOpenWindow-op msg)
  ((specific to-STSFULLOpenWindow)) ` (alien-index
  (alien-value . specific) (bits 0) (bits 2480!))

(defoperator (to-STSFULLOpenWindow-ize-op to-STSFULLOpenWindow)
  ((generic msg)) ` (alien-index (alien-value . generic)
  (bits 0) (bits 2480!))

(defoperator (to-STSFULLOpenWindow-w-op port)
  ((f to-STSFULLOpenWindow)) ` (alien-index
  (alien-value . f) (bits 208) (bits 32!))

(defmacro Access-to-STSFULLOpenWindow-w (r)
  (alien-access (to-STSFULLOpenWindow-w-op . r) port))

(defoperator (to-STSFULLOpenWindow-w-TypeTypeI-op
  (signed-byte 32)) ((f to-STSFULLOpenWindow))
  (alien-index (alien-value . f) (bits 176) (bits 32!)))

(defoperator (to-STSFULLOpenWindow-env-op port)
  ((f to-STSFULLOpenWindow)) ` (alien-index
  (alien-value . f) (bits 272) (bits 32!))
```

```
(defmacro Access-to-STSFULLopenWindow-env (r)
  (alien-access (to-STSFULLopenWindow-env-op . r) port))

(defoperator (to-STSFULLopenWindow-env-TypeTypeI-op
  (signed-byte 32)) ((f to-STSFULLopenWindow))
  (alien-index (alien-value . f) (bits 240) (bits 32)))

(defoperator (to-STSFULLopenWindow-fontName-op (perq-string 255))
  ((f to-STSFULLopenWindow)) (alien-index
  (alien-value . f) (bits 336) (bits 2048)))

(defmacro Access-to-STSFULLopenWindow-fontName (r)
  (alien-access (to-STSFULLopenWindow-fontName-op . r)
  simple-string))

(defoperator (to-STSFULLopenWindow-fontName-TypeTypeI-op
  (signed-byte 32)) ((f to-STSFULLopenWindow))
  (alien-index (alien-value . f) (bits 304) (bits 32)))

(defoperator (to-STSFULLopenWindow-doWrap-op boolean)
  ((f to-STSFULLopenWindow)) (alien-index (alien-value . f)
  (bits 2416) (bits 1)))

(defmacro Access-to-STSFULLopenWindow-doWrap (r)
  (alien-access (to-STSFULLopenWindow-doWrap-op . r) boolean))

(defoperator (to-STSFULLopenWindow-doWrap-TypeTypeI-op
  (signed-byte 32)) ((f to-STSFULLopenWindow))
  (alien-index (alien-value . f) (bits 2384) (bits 32)))

(defoperator (to-STSFULLopenWindow-dispPages-op (signed-byte 16))
  ((f to-STSFULLopenWindow)) (alien-index
  (alien-value . f) (bits 2464) (bits 16)))

(defmacro Access-to-STSFULLopenWindow-dispPages (r)
  (alien-access (to-STSFULLopenWindow-dispPages-op . r)
  (signed-byte 16)))

(defoperator (to-STSFULLopenWindow-dispPages-TypeTypeI-op
  (signed-byte 32)) ((f to-STSFULLopenWindow))
  (alien-index (alien-value . f) (bits 2432) (bits 32)))

(defoperator (Msgsize-from-STSFULLopenWindow-op msg)
  ((specific from-STSFULLopenWindow)) (alien-index
  (alien-value . specific) (bits 0) (bits 288)))

(defoperator (from-STSFULLopenWindow-ize-op
  from-STSFULLopenWindow) ((generic msg))
  (alien-index (alien-value . generic) (bits 0) (bits 288)))
```

```
(defoperator (from-STSFULLOpenWindow-R_e_s_u_l_t-op port)
  ((f from-STSFULLOpenWindow))
  (alien-index (alien-value . f) (bits 256) (bits 32)))

(defmacro Access-from-STSFULLOpenWindow-R_e_s_u_l_t (r)
  (alien-access (from-STSFULLOpenWindow-R_e_s_u_l_t-op .
  r) port))

(defoperator (from-STSFULLOpenWindow-R_e_s_u_l_t-
  TypeTypeI-op (signed-byte 32)) ((f from-STSFULLOpenWindow))
  (alien-index (alien-value . f) (bits 224) (bits 32)))

(defoperator (Msgsize-to-STSFULLLine-op msg)
  ((specific to-STSFULLline)) ` (alien-index
  (alien-value . specific) (bits 0) (bits 176)))

(defoperator (to-STSFULLLine-ize-op to-STSFULLLine) ((generic msg))
  ` (alien-index (alien-value . generic) (bits 0) (bits 176)))

(defoperator (Msgsize-from-STSFULLLine-op msg)
  ((specific from-STSFULLLine)) ` (alien-index
  (alien-value . specific) (bits 0) (bits 272)))

(defoperator (from-STSFULLLine-ize-op from-STSFULLLine)
  ((generic msg)) ` (alien-index (alien-value . generic)
  (bits 0) (bits 272)))

(defoperator (from-STSFULLLine-R_e_s_u_l_t-op boolean)
  ((f from-STSFULLLine)) ` (alien-index (alien-value . f)
  (bits 256) (bits 1)))

(defmacro Access-from-STSFULLLine-R_e_s_u_l_t (r)
  (alien-access (from-STSFULLLine-R_e_s_u_l_t-op .
  r) boolean))

(defoperator (from-STSFULLLine-R_e_s_u_l_t-TypeTypeI-op
  (signed-byte 32)) ((f from-STSFULLLine))
  (alien-index (alien-value . f) (bits 224) (bits 32)))

(defoperator (Msgsize-to-STSGetChar-op msg)
  ((specific to-STSGetChar)) ` (alien-index
  (alien-value . specific) (bits 0) (bits 176)))

(defoperator (to-STSGetChar-ize-op to-STSGetChar)
  ((generic msg)) ` (alien-index (alien-value .
  generic) (bits 0) (bits 176)))

(defoperator (Msgsize-from-STSGetChar-op msg)
  ((specific from-STSGetChar)) ` (alien-index
  (alien-value . specific) (bits 0) (bits 272)))
```

```
(defoperator (from-STSGetChar-ize-op from-STSGetChar)
  ((generic msg)) ` (alien-index (alien-value . generic)
  (bits 0) (bits 272)))

(defoperator (from-STSGetChar-R_e_s_u_l_t-op string-char)
  ((f from-STSGetChar)) ` (alien-index (alien-value . f)
  (bits 256) (bits 8)))

(defmacro Access-from-STSGetChar-R_e_s_u_l_t (r)
  (alien-access (from-STSGetChar-R_e_s_u_l_t-op . r)
  string-char))

(defoperator (from-STSGetChar-R_e_s_u_l_t-TypeTypeI-op
  (signed-byte 32)) ((f from-STSGetChar))
  (alien-index (alien-value . f) (bits 224) (bits 32)))

(defoperator (Msgsize-to-STSGetString-op msg)
  ((specific to-STSGetString)) ` (alien-index
  (alien-value . specific) (bits 0) (bits 176)))

(defoperator (to-STSGetString-ize-op to-STSGetString)
  ((generic msg)) ` (alien-index (alien-value . generic)
  (bits 0) (bits 176)))

(defoperator (Msgsize-from-STSGetString-op msg)
  ((specific from-STSGetString)) ` (alien-index
  (alien-value . specific) (bits 0) (bits 2304)))

(defoperator (from-STSGetString-ize-op from-STSGetString)
  ((generic msg)) ` (alien-index (alien-value . generic)
  (bits 0) (bits 2304)))

(defoperator (from-STSGetString-R_e_s_u_l_t-op (perq-string
  255)) ((f from-STSGetString)) ` (alien-index
  (alien-value . f) (bits 256) (bits 2048)))

(defmacro Access-from-STSGetString-R_e_s_u_l_t (r)
  (alien-access (from-STSGetString-R_e_s_u_l_t-op . r)
  simple-string))

(defoperator (from-STSGetString-R_e_s_u_l_t-TypeTypeI-op
  (signed-byte 32)) ((f from-STSGetString))
  (alien-index (alien-value . f) (bits 224) (bits 32)))

(defoperator (Msgsize-to-STSPutChar-op msg)
  ((specific to-STSPutChar)) ` (alien-index (alien-value .
  specific) (bits 0) (bits 224)))

(defoperator (to-STSPutChar-ize-op to-STSPutChar) ((generic msg)))
```

```
(alien-index (alien-value . generic) (bits 0) (bits 224)))  
  
(defoperator (to-STSPutChar-ch-op string-char)  
  ((f to-STSPutChar))  
   (alien-index (alien-value . f) (bits 208) (bits 8)))  
  
(defmacro Access-to-STSPutChar-ch (r)  
  ` (alien-access (to-STSPutChar-ch-op . r) string-char))  
  
(defoperator (to-STSPutChar-ch-TypeTypeI-op (signed-byte 32))  
  ((f to-STSPutChar)) ` (alien-index (alien-value . f)  
    (bits 176) (bits 32)))  
  
(defoperator (Msgsize-to-STSPutString-op msg)  
  ((specific to-STSPutString)) ` (alien-index  
    (alien-value . specific) (bits 0) (bits 2256)))  
  
(defoperator (to-STSPutString-ize-op to-STSPutString)  
  ((generic msg)) ` (alien-index (alien-value . generic)  
    (bits 0) (bits 2256)))  
  
(defoperator (to-STSPutString-s-op (perq-string 255))  
  ((f to-STSPutString)) ` (alien-index (alien-value . f)  
    (bits 208) (bits 2048)))  
  
(defmacro Access-to-STSPutString-s (r)  
  ` (alien-access (to-STSPutString-s-op . r) simple-string))  
  
(defoperator (to-STSPutString-s-TypeTypeI-op (signed-byte 32))  
  ((f to-STSPutString)) ` (alien-index (alien-value . f) (bits 176) (bits 32)))  
  
(defoperator (Msgsize-to-STSFlushInput-op msg) ((specific  
  to-STSFlushInput)) ` (alien-index (alien-value . specific)  
    (bits 0) (bits 176)))  
  
(defoperator (to-STSFlushInput-ize-op to-STSFlushInput)  
  ((generic msg)) ` (alien-index (alien-value . generic)  
    (bits 0) (bits 176)))  
  
(defoperator (Msgsize-to-STSFlushOutput-op msg) ((specific  
  to-STSFlushOutput)) ` (alien-index (alien-value . specific)  
    (bits 0) (bits 176)))  
  
(defoperator (to-STSFlushOutput-ize-op to-STSFlushOutput)  
  ((generic msg)) ` (alien-index (alien-value . generic)  
    (bits 0) (bits 176)))
```

```
(defoperator (Msgsize-from-STFlushOutput-op msg) ((specific
    from-STFlushOutput)) ` (alien-index (alien-value .
    specific) (bits 0) (bits 224)))

(defoperator (from-STFlushOutput-ize-op from-STFlushOutput)
    ((generic msg)) ` (alien-index (alien-value . generic)
    (bits 0) (bits 224)))

(defoperator (Msgsize-to-STChangeEnv-op msg) ((specific
    to-STChangeEnv)) ` (alien-index (alien-value . specific)
    (bits 0) (bits 240)))

(defoperator (to-STChangeEnv-ize-op to-STChangeEnv)
    ((generic msg)) ` (alien-index (alien-value . generic)
    (bits 0) (bits 240)))

(defoperator (to-STChangeEnv-env-op port)
    ((f to-STChangeEnv)) ` (alien-index
    (alien-value . f) (bits 208) (bits 32)))

(defmacro Access-to-STChangeEnv-env (r)
    (alien-access (to-STChangeEnv-env-op . r) port))

(defoperator (to-STChangeEnv-env-TypeTypeI-op (signed-byte 32))
    ((f to-STChangeEnv))
    (alien-index (alien-value . f) (bits 176) (bits 32)))

(defoperator (Msgsize-to-STSGrabWindow-op msg)
    ((specific to-STSGrabWindow)) ` (alien-index
    (alien-value . specific) (bits 0) (bits 240)))

(defoperator (to-STSGrabWindow-ize-op to-STSGrabWindow)
    ((generic msg)) ` (alien-index (alien-value . generic)
    (bits 0) (bits 240)))

(defoperator (to-STSGrabWindow-kPort-op port)
    ((f to-STSGrabWindow)) ` (alien-index (alien-value . f)
    (bits 208) (bits 32)))

(defmacro Access-to-STSGrabWindow-kPort (r)
    (alien-access (to-STSGrabWindow-kPort-op . r) port))

(defoperator (to-STSGrabWindow-kPort-TypeTypeI-op
    (signed-byte 32)) ((f to-STSGrabWindow))
    (alien-index (alien-value . f) (bits 176) (bits 32)))

(defoperator (Msgsize-from-STSGrabWindow-op msg)
    ((specific from-STSGrabWindow)) ` (alien-index
    (alien-value . specific) (bits 0) (bits 288)))
```

```
(defoperator (from-STSGrabWindow-ize-op from-STSGrabWindow)
  ((generic msg)) ` (alien-index (alien-value . generic)
  (bits 0) (bits 288)))

(defoperator (from-STSGrabWindow-R_e_s_u_l_t-op port)
  ((f from-STSGrabWindow)) ` (alien-index (alien-value . f)
  (bits 256) (bits 32)))

(defmacro Access-from-STSGrabWindow-R_e_s_u_l_t (r)
  (alien-access (from-STSGrabWindow-R_e_s_u_l_t-op , r)
  port))

(defoperator (from-STSGrabWindow-R_e_s_u_l_t-TypeTypeI-op
  (signed-byte 32)) ((f from-STSGrabWindow))
  (alien-index (alien-value . f) (bits 224) (bits 32)))

(defoperator (Msgsize-to-STSPutCharArray-op msg)
  ((specific to-STSPutCharArray)) ` (alien-index
  (alien-value , specific) (bits 0) (bits 400)))

(defoperator (to-STSPutCharArray-ize-op to-STSPutCharArray)
  ((generic msg)) ` (alien-index (alien-value . generic)
  (bits 0) (bits 400)))

(defoperator (to-STSPutCharArray-chars-op
  system-area-pointer) ((f to-STSPutCharArray)) ` (alien-index (alien-value , f) (bits 272) (bits 32)))

(defmacro Access-to-STSPutCharArray-chars (r)
  (alien-access (to-STSPutCharArray-chars-op , r)
  system-area-pointer))

(defoperator (to-STSPutCharArray-chars-TypeTypeI-op
  (signed-byte 32)) ((f to-STSPutCharArray))
  (alien-index (alien-value . f) (bits 176) (bits 32)))

(defoperator (to-STSPutCharArray-chars-TypeName-op
  (signed-byte 16)) ((f to-STSPutCharArray))
  (alien-index (alien-value . f) (bits 208) (bits 16)))

(defmacro Access-to-STSPutCharArray-chars-TypeName (r)
  (alien-access (to-STSPutCharArray-chars-TypeName-op , r)
  (signed-byte 16)))

(defoperator (to-STSPutCharArray-chars-TypeBitSize-op
  (signed-byte 16)) ((f to-STSPutCharArray))
  (alien-index (alien-value . f) (bits 224) (bits 16)))

(defoperator (to-STSPutCharArray-chars-TypeNumObjects-op
```

```
(signed-byte 32)) ((f to-STSPutCharArray))`  
(alien-index (alien-value . f) (bits 240) (bits 32)))  
  
(defmacro Access-to-STSPutCharArray-chars-TypeNumObjects (r)  
  (alien-access (to-STSPutCharArray-chars-TypeNumObjects-op . r)  
    (signed-byte 32)))  
  
(defoperator (to-STSPutCharArray-firstCh-op  
  (signed-byte 16)) ((f to-STSPutCharArray))`  
(alien-index (alien-value . f) (bits 336) (bits 16)))  
  
(defmacro Access-to-STSPutCharArray-firstCh (r)  
  (alien-access (to-STSPutCharArray-firstCh-op . r)  
    (signed-byte 16)))  
  
(defoperator (to-STSPutCharArray-firstCh-TypeTypeI-op  
  (signed-byte 32)) ((f to-STSPutCharArray))`  
(alien-index (alien-value . f) (bits 304) (bits 32)))  
  
(defoperator (to-STSPutCharArray-lastCh-op  
  (signed-byte 16)) ((f to-STSPutCharArray))`  
(alien-index (alien-value . f) (bits 384) (bits 16)))  
  
(defmacro Access-to-STSPutCharArray-lastCh (r)  
  (alien-access (to-STSPutCharArray-lastCh-op . r)  
    (signed-byte 16)))  
  
(defoperator (to-STSPutCharArray-lastCh-TypeTypeI-op  
  (signed-byte 32)) ((f to-STSPutCharArray))`  
(alien-index (alien-value . f) (bits 352) (bits 32)))  
  
(export  
  '(to-STSPutCharArray-lastCh-TypeTypeI-op  
    Access-to-STSPutCharArray-lastCh  
    to-STSPutCharArray-lastCh-op  
      to-STSPutCharArray-firstCh-TypeTypeI-op  
    Access-to-STSPutCharArray-firstCh to-STSPutCharArray-firstCh-op  
    Access-to-STSPutCharArray-chars-TypeNumObjects  
    to-STSPutCharArray-chars-TypeNumObjects-op  
    to-STSPutCharArray-chars-TypeBitSize-op  
    Access-to-STSPutCharArray-chars-TypeName  
      to-STSPutCharArray-chars-TypeName-op  
    Access-to-STSPutCharArray-chars-TypeTypeI-op  
      Access-to-STSPutCharArray-chars  
    to-STSPutCharArray-chars-op to-STSPutCharArray  
      to-STSPutCharArray-ize-op  
    Msgsize-to-STSPutCharArray-op from-STSGrabWindow  
    from-STSGrabWindow-R_e_s_u_l_t-TypeTypeI-op  
    Access-from-STSGrabWindow-R_e_s_u_l_t  
      from-STSGrabWindow-R_e_s_u_l_t-op
```

```
from-STSGrabWindow from-STSGrabWindow-ize-op
  Msgsize-from-STSGrabWindow-op
  to-STSGrabWindow-kPort-TypeTypeI-op
    Access-to-STSGrabWindow-kPort
  to-STSGrabWindow-kPort-op to-STSGrabWindow
    to-STSGrabWindow-ize-op
  Msgsize-to-STSGrabWindow-op
    to-STSGrabWindow-ize-op
  to-STSGrabWindow-kPort-TypeTypeI-op
  Access-to-STSGrabWindow-kPort to-STSGrabWindow-op
    to-STSGrabWindow
  to-STSGrabWindow-ize-op Msgsize-to-STSGrabWindow-op
    from-STSFFlushOutput
  from-STSFFlushOutput from-STSFFlushOutput-ize-op
    Msgsize-from-STSFFlushOutput-op
  to-STSFFlushOutput to-STSFFlushOutput-ize-op
    Msgsize-to-STSFFlushOutput-op
  to-STSFFlushInput to-STSFFlushInput-ize-op
    Msgsize-to-STSFFlushInput-op
  to-STSPutString-s-TypeTypeI-op Access-to-STSPutString-s
    to-STSPutString-s-op
  to-STSPutString to-STSPutString-ize-op
    Msgsize-to-STSPutString-op
  to-STSPutChar-ch-TypeTypeI-op Access-to-STSPutChar-ch
    to-STSPutChar-ch-op
  to-STSPutChar to-STSPutChar-ize-op Msgsize-to-STSPutChar-op
    from-STSGetString
  from-STSGetString-R_e_s_u_l_t-TypeTypeI-op
    Access-from-STSGetString-R_e_s_u_l_t
  from-STSGetString-R_e_s_u_l_t-op from-STSGetString
    from-STSGetString-ize-op
  Msgsize-from-STSGetString-op to-STSGetString
    to-STSGetString-ize-op
  Msgsize-to-STSGetString-op from-STSGetChar
  from-STSGetChar-R_e_s_u_l_t-TypeTypeI-op
    Access-from-STSGetChar-R_e_s_u_l_t
  from-STSGetChar-R_e_s_u_l_t-op from-STSGetChar
    from-STSGetChar-ize-op
  Msgsize-from-STSGetChar-op to-STSGetChar
    to-STSGetChar-ize-op
  Msgsize-to-STSGetChar-op from-STSFULLLine
  from-STSFULLLine-R_e_s_u_l_t-TypeTypeI-op
    Access-from-STSFULLLine-R_e_s_u_l_t
  from-STSFULLLine-R_e_s_u_l_t-op from-STSFULLLine
    from-STSFULLLine-ize-op
  Msgsize-from-STSFULLLine-op to-STSFULLLine
    to-STSFULLLine-ize-op
  Msgsize-to-STSFULLLine-op from-STSFULLOpenWindow
  from-STSFULLOpenWindow-R_e_s_u_l_t-TypeTypeI-op
  Access-from-STSFULLOpenWindow-R_e_s_u_l_t
    from-STSFULLOpenWindow-R_e_s_u_l_t-op
  from-STSFULLOpenWindow from-STSFULLOpenWindow-ize-op
  Msgsize-from-STSFULLOpenWindow-op
    to-STSFULLOpenWindow-dispPages-TypeTypeI-op
  Access-to-STSFULLOpenWindow-dispPages
    to-STSFULLOpenWindow-dispPages-op
  to-STSFULLOpenWindow-doWrap-TypeTypeI-op
```

```
Access-to--STSFull0openWindow-doWrap
to-STSFull0openWindow-doWrap-op
    to-STSFull0openWindow-fontName-TypeTypeI-op
Access-to--STSFull0openWindow-fontName
    to-STSFull0openWindow-fontName-op
to-STSFull0openWindow-env-TypeTypeI-op
    Access-to--STSFull0openWindow-env
to-STSFull0openWindow-env-op
    to-STSFull0openWindow-w-TypeTypeI-op
Access-to--STSFull0openWindow-w
    to-STSFull0openWindow-w-op to-STSFull0openWindow
to-STSFull0openWindow-ize-op Msgsize-to-STSFull0openWindow-op
from-STSFull0open
from-STSFull0open-R_e_s_u_l_t-TypeTypeI-op
    Access-from-STSFull0open-R_e_s_u_l_t
from-STSFull0open-R_e_s_u_l_t-op from-STSFull0open
    from-STSFull0open-ize-op
Msgsize-from-STSFull0open-op
    to-STSFull0open-dispPages-TypeTypeI-op
Access-to--STSFull0open-dispPages
    to-STSFull0open-dispPages-op
to-STSFull0open-doWrap-TypeTypeI-op
    Access-to--STSFull0open-doWrap
to-STSFull0open-doWrap-op
    to-STSFull0open-fontName-TypeTypeI-op
Access-to--STSFull0open-fontName
    to-STSFull0open-fontName-op
to-STSFull0open-env-TypeTypeI-op
    Access-to--STSFull0open-env to-STSFull0open-env-op
to-STSFull0open-vp-TypeTypeI-op Access-to--STSFull0open-vp
    to-STSFull0open-vp-op
to-STSFull0open to-STSFull0open-ize-op
    Msgsize-to-STSFull0open-op
from-STSOpenWindow
    from-STSOpenWindow-R_e_s_u_l_t-TypeTypeI-op
Access-from-STSOpenWindow-R_e_s_u_l_t
    from-STSOpenWindow-R_e_s_u_l_t-op
from-STSOpenWindow from-STSOpenWindow-ize-op
    Msgsize-from-STSOpenWindow-op
to-STSOpenWindow-env-TypeTypeI-op
    Access-to--STSOpenWindow-env
to-STSOpenWindow-env-op to-STSOpenWindow-w-TypeTypeI-op
Access-to--STSOpenWindow-w
    to-STSOpenWindow-w-op to-STSOpenWindow
to-STSOpenWindow-ize-op Msgsize-to-STSOpenWindow-op
from-STSOpen
from-STSOpen-R_e_s_u_l_t-TypeTypeI-op
    Access-from-STSOpen-R_e_s_u_l_t
from-STSOpen-R_e_s_u_l_t-op from-STSOpen
    from-STSOpen-ize-op
Msgsize-from-STSOpen-op to-STSOpen-env-TypeTypeI-op
    Access-to--STSOpen-env
to-STSOpen-env-op to-STSOpen-vp-TypeTypeI-op
    Access-to--STSOpen-vp
to-STSOpen-vp-op to-STSOpen to-STSOpen-ize-op
    Msgsize-to-STSOpen-op))
```

9.2.7. TSUInit.slisp

This is the Accent Lisp user alien initialization file.

```
;;;; -*-Lisp-*-
;;;; Matchmaker generated user alien initialization file for TS
;;;;;
;;;; THIS FILE SHOULD NOT BE HAND EDITED
;;;;;
;;;; To change this interface, edit the interface matchmaker
;;;; file and run it through matchmaker to generate this file.
;;;;
;;;;
(in-package "TSUSER")
(use-package "TSDEFS")
(use-package "BUILTINDEFS")
(use-package "MMINTERNALDEFS")

(use-package "SAPPHDEFSDEFS")

(use-package "ACCINTDEFS")

(defalien to-STSOpen to-STSOpen 304 :static)

(alien-store (msg-msgsize-op (Msgsize-to-STSOpen-op to-STSOpen))
             (signed-byte 32) 38)

(alien-store (msg-simplemsg-op (Msgsize-to-STSOpen-op
                                 to-STSOpen)) boolean NIL)

(alien-store (msg-msgtype-op (Msgsize-to-STSOpen-op to-STSOpen))
             (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSOpen-op to-STSOpen))
             (signed-byte 32) 2800)
(alien-store (to-STSOpen-vp-type-typeI-op to-STSOpen)
             (signed-byte 32) 268509190)
(alien-store (to-STSOpen-env-type-typeI-op to-STSOpen)
             (signed-byte 32) 268509190)

(defalien from-STSOpen from-STSOpen 288 :static)

(defalien to-STSOpenWindow to-STSOpenWindow 304 :static)

(alien-store (msg-msgsize-op (Msgsize-to-STSOpenWindow-op
                             to-STSOpenWindow)) (signed-byte 32) 38)

(alien-store (msg-simplemsg-op (Msgsize-to-STSOpenWindow-op
                             to-STSOpenWindow)) boolean NIL)
```

```
(alien-store (msg-msgtype-op (Msgsize-to-STSSopenWindow-op
    to-STSSopenWindow)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSSopenWindow-op
    to-STSSopenWindow)) (signed-byte 32) 2801)
(alien-store (to-STSSopenWindow-w-typetypeI-op to-STSSopenWindow)
    (signed-byte 32) 268509190)
(alien-store (to-STSSopenWindow-env-typetypeI-op
    to-STSSopenWindow) (signed-byte 32) 268509190)

(defalien from-STSSopenWindow from-STSSopenWindow 288 :static)

(defalien to-STSSfull0open to-STSSfull0open 2480 :static)

(alien-store (msg-msgsize-op (Msgsize-to-STSSfull0open-op
    to-STSSfull0open)) (signed-byte 32) 310)

(alien-store (msg-simplemsg-op (Msgsize-to-STSSfull0open-op
    to-STSSfull0open)) boolean NIL)

(alien-store (msg-msgtype-op (Msgsize-to-STSSfull0open-op
    to-STSSfull0open)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSSfull0open-op to-STSSfull0open))
    (signed-byte 32) 2802)
(alien-store (to-STSSfull0open-vp-typetypeI-op to-STSSfull0open)
    (signed-byte 32) 268509190)
(alien-store (to-STSSfull0open-env-typetypeI-op to-STSSfull0open)
    (signed-byte 32) 268509190)
(alien-store (to-STSSfull0open-fontName-typetypeI-op to-STSSfull0open)
    (signed-byte 32) 285214720)
(alien-store (to-STSSfull0open-doWrap-typetypeI-op to-STSSfull0open)
    (signed-byte 32) 268501248)
(alien-store (to-STSSfull0open-dispPages-typetypeI-op to-STSSfull0open)
    (signed-byte 32) 268505089)

(defalien from-STSSfull0open from-STSSfull0open 288 :static)

(defalien to-STSSfull0openWindow to-STSSfull0openWindow 2480
    :static)

(alien-store
  (msg-msgsize-op (Msgsize-to-STSSfull0openWindow-op
    to-STSSfull0openWindow)) (signed-byte 32) 310)

(alien-store
  (msg-simplemsg-op (Msgsize-to-STSSfull0openWindow-op
    to-STSSfull0openWindow)) boolean NIL)

(alien-store
  (msg-msgtype-op (Msgsize-to-STSSfull0openWindow-op
    to-STSSfull0openWindow)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSSfull0openWindow-op
```

```
        to-STSFULLOPENWINDOW)) (signed-byte 32) 2803)
(alien-store (to-STSFULLOPENWINDOW-w-typetypeI-op
    to-STSFULLOPENWINDOW) (signed-byte 32) 268509190)
(alien-store (to-STSFULLOPENWINDOW-env-typetypeI-op
    to-STSFULLOPENWINDOW) (signed-byte 32) 268509190)
(alien-store (to-STSFULLOPENWINDOW-fontName-typetypeI-op
    to-STSFULLOPENWINDOW) (signed-byte 32) 285214720)
(alien-store (to-STSFULLOPENWINDOW-dowrap-typetypeI-op
    to-STSFULLOPENWINDOW) (signed-byte 32) 268501248)
(alien-store (to-STSFULLOPENWINDOW-disppages-typetypeI-op
    to-STSFULLOPENWINDOW) (signed-byte 32) 268505089)

(defalien from-STSFULLOPENWINDOW from-STSFULLOPENWINDOW 288
:static)

(defalien to-STSFULLLINE to-STSFULLLINE 176 :static)

(alien-store (msg-msgsize-op (Msgsize-to-STSFULLLINE-op
    to-STSFULLLINE)) (signed-byte 32) 22)

(alien-store (msg-simplemsg-op (Msgsize-to-STSFULLLINE-op
    to-STSFULLLINE)) boolean t)

(alien-store (msg-msgtype-op (Msgsize-to-STSFULLLINE-op
    to-STSFULLLINE)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSFULLLINE-op
    to-STSFULLLINE)) (signed-byte 32) 2804)

(defalien from-STSFULLLINE from-STSFULLLINE 272 :static)

(defalien to-STSGETCHAR to-STSGETCHAR 176 :static)

(alien-store (msg-msgsize-op (Msgsize-to-STSGETCHAR-op
    to-STSGETCHAR)) (signed-byte 32) 22)

(alien-store (msg-simplemsg-op (Msgsize-to-STSGETCHAR-op
    to-STSGETCHAR)) boolean t)

(alien-store (msg-msgtype-op (Msgsize-to-STSGETCHAR-op
    to-STSGETCHAR)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSGETCHAR-op to-STSGETCHAR))
(signed-byte 32) 2805)

(defalien from-STSGETCHAR from-STSGETCHAR 272 :static)

(defalien to-STSGETSTRING to-STSGETSTRING 176 :static)

(alien-store (msg-msgsize-op (Msgsize-to-STSGETSTRING-op
    to-STSGETSTRING)) (signed-byte 32) 22)

(alien-store (msg-simplemsg-op (Msgsize-to-STSGETSTRING-op
```

```
to-STSGetString)) boolean t)

(alien-store (msg-msgtype-op (Msgsize-to-STSGetString-op
    to-STSGetString)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSGetString-op
    to-STSGetString)) (signed-byte 32) 2806)

(defalien from-STSGetString from-STSGetString 2304 :static)

(defalien to-STSPutChar to-STSPutChar 224 :static)

(alien-store (msg-msgsize-op (Msgsize-to-STSPutChar-op
    to-STSPutChar)) (signed-byte 32) 28)

(alien-store (msg-simplemsg-op (Msgsize-to-STSPutChar-op
    to-STSPutChar)) boolean t)

(alien-store (msg-msgtype-op (Msgsize-to-STSPutChar-op
    to-STSPutChar)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSPutChar-op to-STSPutChar))
    (signed-byte 32) 2807)
(alien-store (to-STSPutChar-ch-typetypeI-op to-STSPutChar)
    (signed-byte 32) 268503048)

(defalien to-STSPutString to-STSPutString 2256 :static)

(alien-store (msg-msgsize-op (Msgsize-to-STSPutString-op
    to-STSPutString)) (signed-byte 32) 282)

(alien-store (msg-simplemsg-op (Msgsize-to-STSPutString-op
    to-STSPutString)) boolean t)

(alien-store (msg-msgtype-op (Msgsize-to-STSPutString-op
    to-STSPutString)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSPutString-op
    to-STSPutString)) (signed-byte 32) 2808)
(alien-store (to-STSPutString-s-typetypeI-op to-STSPutString)
    (signed-byte 32) 285214720)

(defalien to-STSFlushInput to-STSFlushInput 176 :static)

(alien-store (msg-msgsize-op (Msgsize-to-STSFlushInput-op
    to-STSFlushInput)) (signed-byte 32) 22)

(alien-store (msg-simplemsg-op (Msgsize-to-STSFlushInput-op
    to-STSFlushInput)) boolean t)

(alien-store (msg-msgtype-op (Msgsize-to-STSFlushInput-op
    to-STSFlushInput)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSFlushInput-op
    to-STSFlushInput)) (signed-byte 32) 2809)
```

```
(defalien to-STFlushOutput to-STFlushOutput 176 :static)
(alien-store (msg-msgsize-op (Msgsize-to-STFlushOutput-op
    to-STFlushOutput)) (signed-byte 32) 22)

(alien-store (msg-simplemsg-op (Msgsize-to-STFlushOutput-op
    to-STFlushOutput)) boolean t)

(alien-store (msg-msgtype-op (Msgsize-to-STFlushOutput-op
    to-STFlushOutput)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STFlushOutput-op
    to-STFlushOutput)) (signed-byte 32) 2810)

(defalien from-STFlushOutput from-STFlushOutput 224 :static)

(defalien to-STChangeEnv to-STChangeEnv 240 :static)
(alien-store (msg-msgsize-op (Msgsize-to-STChangeEnv-op
    to-STChangeEnv)) (signed-byte 32) 30)

(alien-store (msg-simplemsg-op (Msgsize-to-STChangeEnv-op
    to-STChangeEnv)) boolean NIL)

(alien-store (msg-msgtype-op (Msgsize-to-STChangeEnv-op
    to-STChangeEnv)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STChangeEnv-op
    to-STChangeEnv)) (signed-byte 32) 2811)
(alien-store (to-STChangeEnv-env-typetypeI-op to-STChangeEnv)
    (signed-byte 32) 268509190)

(defalien to-STSGrabWindow to-STSGrabWindow 240 :static)
(alien-store (msg-msgsize-op (Msgsize-to-STSGrabWindow-op
    to-STSGrabWindow)) (signed-byte 32) 30)

(alien-store (msg-simplemsg-op (Msgsize-to-STSGrabWindow-op
    to-STSGrabWindow)) boolean NIL)

(alien-store (msg-msgtype-op (Msgsize-to-STSGrabWindow-op
    to-STSGrabWindow)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSGrabWindow-op
    to-STSGrabWindow)) (signed-byte 32) 2812)
(alien-store (to-STSGrabWindow-kPort-typetypeI-op
    to-STSGrabWindow) (signed-byte 32) 268509190)

(defalien from-STSGrabWindow from-STSGrabWindow 288 :static)

(defalien to-STSPutCharArray to-STSPutCharArray 400 :static)
(alien-store (msg-msgsize-op (Msgsize-to-STSPutCharArray-op
    to-STSPutCharArray)) (signed-byte 32) 50)
```

```
(alien-store (msg-simplemsg-op (Msgsize-to-STSPutCharArray-op
    to-STSPutCharArray)) boolean NIL)

(alien-store (msg-msgtype-op (Msgsize-to-STSPutCharArray-op
    to-STSPutCharArray)) (signed-byte 32) 0)

(alien-store (msg-id-op (Msgsize-to-STSPutCharArray-op
    to-STSPutCharArray)) (signed-byte 32) 2813)
(alien-store (to-STSPutCharArray-chars-typeTypeI-op
    to-STSPutCharArray) (signed-byte 32) 536870912)
NIL
(alien-store (to-STSPutCharArray-chars-TypeBitSize-Op
    to-STSPutCharArray) (signed-byte 16) 8)
(alien-store (to-STSPutCharArray-chars-typename-op
    to-STSPutCharArray) (signed-byte 16) 8)
(alien-store (to-STSPutCharArray-firstCh-typeTypeI-op
    to-STSPutCharArray) (signed-byte 32) 268505089)
(alien-store (to-STSPutCharArray-lastCh-typeTypeI-op
    to-STSPutCharArray) (signed-byte 32) 268505089)
```

9.3. Server-Supplied Code

Server supplied code does its own initialization. No Init routine is supplied for the server side. The server code has a top level control loop, as in the following example:

```
-Initialize-
loop: if not Serving then goto DoFrom;

        GR := Receive( InMsg );
        if GR = Success then TSServer( InMsg, OutMsg );
        unless [OutMsg.RetCode = NoReply] Send( OutMsg );

DoFrom: call a From msg matchmaker routine;
        goto loop;
```


PERQ C PROGRAMMING

December 7, 1984

Copyright © 1984 PERQ Systems Corporation
2600 Liberty Avenue
P. O. Box 2600
Pittsburgh, PA 15230
(412) 355-0900

Accent is a trademark of Carnegie-Mellon University.

Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its Spice Project.

This document is not to be reproduced in any form or transmitted in whole or in part without the prior written authorization of PERQ Systems Corporation.

The information in this document is subject to change without notice and should not be construed as a commitment by PERQ Systems Corporation. The company assumes no responsibility for any errors that may appear in this document.

PERQ Systems Corporation will make every effort to keep customers apprised of all documentation changes as quickly as possible. The Reader's Comments card is distributed with this document to request users' critical evaluation to assist us in preparing future documentation.

PERQ, PERQ2, LINQ, and Qnix are trademarks of PERQ Systems Corporation.

UNIX is a trademark of Bell Laboratories.

Table of Contents

Page

<u>1. Introduction</u>	<u>CP-1</u>
<u>2. Pre-Processing the File</u>	<u>CP-3</u>
<u>3. Compiling the File</u>	<u>CP-9</u>
<u>4. Assemblying the File</u>	<u>CP-11</u>
<u>5. Linking the File</u>	<u>CP-13</u>

1. Introduction

To write a program for the PERQ workstation using the C programming language, see *C, A Reference Manual* (Samuel P. Harbison and Guy L. Steele, Jr., Tartan Laboratories, 1984), available from PERQ Systems.

This document describes how to prepare a C program for use after it's written.

Preparing a C program for use on the PERQ workstation involves four separate steps:

1. Pre-processing
2. Compiling
3. Assembling
4. Linking

which are explained on the following pages.

2. Pre-Processing the File

The first step in using the C compiler is to pre-process your file. This is done with the CPP (C pre-processor) command. The syntax of this command is:

```
CPP [-C] ...[-P] ...[-R] ... [-Dname] ... [-Dname=def] ...
[-Idirectory] ...[-Uname] ... [<infile>] [<outfile>]
```

where:

-C directs the pre-processor to pass comments on to the next phase of the compiler. Normally, comments are ignored.

-P prevents *line #n from file filename* messages from being inserted into the output file.

-R allows recursive macros.

-Dname=value defines name the same way that a #define would inside the program. The value defaults to 1.

-Dname=def defines name as def. Note that the equal sign (=) must be single quoted ('=').

-Idirectory specifies that #include files should be searched for in directory path name before the standard search list is used.

-Uname undefines name.

<infile> the name of the input C file to be compiled.

and

<outfile> the name of the output (or intermediate) file.

If there are two non-flag arguments then the first is the name of the input file and the second is the name of the output file. If there is one non-flag argument, then it is the name of the input file and the output is written on the standard output (the screen). If there are no non-flag arguments, then the input is taken from the standard input and the output is written on the standard output.

The pre-processor pass allows compiler directives such as #include and #define files to be handled for compilation.

The following are some programming hints about using CPP:

Symbols defined on the command line by -Dfoo are defined as "1." It is as if they had been defined by #define foo 1 or -Dfoo=1.

The directory search order for #include files is as follows:

1. The directory of the file that contains the #include request (e.g. #include is relative to the file being scanned when the request is made).
2. The directories specified by -I, in left-to-right order.
3. The standard directory(s).

An unescaped linefeed (the single character "\n") terminates a character constant or quoted string. An escaped linefeed (the two-character sequence "\\n") can be used in the body of a #define statement to continue the definition onto the next line. To generate the character sequence "\\n", type /<cr>. The escaped linefeed is not included in the macro body.

Comments are uniformly removed (except if the argument -C is specified). They are also ignored, except that a comment terminates a token.

Macro formal parameters are recognized in #define bodies, even inside character constants and quoted strings. The output from the sequence:

```
#define foo(a) '/a'  
foo(bar)
```

is the seven characters "'\\bar'". However, macro names are *not* recognized inside character constants or quoted strings during the regular scan. Thus, the sequence:

```
#define foo bar  
printf("foo");
```

does not expand 'foo' in the second line, because it is inside a quoted string that is not part of a #define macro definition.

Macros are not expanded while processing a #define or #undef. Thus, the sequence:

```
#define foo star  
#define bar foo  
#undef foo  
bar
```

produces "foo." The token appearing immediately after a '#ifdef' or '#ifndef' is not expanded.

Macros are not expanded during the scan that determines the actual parameters to another macro call. Thus, the sequence:

```
#define foo(a,b)b a
```

```
#define bar hi
foo bar
#define bar bye
)
```

produces "bye" (and warns about the refeinition of 'bar').

Some things to keep in mind about the preprocessor:

Any kind and amount of white space (space, tab, linefeed, vertical tab, formfeed, or carriage return) is allowed between a macro name and the left parenthesis that introduces its actual parameters.

The comma (,) operator, macros with parameters, and single character character constants are all legal in preprocessor `#if` statements.

If the -R flag is not specified, then the invocation of some recursive macros is trapped and the recursion forcibly terminated with an error message. The recursions that are trapped are the ones in which the nesting level is non-decreasing from some point on. For example:

```
#define a a
a
```

will be detected. (Use "`#undef a`" if that is what you want.) However, the recursion:

```
#define a c b
#define b c a
#define c foo
a
```

will not be detected because the nesting level decreases after each

expansion of "c".

The -R flag specifically allows recursive macros and recursion will be strictly obeyed (to the extent that space is available). Assuming that -R is specified:

```
#define a a  
a
```

causes an infinite loop with very little output. The tail recursion:

```
#define a <b  
#define b >a
```

causes the string "<>" to be output infinitely many times. The non-tail recursion:

```
#define a b>  
#define b a<  
a
```

complains "too much pushstack", dumps the pushstack, and continues (again, infinitely).

Mismatch between the number of formals and actuals in a macro call produces only a warning, and not an error. Excess actuals are ignored; missing actuals are turned into null strings.

The backslash (/) in *a=/*b* is interpreted as the first character of the pair /* which introduces a comment, rather than as the second character of the divide-and-replace operator =/.

3. Compiling the File

The second step in the process is the compilation of your file. This is done with the PCC (PERQ C Compiler) command. The syntax of this command is:

PCC filename.i filename.s

where:

filename.i is the name of the output (or intermediate) file generated by the pre-processor pass in step 1.

and

filename.s is the name of the target file that is to contain the assembly code.

The filenames are optional. The compiler will use the default I/O files if no files are specified.

4. Assemblying the File

The next step in the process is to assemble your file. This is done with the ASM (Assembly) command. The syntax of this command is:

ASM [-O] ... [-D | -d] ... [-o] ... filename.o filename.s

where:

-O is the optimize switch.

-D turns off the output of assembler debugging information.
(Default)

-d turns on the output of assembler debugging information.

-o is the output switch.

filename.o is the value for the -o switch; they must appear together.

and

filename.s is the name of the input file that contains the assembly language code.

The filenames are optional. The compiler will use the default I/O files if no files are specified.

5. Linking the File

The fourth and last step in the process is to link your file with other required files. This is done with the LNK (Link) command. The syntax of this command is:

LNK -o runfilename libc.lib {x.o y.o z.o}

where:

-o is the output switch. It must be placed before the output filename.

runfilename is the name of the target file that is to contain the executable code; for example:

lnk -o runfile libc.lib {x.o ... z.o filename.o}

libc.lib is the name of the file containing all the required libraries. It should always be linked with the main module (although it does not have to come first in the command line).

filename.o is the name of the input file that contains the object code for linking.

Optional switches that may be used with the LNK command are:

{x.o y.o z.o}

are the names of any additional files you may wish to link with the main module.

-m indicates a library is being linked.

@indirectfile

replaces **{x.o ... z.o}** **filename.o**. Indicates a list of files is to be read from *indirectfile*. Each line of the file may contain the name of a file to load, another indirectfile command, or a comment.

Comments begin with a exclamation point (!) in column 1. Blank lines are ignored. Note that you may NOT put switches into indirect files.

-g mainsymbolname

indicates program is to start by calling the routine *mainsymbolname*. The default is *main*.

- x indicates that local symbols are not to be preserved in the output symbol table. This switch saves some space, but makes debugging harder.
- L saves all local symbols in the output symbol table, even those starting with *L* and ', which are compiler-generated temporary symbols.
- dfu turns on debugging output for fixups.
- dplc turns on debugging output for placement.
- dsym turns on debugging output for symbols.
- dif turns on debugging output for input files.
- dof turns on debugging output for output file.
- hash generates has table statistics.
- HASH runs long enough to test hash table; turns on -hash switch.
- dall turns on all debugging output.
- and
- help prints the help message.

Some examples of the LNK command:

Example 1.

```
lnk -o foo.exe libc.lib foo.o
```

links *foo.o* with reference to library *libc.lib*
into the file *foo.exe*

Example 2.

```
lnk -m -o libxx.exe —libfiles.lmd
```

links library *libxx.exe* from the files listed in
libfiles.lmd.

Example 3.

```
lnk -o bar.exe bar1.o bar2.o bar3.o libbar.exe
```

links *bar.exe* from *bar1.o*, *bar2.o*, *bar3.o*
and the library *libbar.exe*.

Example 4.

```
lnk -o bartest.exe bar.exe bartest.o
```

links *bartest.exe* from *bartest.o* and the previously
linked program *bar.exe*.

Following are some programming hints about using LNK:

Notice that there is no "library switch." This is because LNK determines if a file is a library by looking at the library's contents.

Putting libraries in the front of the command line will make things go a little faster.

If a symbol is defined in both a library and a *.o* file, the *.o* file definition takes precedence. This means that you can have your own version of a routine that is also in a library. However, any references to that routine by other routines in the library will be to the routine in the library, NOT to your new routine. Note, too, that regardless of the order of libraries and *.o* files in the command line, symbols from *.o* files take precedence.

The output file from a LNK run can always be used as a library. You will get a message about *main* being redefined if you do use the output file as a library, but the message is informational only. It will not affect the LNK.

INDEX

KEY TO ABBREVIATIONS - LANGUAGE MANUAL

Volume 1

PE - PERQ/Accent Pascal Extensions

PL - Pascal Library

PM - PasMac: A Pascal Macro-Processor

Volume 2

MD - Module Index

KR - Kraut: PERQ Pascal Remote
Symbolic Debugger

MM - Matchmaker: The Accent Remote Procedure
Call Language

CS - C System Interfaces
(index included with document)

CP - PERQ C Programming

\$ command KR-32
& command KR-33
'S'! command KR-24
'S'? command KR-23
-- command KR-33
:= command KR-28
<CR> command KR-28
= command KR-27
@ command KR-32

Abbreviations KR-19
Act counter KR-14
Action command KR-25
Address command KR-33
AddrToBlkNum PL-85
AddSearchWord PL-63
Adjust PL-167
AllocCommandNode PL-59
ALoadError PL-24
AppendChar PL-167
AppendString PL-168
ArcCos PL-143
ArcSin PL-142
Arctan PL-143
ArcTan2 PL-143
Around command KR-23
ARunLoad PL-23
ASM CP-11
Assembling a C program CP-11

Backward search command KR-23
Bargraph KR-38
Basic path expressions KR-13
Bind command KR-39
BlkNumToAddr PL-86
BottomStack command KR-29
Break command KR-23
Breakpoint commands KR-22
Bugs KR-1

C assembler CP-11
C compiler CP-9
C linker CP-13
C pre-processor CP-3
C programming language CP-1
C programs
 #define files CP-4
 #include files CP-4
 Assembly language code CP-11
 Executable code CP-13
 Required libraries CP-13

C-specific information
 Matchmaker MM-45
Calls command KR-29
 Cat3 PL-168
 Cat4 PL-168
 Cat5 PL-169
 Cat6 PL-169
 CF_IOBoard PL-73
 Cf_Monitor PL-73
 Cf_Network PL-74
 Cf_OldZ80 PL-74
 ChangeUserParams PL-29
 Changing directory KR-20
 ChDir command KR-20
 Check expressions KR-15
 CheckHeap PL-91
 CheckPath command KR-25
 CheckUser PL-28
 CLoadProcess PL-39
 Close command KR-21
 Command language KR-19
 Command sequence KR-19
 CommandLanguage KR-1
 Comment KR-33
 Compiling a C program CP-9
 Compute command KR-33
 ComputeProgress PL-208
 Concat PL-170
 ConfirmUser PL-28
 ConvertPoolToString PL-65
 ConvertStringToPool PL-65
 ConvUpper PL-170
 Coping KR-1
 Coping with debugger bugs KR-49
 CoreRunLoad PL-79
 Cos PL-141
 CosH PL-144
 CoTan PL-142
 Counter KR-14
 Counter command KR-24
 CPP CP-3

Create window command KR-37
CreateHeap PL-89
Current command KR-33
Current restrictions KR-51
CVD PL-170
CVH PL-171
CVHS PL-171
CVHSS PL-172
CvInt PL-172
CvL PL-173
CVLS PL-174
CVN PL-173
CVO PL-174
CVOS PL-175
CVOSS PL-175
CVS PL-176
CVSS PL-176
CvUp PL-177

DateString PL-24
DBind command KR-40
DCounter command KR-24
Debugger commands
 \$ command KR-32
 & command KR-33
 'S'? command KR-23
 -- command KR-33
 := command KR-28
 <CR> command KR-28
 = command KR-27
 command KR-32
 'S'! command KR-24
Action command KR-25
Address command KR-33
Around command KR-23
Backward search command KR-23
Bind command KR-39
BottomStack command KR-29
Break command KR-23
Calls command KR-29
ChDir command KR-20

CheckPath command KR-25
Close command KR-21
Command language KR-19
Command sequence KR-19
Compute command KR-33
Counter command KR-24
Create window command KR-37
Current command KR-33
DBind command KR-40
DCounter command KR-24
Define Function command KR-24
Delete window command KR-37
Disable command KR-25
DModel command KR-40
DownStack command KR-29
DTrace command KR-22
EditPathrule command KR-26
Enable command KR-26
EXCEPTION command KR-33
Execute command KR-30
FindPath command KR-26
Forward search command KR-24
Go command KR-32
Grep command KR-24
Halt command KR-32
Help command KR-34
Indent command KR-34
Locals command KR-27
Mace command KR-34
Maintainer command KR-34
Model command KR-38
News command KR-34
Next command KR-28
Open command KR-20
Parameters command KR-27
PathRules command KR-27
Quit command KR-34
Radix command KR-27
Remove Action command KR-27
Remove command KR-27
Report command KR-34

Run command KR-32
Script command KR-35
Scroll command KR-23
SetSearch command KR-21
Shell command KR-35
Show command KR-36
Show modules command KR-21
Silence command KR-35
System command KR-21
TerminateTarget command KR-32
TopStack command KR-29
Trace command KR-21
Type command KR-23
Unwind command KR-32
UpStack command KR-29
Version command KR-37
Define Function command KR-24
Definition commands KR-24
Delete window command KR-37
DeleteChars PL-177
DeleteSearchWord PL-64
DestroyChPool PL-66
DestroyCommandList PL-59
DestroyCommandParse PL-58
DestroyHeap PL-90
DestroySearchTree PL-64
Disable command KR-25
DisablePrvs PL-122
DisposeP PL-90
DModel command KR-40
DownStack command KR-29
DstryCmdFiles PL-58
DTrace command KR-22

Editor commands KR-23
EditPathrule command KR-26
Enable command KR-26
EnablePrvs PL-123
ErrorMsgPMBroadcast PL-151
EXCEPTION command KR-33
Exec PL-159

Execute command KR-30
ExerciseParseEngine PL-60
ExitAllCmdFiles PL-57
ExitCmdFile PL-57
Exp PL-140
Exported definitions MD-7
ExtractEvent PL-107

Find expressions KR-15
FindPath command KR-26
FinishDiskUtils PL-87
Forward search command KR-24
FullLn PL-194

Generalized path expressions KR-13
GetB PL-191
GetBreak PL-178
GetC PL-191
GetCharacterPool PL-72
GetCmd PL-68
GetConfirm PL-71
GetDiskInfo PL-87
GetEventPort PL-107
GetIthWordPtr PL-66
GetParsedUserInput PL-52, PL-70
GetShellCmd PL-69
Getting Started KR-1, KR-2
GetUserName PL-30
Global commands KR-27
Go command KR-32
Grep command KR-24
GRErrorMsg PL-151
GRStdErr PL-152
GRStdError PL-150
GRWriteErrorMsg PL-150
GRWriteStdError PL-149

Halt command KR-32
Help command KR-34

Indent command KR-34

InitAuth PL-31
InitCmdFile PL-56
InitCommandParse PL-58
InitDiskUtils PL-87
InitDynamic PL-89
InitExceptions PL-96
Initial PL-178
InitPascal PL-122
InitProcess PL-122
InitStream PL-193
InitWordSearchTable PL-63
InsertChars PL-178
Introduction KR-1
IOGetTime PL-41
IsPattern PL-131
IsStreamDevice PL-195

JumpControlStore PL-77

KBFlushBoardOutput PL-193
Known bugs KR-51
Kraut commands
 See also Debugger commands

Linking C programs CP-13
LinkTypeStr PL-24
Lisp-specific information
 Matchmaker MM-46
ListLoggedInUsers PL-30
Ln PL-140
LINK CP-13
LoadControlStore PL-76
LoadMicroInstruction PL-76
Locals command KR-27
Log10 PL-140
LoginUser PL-27
LogoutUser PL-27
Lop PL-179

Mace KR-1, KR-41
Mace command KR-34

MaceCommands KR-44
Macro pre-processor PM-1
Maintainer command KR-34
Manipulating path rules KR-17
MapAddr PL-86
Match KR-16
Matchmaker
 BNF MM-39
 C-specific information MM-45
 Call arguments MM-22
 Call options MM-27
 Command line syntax MM-36
 Data types MM-9
 Example MM-49
 Expression operators MM-7
 Expressions MM-7
 File names MM-35
 Goals MM-3
 Interface specifications MM-6
 Kinds of specifications MM-5
 Lexical structure MM-5
 Lisp-specific information MM-46
 Message IDs MM-30
 Order of declarations MM-6
 Pascal-specific information MM-45
 Substitute-typedef MM-13, MM-47
 Types of calls MM-19
 Types specifications MM-6
 Using server code MM-34
 Using user code MM-33
 See also Matchmaker calls, Matchmaker data types,
 Matchmaker e
Matchmaker calls MM-19
 Alternate_Reply MM-22
 Call options MM-27
 Data arguments MM-23
 Default option values MM-28
 Message MM-20
 Message IDs MM-30
 Next_ID MM-30
 Non-call argument options MM-28

Options declarations MM-27
Remote_Procedure MM-19
Server_Message MM-21
Skip_ID MM-30
Special arguments MM-26
 See also Special arguments
Matchmaker data declarations
 Built-in types MM-10
 Use declarations MM-11
Matchmaker data types MM-9, MM-23
 Arrays MM-14
 Constant declarations MM-9
 Enumeration types MM-15
 Packed arrays MM-15
 Packed records MM-13
 Pointers MM-17
 Records MM-13
 Type declarations MM-9
 Union types MM-17, MM-25
 Variable-sized arrays MM-14, MM-24
Matchmaker example MM-49
 Initialization code MM-127
 TS.mm-pas MM-51
 TSDefs.pas MM-51
 TSDefs.slisp MM-91
 TSMsgDefs.slisp MM-108
 TSServer.pas MM-74
 TSUInit.slisp MM-122
 TSUser.pas MM-51
 TSUser.slisp MM-92
 Typescript.mm MM-49
Matchmaker expressions
 Boolean literals MM-7
 Constant names MM-7
 Enumeration names MM-7
 Literals MM-7
 Operators MM-7
Matchmaker lexical structure MM-5
 Characters MM-5
 Comments MM-5
 Identifiers MM-5

Integers MM-5
Strings MM-5
Mode qualifier KR-41
Model command KR-38
Model commands KR-38
MultiLevelProgress PL-209
MultiStreamProgress PL-209

Naming KR-1, KR-7
NewP PL-90
News command KR-34
NewToOldTime PL-119
Next command KR-28
NextCh PL-133
NoMatch KR-16
Null_CommandBlock PL-55

OldCurrentTime PL-119
OldToNewTime PL-120
Open command KR-20
OpenCmdFile PL-56

Pad PL-179
Parameters command KR-27
ParseChPool PL-61
ParseCommand PL-61
Pascal
 Activations PE-6
 ALL PE-20
 ALL EXCEPTION PE-20
 AND PE-35
 Arctan PE-31
 Arithmetic Functions PE-31
 Array-Types PE-9
 Assignment-Compatibility PE-14
 AUTO PE-57
 Automatic RESET/REWRITE PE-57
 Block PE-5
 Blocks, Scope, and Activations PE-5
 Boolean Functions PE-33
 Buffer-Variables PE-17

CASE Statements PE-42
CHR PE-33
CLOSE PE-23
Command Line PE-53
COMMENT Switch PE-60
Compatible Types PE-14
Compiler Switches PE-54
Component-Variables PE-17
Conditional Compilation PE-61
Constant Expressions PE-7
Constant-Definitions PE-7
Control Structures PE-39
Cos PE-31
Declaration Relaxation PE-5
Declarations and Denotations of Variables PE-17
DISPOSE PE-25
Dynamic Allocation Procedures PE-23
Dynamic Space Allocation and Deallocation PE-23
ELSEC PE-61
ENDC PE-61
Entire-Variables PE-17
Error notification file PE-62
Errorfile Switch PE-62
EXCEPTION PE-19
Exception-Declarations PE-19
Exclusive Or PE-35
EXIT PE-39
Exp PE-31
EXPORTS Declarations PE-50
Expressions PE-37
Field-Designators PE-17
FILE PE-11
File Handling Procedures PE-22
File Inclusion PE-54
File-Types PE-11
FLOAT PE-32
Floating Point Numbers PE-4
Function Result Type PE-19
Function-Declarations PE-19
Functions PE-19
General Type-Definitions PE-9

Generic Files PE-11
Generic Pointers PE-13
Global INPUT/OUTPUT Switch PE-57
HANDLER PE-21
Handler-Declarations PE-21
Help switch PE-63
Identified-Variables PE-17
Identifiers PE-3
IFC PE-61
IMPORTS Declarations PE-50
INCLUDE PE-54
Inclusive Or PE-35
Indexed-Variables PE-17
InLineAWord PE-29
InLineByte PE-29
InLineWord PE-29
Input and Output PE-45
Input/Output Intrinsics PE-22
INTEGER PE-3
INTEGER Logical Operations PE-34
LAND PE-35
LENGTH PE-34
Lexical Alternatives PE-4
Lexical Tokens PE-3
LIST PE-55
Ln PE-31
LNOT PE-35
LoadAdr PE-30
LoadExpr PE-30
Logical Operations PE-34
LONG PE-3
LONG Logical Operations PE-34
LOR PE-35
LXOR PE-35
MakePtr PE-28
MakeVRD PE-28
MESSAGE PE-60
Mixed-Mode Expressions PE-37
Mixed-Mode Permission Switch PE-58
Modules PE-49
NEW PE-24

NoPageFault PE-30
NOT PE-35
Numbers PE-3
Octal Whole Number Constants PE-3
ODD PE-33
OR PE-35
ORD PE-33
Ordinal Functions PE-33
OTHERWISE PE-42
Pack PE-25
Parameter Lists PE-22
Parameters PE-22
PERQ-Specific Functions PE-34
PERQ-Specific Procedures PE-25
POINTER PE-13
Pointer-Types PE-13
Pre-processor PM-1
PRED PE-33
PRIVATE PE-50
Procedure-Declarations PE-19
Procedure / Function Names PE-58
Procedures PE-19
Programs and Modules PE-49
QUIET PE-56
Quirks PE-65
RAISE PE-40
RANGE PE-56
Range Checking PE-56
RasterOp PE-26
READ PE-45
READLN PE-45
REAL PE-4
RECAST PE-32
Record Comparisons PE-38
Record-Types PE-11
References PE-69
Required Functions PE-31
Required Procedures PE-22
Required Simple-Types PE-9, iv
RESET PE-23
REWRITE PE-22

ROTATE PE-36
ROUND PE-31
Scope PE-6
Scrounge Switch PE-59
Set-Types PE-11
Sets PE-11
SHIFT PE-35
SHRINK PE-32
Simple-Statements PE-39
Simple-Types PE-9
Sin PE-31
Single and Double Precision Constants PE-3
Sqrt PE-31
StartIO PE-25
Statements PE-39
StorExpr PE-30
STRETCH PE-32
STRING PE-9
String-Types PE-9
Structured-Statements PE-42
Structured-Types PE-9
Subrange-Types PE-9
SUCC PE-33
Switches PE-53
Procedure Page PE-47
Procedure Read PE-45
Procedure Readln PE-45
Procedure Write PE-46
Procedure Writeln PE-46
Transfer Functions PE-31
Transfer Procedures PE-25
TRUNC PE-31
Type Coercion PE-32
Type Coercion Intrinsics PE-31
Type Compatibility PE-14
Type-Definitions PE-9
Unpack PE-25
Variable-Declarations PE-17
VERSION PE-59
Whole Numbers PE-3
WordSize PE-34

WRITE PE-46
WRITELN PE-46
Pascal library MD-3
 Exported definitions MD-7
Pascal-specific information
 Matchmaker MM-45
PasMac PM-1
 Case sensitivity PM-20
 Evaluation PM-5
 Example PM-21, PM-22
 Expansion PM-5
 Invoking PM-7
 Macro calls PM-13
 Macro definitions PM-11
 Macro variables PM-16
 MEQU operator PM-18
 MEXP statements PM-18
 MIF statements PM-18
 MINCLUDE statements PM-18
 MLS operator PM-18
 Pascal comments PM-19
 Semantics PM-11
 Syntax PM-9
Path actions KR-16
Path function KR-13
Path rule commands KR-25
Path rules KR-13
Path variable KR-15
PathRules KR-1
PathRules command KR-27
PattCheck PL-134
PattDebug PL-131
PattMap PL-132
PattMatch PL-132
PCC CP-9
PosC PL-180
PosString PL-180
Power PL-140
PowerI PL-141
Pre-processing a C program CP-3
PReadIn PL-192

Precedence of operators KR-42
Predicate KR-14
Primary value KR-42
Procedure AlwaysEof PL-59
Process Control Characters KR-30
Public modules MD-3
PutB PL-192
PutC PL-192
PWriteIn PL-193

Quit command KR-34
QuitMultiProgress PL-210
QuitProgress PL-209

Radix command KR-27
Radix qualifier KR-41
RaiseP PL-96
RandomProgress PL-209
ReadD PL-125
ReadR PL-127
RecRecord PL-104
Redirection of output KR-38
Remove Action command KR-27
Remove command KR-27
RemoveWindowAttentionFlag PL-208
RemoveWindowErrorFlag PL-207
RemoveWindowRequestFlag PL-208
ReplaceChars PL-181
Report command KR-34
ResetHeap PL-89
RevPosC PL-181
RevPosString PL-182
Routines
 PL-210
 AddrToBlkNum PL-85
 AddSearchWord PL-63
 Adjust PL-167
 AllocCommandNode PL-59
 ALoadError PL-24
 AlwaysEof PL-59
 AppendChar PL-167

AppendString PL-168
ArcCos PL-143
ArcSin PL-142
Arctan PL-143
ArcTan2 PL-143
ARunLoad PL-23
BlkNumToAddr PL-86
Cat3 PL-168
Cat4 PL-168
Cat5 PL-169
Cat6 PL-169
CF_IOBoard PL-73
Cf_Monitor PL-73
Cf_Network PL-74
Cf_OldZ80 PL-74
ChangeUserParams PL-29
CheckHeap PL-91
CheckUser PL-28
CLoadProcess PL-39
ComputeProgress PL-208
Concat PL-170
ConfirmUser PL-28
ConvertPoolToString PL-65
ConvertStringToPool PL-65
ConvUpper PL-170
CoreRunLoad PL-79
Cos PL-141
CosH PL-144
CoTan PL-142
CreateHeap PL-89
CVD PL-170
CVH PL-171
CVHS PL-171
CVHSS PL-172
CvInt PL-172
CvL PL-173
CVLS PL-174
CVN PL-173
CVO PL-174
CVOS PL-175
CVOSS PL-175

CVS PL-176
CVSS PL-176
CvUp PL-177
DateString PL-24
DeleteChars PL-177
DeleteSearchWord PL-64
DestroyChPool PL-66
DestroyCommandList PL-59
DestroyCommandParse PL-58
DestroyHeap PL-90
DestroySearchTable PL-64
DisablePrvs PL-122
DisposeP PL-90
DstryCmdFiles PL-58
EnablePrvs PL-123
ErrorMsgPMBroadcast PL-151
Exec PL-159
ExerciseParseEngine PL-60
ExitAllCmdFiles PL-57
ExitCmdFile PL-57
Exp PL-140
ExtractEvent PL-107
FinishDiskUtils PL-87
FullLn PL-194
GetB PL-191
GetBreak PL-178
GetC PL-191
GetCharacterPool PL-72
GetCmd PL-68
GetConfirm PL-71
GetDiskInfo PL-87
GetEventPort PL-107
GetIthWordPtr PL-66
GetParsedUserInput PL-52, PL-70
GetShellCmd PL-69
GetUserName PL-30

GRErrorMsg PL-151
GRStdErr PL-152
GRStdError PL-150
GRWriteErrorMsg PL-150
GRWriteStdError PL-149
InitAuth PL-31
InitCmdFile PL-56
InitCommandParse PL-58
InitDiskUtils PL-87
InitDynamic PL-89
InitExceptions PL-96
Initial PL-178
InitPascal PL-122
InitProcess PL-122
InitStream PL-193
InitWordSearchTable PL-63
InsertChars PL-178
IOGetTime PL-41
IsPattern PL-131
IsStreamDevice PL-195
JumpControlStore PL-77
KBFlushBoardOutput PL-193
LinkTypeStr PL-24
ListLoggedInUsers PL-30
Ln PL-140
LoadControlStore PL-76
LoadMicroInstruction PL-76
Log10 PL-140
LoginUser PL-27
LogoutUser PL-27
Lop PL-179
MapAddr PL-86
MultiLevelProgress PL-209
MultiStreamProgress PL-209
NewP PL-90

PattMap PL-132
PattMatch PL-132
PosC PL-180
PosString PL-180
Power PL-140
Powerl PL-141
PReadIn PL-192
PutB PL-192
PutC PL-192
PWriteIn PL-193
QuitMultiProgress PL-210
QuitProgress PL-209
RaiseP PL-96
RandomProgress PL-209
ReadD PL-125
ReadR PL-127
RecRecord PL-104
RemoveWindowAttentionFlag PL-208
RemoveWindowErrorFlag PL-207
RemoveWindowRequestFlag PL-208
ReplaceChars PL-181
ResetHeap PL-89
RevPosC PL-181
RevPosString PL-182
Scan PL-182
SendRecord PL-103
SetBreak PL-183
ShowBreak PL-184
ShowPathAndTitle PL-207
ShowRun PL-23
ShowWindowAttentionFlag PL-208
ShowWindowErrorFlag PL-207
ShowWindowRequestFlag PL-207
Sin PL-141
SinH PL-144
Spawn PL-160
Split PL-160
Sqrt PL-139
Squeeze PL-184
Str PL-185
StreamClose PL-190

StreamFlushOutput PL-195
StreamInit PL-189
StreamKeyBoardReset PL-193
StreamName PL-194
StreamOpen PL-190
StreamProgress PL-208
Strip PL-185
SubStrFor PL-186
SubStrTo PL-186
Tan PL-142
TanH PL-144
Trim PL-187
ULInitial PL-187
ULPosString PL-187
UniqueWordIndex PL-64
UnMapAddr PL-86
UpCh PL-133
UpChar PL-188
UpEQU PL-188
WordifyPool PL-62
WriteChars PL-195
WriteD PL-126
WriteNChars PL-194
WriteR PL-128
Run command KR-32
Runtime stack commands KR-29

Scan PL-182
Script command KR-35
Scroll command KR-23
Search list commands KR-20
SendRecord PL-103
SetBreak PL-183
SetSearch command KR-21
Shell command KR-35
Show command KR-36
Show modules command KR-21
ShowBreak PL-184
ShowPathAndTitle PL-207
ShowRun PL-23
ShowWindowAttentionFlag PL-208

ShowWindowErrorFlag PL-207
ShowWindowRequestFlag PL-207
Silence command KR-35
Sin PL-141
SinH PL-144
Spawn PL-160
Special arguments MM-26
 Defaults MM-27
 LocalPort MM-26
 MsgType MM-26, MM-28
 Receive_Timeout MM-27, MM-28
 RemotePort MM-26
 ReplyType MM-26, MM-28
 Send_Option MM-26, MM-28
 Send_Timeout MM-26
Split PL-160
Sqrt PL-139
Squeeze PL-184
Str PL-185
StreamClose PL-190
StreamFlushOutput PL-195
StreamInit PL-189
StreamKeyBoardReset PL-193
StreamName PL-194
StreamOpen PL-190
StreamProgress PL-208
Strip PL-185
Substitute-typedef MM-13, MM-47
SubStrFor PL-186
SubStrTo PL-186
System command KR-21

Tan PL-142
TanH PL-144
Target process control KR-30
Term counter KR-14
TerminateTarget command KR-32
TopStack command KR-29
Trace command KR-21
Trace commands KR-21
Transcript of Debugging session KR-4

Trim PL-187
Type command KR-23

ULInitial PL-187
ULPosString PL-187
UniqueWordIndex PL-64
UnMapAddr PL-86
Unwind command KR-32
UpCh PL-133
UpChar PL-188
UpEQU PL-188
UpStack command KR-29

Variable inspection commands KR-27
Version command KR-37

Window commands KR-37
WordifyPool PL-62
WriteChars PL-195
WriteD PL-126
WriteNChars PL-194
WriteR PL-128