SECTION Translator Debugging		
Immediate	=%s	
Debug_Logicals	FALSE	
Debug_Inherit	FALSE	
Debug_Include	FALSE	
Debug_Dictionary	FALSE	
Debug_Lines	FALSE	
Debug_Tokens	FALSE	
Debug_Cdd	FALSE	
KillAlarm	0	Kill Translator after N seconds 0 = dont

SECTION VMS System Files		
LogicalsPath	sys$library:/opt/vxrt/pascal	
SystemInherit	pascal$cli_routines	
SystemInherit	pascal$d_float	
SystemInherit	pascal$g_float	
SystemInherit	pascal$lbr_routines	
SystemInherit	pascal$lbr_routines	
SystemInherit	pascal$lib_routines	
SystemInherit	pascal$smg_routines	
SystemInherit	pascal$str_routines	
SystemInherit	passtatus	
SystemInherit	starlet	
SystemInherit	decw$dwtmsg	
SystemInherit	decw$motif	
SystemInherit	decw$xlibdef	
SystemInherit	decw$xlibsmg	
SystemInherit	pascal$starlet	
SystemInherit	sqlliterals	
SystemInherit	sql_literals	

INCLUDE AND INHERIT		
StarLet	starlet_mod.hxx	
IncludeFrom		Names of modules which need special
IncludeFrom		#include file names. Formats:
IncludeFrom		IncludeFrom foo bar.h => #include <bar.h>
IncludeFrom		if QuoteIncludes=0 else #include 'bar.h'
IncludeFrom		IncludeFrom foo <bar.h> => #include <bar.h>
IncludeFrom		IncludeFrom foo 'bar.h' => #include 'bar.h'
IncludeFrom		IncludeFrom foo => no #include at all
IncludeFrom		iodeclarations <vxpas/iodecl.h>
		IncludeFrom system
		IncludeFrom printer
		IncludeFrom dos
		IncludeFrom crt
		ImportFrom Names of modules whose import text
		ImportFrom resides in the specified files.
		ImportFrom E.g.: ImportFrom mymod /usr/me/mymod.imp
		ImportFrom These are the Turbo Pascal standard units
ImportDir	%_s.p	Search list of other places to look for
ImportDir	%_s.pas	the module named %s.
ImportDir	%_s	
ImportDir	%H/%_s.imp	
IncludeDir	%s	
IncludeDir	%_s	Search list of places to look for the
IncludeDir	%_s	
IncludeDir	%_s	
IncludeDir	%^s	
IncludeDir	%Rs	
IncludeDir	%R_s.txt	
IncludeDir	%R_s.rpa	

IncludeDir	%R_s.pin	
IncludeDir	%R_s.pen	
IncludeDir	%R_s.pas	
IncludeDir	%R_s.pas	
IncludeDir	%R_s.inc	
IncludeDir	%R_s.def	
IncludeDir	%R_s.TXT	
IncludeDir	%R_s.RPA	
IncludeDir	%R_s.PAS	
IncludeDir	%R_s.PAS	
IncludeDir	%R_s.INC	
IncludeDir	%R^s.PIN	
IncludeDir	%R^s.PEN	
IncludeDir	%R^s.DEF	
InheritDir	%s	
InheritDir	%_s	
InheritDir	%^s	
InheritDir	%Rs	
InheritDir	%R_s.txt	
InheritDir	%R_s.rpa	
InheritDir	%R_s.pin	
InheritDir	%R_s.pen	
InheritDir	%R_s.pas	
InheritDir	%R_s.inc	
InheritDir	%R_s.def	
InheritDir	%R_s.TXT	
InheritDir	%R_s.RPA	
InheritDir	%R_s.PAS	
InheritDir	%R_s.INC	
InheritDir	%R^s.PIN	
InheritDir	%R^s.PEN	
InheritDir	%R^s.DEF	

OUTPUT & INPUT FILERS		
CodeFileName	%^Rs.cpp	Name of .c output file for a program,
		%s=input file name.
ModuleFileName	%^S.cpp	Name of .c output file for a module,
		%s=input file name, %S=module name.
		HeaderFileName %^S.h Name of .h output file for a module,
		HeaderFileName %/R_S.h Name of .h output file for a module,
HeaderFileName	%/R_S.hxx	Name of .h output file for a module,
		%s=input file name, %S=module name.
HeaderFileName2		If defined, different format to use when
		generating #include's, otherwise same.
		May be quoted as in IncludeFrom.
SelfIncludeName		Format to apply after HeaderFileName to use
		when a module includes its own header file.
		LogFileName %/R^s.log Name of log file name for -V mode.
LogFileName	%R^s.log	Name of log file name for -V mode.
		%s=input file name, %S=output file name.
		IncludeFileName %/R^s.h Format for translating Pascal include-file
		IncludeFileName %/R_s.h FFormat for translating Pascal include-file names.
IncludeFileName	%/R_s_inc.hxx	Format for translating Pascal include-file names.
		Controls format of NNNN in include NNNN
		see also quoteincludes
		IncludeOutFileName %Rs.h Name of .c output file for an INCLUDE
		IncludeOutFileName %R^s.h Name of .c output file for an INCLUDE
		IncludeOutFileName %R_s.h Name of .c output file for an INCLUDE
IncludeOutFileName	%R_s_inc.hxx	Name of .c output file for an INCLUDE

		file generated in ExpandIncludes=0 mode.
		InheritFileName %/R_s.mod Format for translating VMS PASCAL INHERIT names.
InheritFileName	%/R_s_mod.hxx	Format for translating VMS PASCAL INHERIT names.
		Controls format of NNNN in include NNNN
		see also quoteincludes
		InheritOutFileName %R^s.h Name of .c output file for an INHERIT
		InheritOutFileName %R_s.mod Name of .c output file for an INHERIT
InheritOutFileName	%R_s_mod.hxx	Name of .c output file for an INHERIT
		file generated in ExpandIncludes=0 mode.
		this is not used...
		InheritOutHeaderName %/R^s.h Name of .h output file for an INHERIT
		InheritOutHeaderName %/R_s.h Name of .h output file for an INHERIT

SECTION Pre Load inherit files		
PreLoad	starlet.pen	
PreLoad	forms$pas_definitions.pen	
PreLoad	pascal$c_routines.pen	
PreLoad	pascal$cli_routines.pen	
PreLoad	pascal$d_float.pen	
PreLoad	pascal$g_float.pen	
PreLoad	pascal$lbr_routines.pen	
PreLoad	pascal$lib_routines.pen	
PreLoad	pascal$mth_routines.pen	
PreLoad	pascal$ots_routines.pen	

PreLoad	pascal$smg_routines.pen	
PreLoad	pascal$str_routines.pen	
PreLoad	sql_literals.pas	
PreLoad	tcpip$inetdef.pen	

NameOf	int = __int	Format: NameOf name = newname
NameOf	new = __new	Rename the specified symbol. The name may
NameOf	delete = __delete	be of the form 'modulename.name' or
NameOf		'procname.name'; otherwise, all usages of
NameOf		the symbol are renamed.

ABSTRACT VMS ERROR CODES		
Nameof	EPERM:VMS_E_PERM	
Nameof	ENOENT:VMS_E_NOENT	
Nameof	ESRCH:VMS_E_SRCH	
Nameof	EINTR:VMS_E_INTR	
Nameof	EIO:VMS_E_IO	
Nameof	ENXIO:VMS_E_NXIO	
Nameof	E2BIG:VMS_E_2BIG	
Nameof	ENOEXEC:VMS_E_NOEXEC	
Nameof	EBADF:VMS_E_BADF	
Nameof	ECHILD:VMS_E_CHILD	
Nameof	EAGAIN:VMS_E_AGAIN	
Nameof	ENOMEM:VMS_E_NOMEM	
Nameof	EACCES:VMS_E_ACCES	
Nameof	EFAULT:VMS_E_FAULT	
Nameof	ENOTBLK:VMS_E_NOTBLK	
Nameof	EBUSY:VMS_E_BUSY	

Nameof	EEXIST:VMS_E_EXIST	
Nameof	EXDEV:VMS_E_XDEV	
Nameof	ENODEV:VMS_E_NODEV	
Nameof	ENOTDIR:VMS_E_NOTDIR	
Nameof	EISDIR:VMS_E_ISDIR	
Nameof	EINVAL:VMS_E_INVAL	
Nameof	ENFILE:VMS_E_NFILE	
Nameof	EMFILE:VMS_E_MFILE	
Nameof	ENOTTY:VMS_E_NOTTY	
Nameof	ETXTBSY:VMS_E_TXTBSY	
Nameof	EFBIG:VMS_E_FBIG	
Nameof	ENOSPC:VMS_E_NOSPC	
Nameof	ESPIPE:VMS_E_SPIPE	
Nameof	EROFS:VMS_E_ROFS	
Nameof	EMLINK:VMS_E_MLINK	
Nameof	EPIPE:VMS_E_PIPE	
Nameof	EDOM:VMS_E_DOM	
Nameof	ERANGE:VMS_E_RANGE	
Nameof	EWOULDBLOCK:VMS_E_WOULDBLOCK	
Nameof	EINPROGRESS:VMS_E_INPROGRESS	
Nameof	EALREADY:VMS_E_ALREADY	
Nameof	ENOTSOCK:VMS_E_NOTSOCK	
Nameof	EDESTADDRREQ:VMS_E_DESTADDRREQ	
Nameof	EMSGSIZE:VMS_E_MSGSIZE	
Nameof	EPROTOTYPE:VMS_E_PROTOTYPE	
Nameof	ENOPROTOOPT:VMS_E_NOPROTOOPT	
Nameof	EPROTONOSUPPORT:VMS_E_PROTONOSUPPORT	
Nameof	ESOCKTNOSUPPORT:VMS_E_SOCKTNOSUPPORT	
Nameof	EOPNOTSUPP:VMS_E_OPNOTSUPP	
Nameof	EPFNOSUPPORT:VMS_E_PFNOSUPPORT	
Nameof	EAFNOSUPPORT:VMS_E_AFNOSUPPORT	
Nameof	EADDRINUSE:VMS_E_ADDRINUSE	
Nameof	EADDRNOTAVAIL:VMS_E_ADDRNOTAVAIL	
Nameof	ENETDOWN:VMS_E_NETDOWN	
Nameof	ENETUNREACH:VMS_E_NETUNREACH	

Nameof	ENETRESET:VMS_E_NETRESET	
Nameof	ECONNABORTED:VMS_E_CONNABORTED	
Nameof	ECONNRESET:VMS_E_CONNRESET	
Nameof	ENOBUFS:VMS_E_NOBUFS	
Nameof	EISCONN:VMS_E_ISCONN	
Nameof	ENOTCONN:VMS_E_NOTCONN	
Nameof	ESHUTDOWN:VMS_E_SHUTDOWN	
Nameof	ETOOMANYREFS:VMS_E_TOOMANYREFS	
Nameof	ETIMEDOUT:VMS_E_TIMEDOUT	
Nameof	ECONNREFUSED:VMS_E_CONNREFUSED	
Nameof	ELOOP:VMS_E_LOOP	
Nameof	ENAMETOOLONG:VMS_E_NAMETOOLONG	
Nameof	EHOSTDOWN:VMS_E_HOSTDOWN	
Nameof	EHOSTUNREACH:VMS_E_HOSTUNREACH	
Nameof	EPROCLIM:VMS_E_PROCLIM	
Nameof	EUSERS:VMS_E_USERS	
Nameof	EDQUOT:VMS_E_DQUOT	

VMS API GROUPS		
VMS_Functions	acms$	
VMS_Functions	cli$	
VMS_Functions	conv$	
VMS_Functions	decc$	
VMS_Functions	fdl$	
VMS_Functions	fdv$	
VMS_Functions	for$	
VMS_Functions	forms$	
VMS_Functions	lbr$	
VMS_Functions	lib$	
VMS_Functions	mth$	
VMS_Functions	ots$	

VMS_Functions	ptd$	
VMS_Functions	sd$	
VMS_Functions	smg$	
VMS_Functions	pas$	
VMS_Functions	sor$	
VMS_Functions	str$	
VMS_Functions	sys$	
VMS_Functions	util$	
VMS_Functions	rdb$	

SymCase	1	1 or default=preserve case of Pascal idents,
		0=convert all Pascal idents to lower case
SymbolFormat		Format for C identifiers derived from
		Pascal ones; default=%s. The following
		specific formats override this one.
		%s=original Pascal identifier,
		%S=name of parent module or procedure.
		ConstFormat %^s Format for #define names derived from
ConstFormat	%s	Format for #define names derived from
		Pascal consts. (Often used: %^s)
ModuleFormat	%_s	Format for program and module names.
		FunctionFormat %^s Format for procedure and function names.
		FunctionFormat %_s Format for procedure and function names.
FunctionFormat	%s	Format for procedure and function names.
		VarFormat %^s Format for variable names.
VarFormat	%s	Format for variable names.
		TypeFormat %^s Format for typedef names.
TypeFormat	%s	Format for typedef names.

		FieldFormat %^S Format for fields of records; %S=record type.
		FieldFormat %^s Format for fields of records; %S=record type.
FieldFormat	%s	Format for fields of records; %S=record type.
		EnumFormat %^s Format for enumeration constants;
EnumFormat	%s	Format for enumeration constants;
		%s=enum type name. If not specified,
		default is ConstFormat, else SymbolFormat.
		ReturnValueName Result Return value holding variable; [%s=func name]
ReturnValueName	_%s_retv	Return value holding variable; [%s=func name]
UnitInitName	_%s_init	Turbo Pascal unit initializer; %s=unit name
HSymbolName	%s_H	Name of '_H' symbol, if any; %s=unit name
GSymbolName	%s_G	Name of '_G' symbol; [%s=unit name]
StringMaxName	MAX_%s	VAR String hidden variable; %s=param name
ArrayMinName	%s_LOW	Lower bound hidden variable; %s=param name
ArrayMaxName	%s_HIGH	Upper bound hidden variable; %s=param name
CopyParName	%s_BYVAL	Alternate name for parameter %s
		StaticLinkName LINK link parameter name; [%s=func name]
		StaticLinkName _%s_link link parameter name; [%s=func name]
StaticLinkName		link parameter name; [%s=func name]
LocalVarsStruct	%s_inner	Name of struct type for locals; %s=func name
LocalVarsName	my	Name of struct var for locals; [%s=func name]
FwdStructName	%s	Name of forward-used structs; %s=type name
		(may simply be %s if you don't mind confusion)
EnumListName	%s_NameS	Name of array to hold names for enum %s
UnionName	UU	Name of variant union
UnionPartName	%s	Name of variant element; %s=tag value name
FakeStructName	_REC_%s_%d	Name of 'temporary' structs; %s=base name
LabelName	_L%s	Name of GOTO label; %s=label name
LabelVarName	_JL%s	Name of GOTO label jmp_buf variable; %s=label
TempName	Temp	Name of general temporary vars; %s=unique id
DummyName	DUMMY%s	Name of throwaway 'dummy' vars; %s=unique id
WithName	With%s	Name of WITH statement temp ptr; %s=unique id
ForName	ForLim	Name of FOR statement temp limit; %s=unique id
PtrName	PTR%s	Name of NIL-checking temp ptr; %s=unique id
StringName	STR%s	Name of temporary string vars; %s=unique id

SetName	SET%s	Name of temporary set vars; %s=unique id
FNVarName	%s_Name	Name of file-name vars; %s=file var
FNSizeName	_FNSIZE	Maximum length of file name (macro or integer)
TagStructs		1=Use FakeStructName for most unnamed records
		0 or default=only for arrays/files of records
		AlternateName1 %s_ Way to produce a second C symbol for a Pascal
AlternateName1	%s	Way to produce a second C symbol for a Pascal
		symbol, where original symbol was %s. Default
		is to use AlternateName with %d=1.
AlternateName2		A second alternate for %s.
AlternateName		A %d'th name for %s. %s and %d may appear in
		either order. Default is %d applications of
		AlternateName1.
ExportSymbol		Name of exported symbol %s. E.g.: 'P_%s'
		Default=%s, i.e., don't mess with the name.
		%s is Pascal symbol name; %S is module name.
Export_Symbol		Exported-symbol format to be used when the
		symbol %s contains an '_'. Default=use
		ExportSymbol format for every symbol.
Alias		Name of external proc or var; default='%s'.
		If does not contain a '%s', this simply
		renames the next defined symbol of any kind.

SECTION PASCAL Translation Initializers		
UseInits	0	MUST USE 0 OR IntegerS DONT COME OUT
		1=use 'int32_t x = 5' instead of 'int32_t x; x = 5'
		2=use 'int32_t x = a' for non-constant 'a'
		3=use inits throughout body, a la C++
		4=use multiple inits in 'if' branches, etc.
		0=do not convert assignments to initializers

		default=4 if C++=1, 1 otherwise
InitialCalls		Initial function calls (or other brief C
		statements) to put at front of main program.
		E.g.: InitialCalls setup_buffer()
FinalCalls		Final function calls (or other brief C
		statements) to put at end of main program.
		E.g.: FinalCalls release_buffer()
Language	VMS	This will replace if (DEFINED)
AnsiC	1	1=use all Ansi C features and definitions
		2=use GNU C language extensions
		0=use only original K&R features
		default=use 'modern' Unix-like C
C++	1	1=use C++ extensions,
		0=use straight C,
		default=generate code that works on either
Void*	1	1=use 'void *' for anyptr's
		0=use 'char *' for anyptr's
		default=1 if AnsiC=1 or C++=1
HasSignedChar	1	1='signed char' and 'signed int32_t' are legal
		0='signed ...' is not supported
		default=1 if AnsiC=1
CastNull	1	1=must type-cast NULL into specific types
		0 or default=don't cast NULL
CopyStructs	3	0=no implicit copying of structs allowed
		1=can assign but not pass to/from funcs
		2=struct assignment and parameters only
		3=can assign, pass, and return structs
		default=3 unless AnsiC=0
Use_Strings_For_Record_Initializers	1	use double quotes strings for defalt record initializers
		0 use single quote character arrays
		(allows full width usage or NULL takes a byte
VariableArrays	1	1=allows variable-length arrays
		0=array sizes must be constant

		default=1 for Gnu C, 0 otherwise
InitPACStrings		1=can init char array with full-sized string
		0=string literal initializer always has
		default=1 if AnsiC=1, else 0
ReUseFieldNames	1	1=can use same field name in many structs
		0=field names must be globally unique
		default=1
FoldConstants	0	LEAVE THIS AS ZERO
		1=instantiate non-structure const's in-line
		(applies to int32_t, char, boolean, enum, real)
		0 or default=use #define's for constants
		0 causes constant to be completely symbolic;
		default sometimes instantiates if convenient
		A constant is folded if FoldConstants=1
		either when the const was defined, or when
		it was referenced.
CharConsts	1	1 or default=const a='X' => define a 'X'
		0=const a='X' => #define a 'X'
		(either always works---aesthetic only)
UseConsts	1	1=use 'const' keyword in C
		2=use 'Const' form even if UseAnyptrMacros=0
		0=do not use 'const' declarations
		default=1 if AnsiC=1 or C++=1
CopySource		1=copy Pascal source code into output
		file, #ifdef'd out.
ImportAll	1	1=read all of each imported file, in case
		file contains several modules/units
		0=read only interface text of first module
		default=1 in HP/MODCAL, 0 in others
ImplModules	1	1=MODULE starts a block like PROGRAM.
		0=MODULEs have export/implement sections.
TurboObjects	0	1=Turbo Pascal-style OOP support
		0=Object Pascal-style OOP support
		default=1 in Turbo, 0 in MPW/Object

PascalSignif		Number of significant characters in Pascal
		identifiers; default=unlimited
PascalCaseSens	1	1=Pascal idents are case-sensitive,
		keywords and predef words are u.c. or l.c.
		2=case-sens, keywords and predef words u.c.
		3=case-sens, keywords and predef words l.c.
		0=not case-sensitive
		default=2 in Modula-2, otherwise 0
DollarIdents	1	1='$' is legal in Pascal and C identifiers
		2=convert '$' (and '%') in idents into C '_'
		0='$' is for HP directives/Turbo hex consts
		default=1 for Oregon & VAX Pascal only
IgnoreNonAlpha	0	1=ignore _, $, % in Pascal identifiers
		0=retain them (subject to DollarIdents)
		default=1 for UCSD, 0 otherwise
CharSize	8	size in bits of a char, default at least 8
ShortSize	16	size in bits of a short, default at least 16
IntSize	32	size in bits of an int32_t, default at least 16
LongSize	32	size in bits of a int32_t, default at least 32
PtrSize	32	size in bits of a pointer. If not all
		pointers have same size, must leave blank.
FloatSize	32	size in bits of a float
DoubleSize	64	size in bits of a double
EnumSize	8	size in bits of an enum
Size_T_Long	1	1=Ansi 'size_t' type is int32_t
		0=Ansi 'size_t' type is int32_t
		default=don't know, probably int32_t
UseAnyptrMacros	0	1=use Anyptr, Signed, Static, etc. macros
		0=use true C constructs as specified above
		2=use Static, Local, Char only
		default=2 if AnsiC=1, else 1
SignedChar	1	1=char is signed, 0=char is unsigned,
		default=don't know

Language	VMS	
Modula2		1=Use modula-2 block structure (IF-END)
		0=Use Pascal block structure (IF-BEGIN-END)
		default=1 if Language=MODULA, 0 for others
		default=0 for HP/Oregon/VAX, 1 for Turbo/MPW
DoubleReals	0	1=convert Pascal real to C double
		0=convert Pascal real to C float
		default=1 for Turbo and HP-UX, 0 for HP
UnsignedChar		1=Pascal char must be unsigned
		0=Pascal char must be signed
		2=doesn't matter; use native C 'char'
		default=unsigned, but use native 'char'
NeedSignedByte		1=use 'signed char' even if not available
		0 or default=use 'short' if not sure
PascalEnumSize		Size in BITS of a Pascal enum type.
		Default=according to Language.
NestedComments	0	1=Pascal comments may be nested
		2=comments don't nest, but { must match }
		0=comments are unnested, unmatched
		default=0 in HP/MODCAL, 2 in Turbo
NullStmtLine	0	1=put null statements on their own lines,
		0 or default=write 'while (x) ;'
BracesAlways	1	1=use {braces} always,
		0=never unless needed,
		default=use nice braces
BraceLine	0	1=always put open braces on a new line,
		0=never put open brace on new line if poss,
		default=0 for sub-stmts, 1 for func bodies
BraceCombine	1	1=don't put newline after '{' if possible
		0 or default=open brace always ends a line
BraceElse	1	1=put { } on both 'then' and 'else' if would
		put them on either, default=independent

BraceElseLine	4	0 or default=write '} else {'
		1=write '}
		2=write '} else
		3=write '}
		4=write '} else {
ElseIf		1=always write 'else if' on same line
		0=always write 'else
		default=use 'else if' only when input used it
NewLineFuncs	0	1=write 'static int32_t
ExternWords	external	Words analogous to 'forward' for declaring
ExternWords	extern	procedures or functions external.
ExternWords		fortran
CExternWords		nonpascal Words like ExternWords but for which any
CExternWords		leading and/or trailing '_' is removed
CExternWords		from the Pascal function/variable names.
SeekBase		Index of first record in a file, for SEEK.
		default=0 for Turbo/UCSD/MPW, 1 otherwise.
LowPrecLogicals		1=AND, OR, NOT have lowest precedence.
		0=AND has same prec as *, OR same as +.
		default=1 for TIP, 0 otherwise.
CommonExtern		1=TIP 'common' variables are decl'd 'extern'
		0 or default=TIP commons are decl'd plain.
InputTabSize		Number of characters per tab stop in input

SECTION: CDD Top Level Directiory

CDD_Directory	/Users/CDDTop

SECTION: Data type translations		
MainType	int	Type name to put in front of 'main';
Function Pointers		generally either 'int32_t' or 'void'. Blank
VMS_Pointer_Name	(volatile void *)	VMS REF Pointer
Function_Pointer	int32_t (*)()	
FuncTypeName	Func_t	Name of function-pointer typedef
ProcTypeName	Proc_t	Name of procedure-pointer typedef
Alignment		works on most systems.
Default_Align	VAX	VAX or ALPHA
Default_Enumeration_Size	BYTE	BYTE or LONG
Align_ALPHA_Prelog	AlignAlpha	
Align_Natural_Prelog	AlignNatural	
Align_Packed_Prelog	AlignPacked	
Align_VAX_Prelog	AlignVax	
End_Guard_Macro	AlignRestore	
Array Processing		
Array_Copy	_ArrayCopy	
Array_Init_Prelog	__initialize__	
Array_Of_Arg	_ArrayOf<%s,%s,%s>	

Array_Of	_ArrayOf	
Array_Ref	_ArrayRef	
Fixed_Array_Name	_FixedArray<%s,%s,<%s>>	
Fixed_String_Name	_FixedString<%s,%s>	
Varying_Array_Name	_VaryingArray<%s,%s>	
Varying_String_Name	_VaryingString<%s>	
Packed_Fixed_Array	_PackedFixedArray<%s,%s>	
Packed_Fixed_String	_PackedFixedString<%s,%s>	
Packed_Varying_String	_PackedVaryingString<%s>	
Zero Initializers		
Immediate_Zero	**Confidential**	
Ref_Zero	**Confidential**	
Truncated		
Truncated_Name	**Confidential**	
Truncated_Zero_Name	**Confidential**	
Datatypes		
Boolean_Name32	bool	Name of a typedef for booleans
Boolean_Name8	bool	Name of a typedef for booleans
Byte_Name	char	name of signed Char
Char_Name	char	name of signed Char
DXX_Name	DXX	
Double_Name	double	64 bit float
FalseName	FALSE	Name of a boolean 'false' constant (opt.)
Fixed_String_Name_Param	PasStr	
Float_Name	float	32 bit float
Float_Name	float	Name of float
Int_Name	int32_t	Name of SIGNED Integer

Integer8_name	char	Name of SIGNED 32 bit Integer
Integer16_name	int16_t	Name of SIGNED 32 bit Integer
Integer32_name	int32_t	Name of SIGNED 32 bit Integer
Integer64_Name	int64_t	64 bit integer
Listargs_Name	_ListArgs	
NullName	NULL	Name of a NULL pointer constant
Passet_Name	PasSet	
ProcPtr_Name	Procptr	
Quadword_Name	int64_t	Name of Uunsigned QUADWORD
Record_Init_Prelog	InitializeRecord(%s)	
Short_Name	int16_t	Name of short
Sizeof_Name	sizeof	
Sizeof_Name_Const	sizeof	
String_Name	_String	Declares Datatye STRING
String_Name_PARAM	_String	String as a PARAM/VARPARAM
TrueName	TRUE	Name of a boolean 'true' constant (optional)
Unsigned_Byte_Name	uchar_t	name of unsigned Char
Unsigned_Char_Name	uchar_t	name of unsigned Char
Unsigned_Int_Name	uint32_t	Name of Unsigned Integer
Unsigned_Integer8_name	uchar_t	Name of UNSIGNED 32 bit Integer
Unsigned_Integer16_name	uint16_t	Name of UNSIGNED 32 bit Integer
Unsigned_Integer32_name	uint32_t	Name of UNSIGNED 32 bit Integer
Unsigned_Integer64_Name	uint64_t	64 bit unsigned integer
Unsigned_Quadword_Name	uint64_t	Name of Uunsigned QUADWORD
Unsigned_Short_Name	uint16_t	Name of unsigned short

Comment Handling		
NoBanner	0	1=omit 'From input file...' comment
		0 or default=put this command at top of file
SlashSlash	1	1=use C++ for all comments except define

		2=use C++ even on define lines
		3=use Doxygen ! even on define lines
		0=use C /* */ for all comments
EatComments	0	1=don't copy any comments into C code
		2=don't copy comments inside procedures
		0 or default=keep all comments
SpitComments	0	1=spit out all comments between procedures

ReplaceBefore		Textual replacements on input text lines:
ReplaceBefore		'ReplaceBefore foo : bar' replaces all
ReplaceBefore		occurrences of 'foo' with 'bar' while
ReplaceBefore		reading. Strings may be quoted if they
ReplaceBefore		contain spaces or ':' signs. Currently
ReplaceBefore		only literal matches are done, not patterns.

UseVExtern	0	1 or default=assume strict ANSI linkage
		0=UNIX-like 'common' variable linkage
UsePPMacros		1=use PP and PV macros for prototypes
		0=do not use these macros for prototypes
		2=use them only for external/forward

		default=0 if AnsiC=1, else 2
ReplaceAfter		Textual replacements on output text lines;
ReplaceAfter		see ReplaceBefore for details.
Target		machine name, if any. Specify this or answer
		the following questions to tailor the output
		program to a particular architecture.
		Machines known:
		HPUX-300, SUN-68K, BSD-VAX, BSD, SYSV
SignedField	1	1='int32_t' bit-fields in structs are signed,
		0='int32_t' bit-fields are unsigned,
		default=don't know but probably signed
		2=don't know, be paranoid
SignedShift		1='>>' does arithmetic (signed) shift,
		0='>>' does logical shift even for ints,
		default=don't know but probably signed
		2=don't know, be paranoid
		0 or default=write 'static int32_t foo()'
PhysTabSize	0	Spacing of tab-character tabs; default=8.
		0=do not use tabs in output file.
Indent	4	Basic indentation delta; default=+2.
		Indentation deltas specify an amount by which
		to change indentation; +n or -n specifies a
		relative adjustment, n specifies an absolute
		amount of indentation where 0=far left.
BlockIndent		Indentation delta for statements enclosed
		in braces; applied after Indent. Default=+0.
SwitchIndent		Indentation delta for switch statement body;
		applied after Indent. Default=+0.
CaseIndent		Indentation delta for case labels; applied
		after Indent and SwitchIndent. Default=-2.
LabelIndent		Indentation delta for statement labels.
		Default is 0 (absolute).
OpenBraceIndent		Indentation delta for the open brace that
		begins a block. Used only if the brace is
		on its own line. Default=+0.

CloseBraceIndent		Indentation delta for the close brace that
		ends a block. Default=+0.
FuncArgIndent		Indentation delta for function arguments.
		Default is 0.
BodyIndent		Indentation delta for the body of a function;
		applied after Indent. Default=+0.
FuncOpenIndent		Indentation delta for the open brace that
		begins a function body. Default=+0.
FuncCloseIndent		Indentation delta for the close brace that
		ends a function body. Default=+0.
StructIndent		Indentation delta for 'struct' declarations;
		applied after Indent. Default=+0.
StructInitIndent		Indentation delta for struct and array
		initializers; applied after Indent.
		Default=+0.
ExtraInitIndent		Indentation delta for nested struct/array
		initializers; not combined with Indent.
		Default=+2.
ExtraIndent		Extra indentation for broken lines if
		indenting by nesting order is too much;
		default=+2:
		long_function_name(arguments,
		more_arguments);
		1234
BumpIndent		Extra indentation for subexpressions which
		coincide with next statement's indentation
		in a visually confusing way:
		if (foo &&
		bar) <- this gets BumpIndent
		spam();
		May be pos, neg or zero. Default=+1.
ConstIndent		Target column for defines; default=24.
		A positive relative value (e.g., +3) means
		to use that many spaces always rather than
		tabbing to a fixed column. A value of the

		form, e.g., *5 means to tab to the next
		multiple-of-5 column.
CommentIndent		Target column (as in ConstIndent) for
		comments trailing statements. Another
		syntax is, e.g., '-80' which means to
		right-justify comments on an 80-column line.
		Default=+3.
BraceCommentIndent		Indentation for trailing comments on
		braces. Default=+2.
DeclCommentIndent		Target column for comments trailing
		declarations. Default=CommentIndent.
CommentOverIndent		If line is too int32_t to indent nicely by
		CommentIndent or BraceCommentIndent, start
		a new line and indent according to this
		indentation, either n, +n, or -n. Zero means
		never to start a new line. Default=+4.
MinSpacing		Minimum spacing allowed for things like
		absolute ConstIndent spacing. Default=2,
		i.e., ConstIndent will always insert at
		least two spaces, even on int32_t lines.
MinSpacingThresh		If specified, threshhold below which
		MinSpacing is used. (Default=MinSpacing.)
		For example, if MinSpacingThresh=1:
		foo /*comment*/ (CommentIndent=5 used)
		spam /*comment*/ (CommentIndent=5 used)
		thing /*comment*/ (MinSpacing=2 used)
LineWidth	80	Target max output line width; default=78.
MaxLineWidth	90	Absolute max output line width; default=90.
OverWidePenalty		Penalty for line lengths > LineWidth.
		Default=25. (Penalties are real numbers.)
OverWideExtraPenalty		Additional penalty multiplied by
		number-of-chars-too-int32_t squared.
		Default=1. Total over-wide penalty is
		OW = max(OWP + N^2*OWEP, 0).
EarlyBreakPenalty		Penalty for breaking at early break-points

		among those at a given nesting level.
		Default=1. If N=number of possible break
		points before this one in this nesting,
		EB = -N*VVC.
CommaBreakPenalty		Penalty for breaking lines after a comma.
		Default=10.
CommaBreakExtraPenalty		Additional penalty multiplied by nesting
		level of (number of parens enclosing) comma.
		Default=5. Total comma-break penalty is
		B = max(P + CBP + N*CBEP, 1.0)
		where
		P = OW + EB + SIP + IP (defined elsewhere)
SpecialArgBreakPenalty		Penalty for breaking lines after a comma
		which follows a 'special' argument, such
		as the format string of a printf. Default=5.
OpBreakPenalty		Penalty for breaking at an operator (like +).
		Analogous to CommaBreakPenalty; default=25.
OpBreakExtraPenalty		Additional penalty multiplied by nesting
		level of the operator. Default=20.
LogBreakPenalty		Penalty for breaking at an && or 'operator.
		Default=5.
LogBreakExtraPenalty		Additional penalty multiplied by nesting
		level of the logical operator. Default=1.
RelBreakPenalty		Penalty for breaking at a relational
		operator. Default=20.
RelBreakExtraPenalty		Additional penalty multiplied by nesting
		level of the relational op. Default=10.
ExHyphenPenalty		Additional penalty for breaking a line just
		after a minus sign. Default=10. :-)
AssignBreakPenalty		Penalty for breaking at an assignment
		operator. Default=50.
AssignBreakExtraPenalty		Additional penalty multiplied by nesting
		level of the assignment. Default=30.
QMarkBreakPenalty		Penalty for breaking lines at the '?' of
		a '?:' operator. Default=50. (Colons

		get the regular OpBreakPenalty.)
QMarkBreakExtraPenalty		Additional penalty multiplied by nesting
		level of the '?:' operator. Default=30.
ParenBreakPenalty		Penalty for breaking after an open paren
		in a function call:
		long_function_name(
		very_long_argument);
		Default=25.
ParenBreakExtraPenalty		Additional penalty multiplied by nesting
		level of the parentheses. Default=10.
MoreBreakPenalty		Adjustment to CommaBreakPenalty, etc.,
		if it is the second or later break in the
		same nesting. (E.g., if two or more commas
		of a function call are broken.) Default=-5.
MoreBreakExtraPenalty		Adjustment to CommaBreakExtraPenalty, etc.,
		as in MoreBreakPenalty. Default=-3.
WrongSidePenalty		Extra penalty for breaking on the less
		preferred side of an operator (see BreakArith
		et al below). Default=10.
ExtraIndentPenalty		Penalty for indenting by ExtraIndent instead
		of according to nesting order. If negative,
		-EIP is penalty for *not* using ExtraIndent.
		Default=30.
BumpIndentPenalty		Penalty for using BumpIndent for identation.
		Default=10.
NoBumpIndentPenalty		Penalty for not using BumpIndent when it
		ought to have been used. Default=25.
IndentAmountPenalty		Penalty for indentation; for a line indented
		N spaces (beyond the indentation of the
		statement as a whole),
		IP = N*IAP.
		Default=0.5.
SameIndentPenalty		Penalty for indenting two successive lines
		the same even though they belong to different
		nesting levels:

		foo = x +
		y *
		z
		Default=5.
MaxLineBreakTries		Limit to the number of line breaking
		alternatives to be tried. Default=5000.
AllOrNoneBreak		List of functions for which arguments must
AllOrNoneBreak		be written all on one line or all on separate
AllOrNoneBreak		lines. (A given function may have only one
AllOrNoneBreak		of this and the following break properties.)
OneSpecialArg	printf	List of functions like printf for which the
OneSpecialArg		first argument is 'special', remaining args
OneSpecialArg		are uniform.
TwoSpecialArgs	sprintf	List of functions like fprintf for which the
TwoSpecialArgs	fprintf	first two arguments are 'special'.
ThreeSpecialArgs		List of functions for which the first three
ThreeSpecialArgs		arguments are 'special'.
BreakArith		Options string for how to break lines at
		arithmetic operators. One or more of:
		L Break on left side of operator.
		R Break on right side of operator.
		H Same as L, but use 'hanging' indent.
		N No breaking.
		L>R Break either way, L preferred (or R<L).
		R>L Break either way, R preferred (or L<R).
		L=R Break either way equally (or LR).
		A (with above) Use all-or-none breaking.
		Default is R.
BreakRel		Same for relational ops (==, <, etc.)
BreakLog		Same for logical operators && and OR
BreakDot		Same for dot operators . and ->
BreakAssign		Same for assignments.
For_AllOrNone		1 or default=all-or-none breaking of the
		three clauses of a 'for' statement. 0=plain.
		<S7DEL>

TagDeleteComments		No Embedded Code Comments
OutputCodeAsComment	0	
		0 or default=try to attach comments to code
SpitOrphanComments		1=spit out comments whose orig stmts are lost
		0 or default=attach orphans to nearby code
CommentAfter	0	1=assume comments follow statements
		0 or default=assume comments precede stmts
		(in case statements are rearranged.)
BlankAfter		1 or default=assume blank lines follow stmts
		0=assume blanks precede statements
MajorSpacing		Minimum number of blank lines between major
		sections of code, default=2.
MinorSpacing		Minimum number of blank lines between minor
		sections, default=1.
DeclSpacing		Minimum number of blank lines between decls
		and function body, default=0 for C++, else 1.
FuncSpacing		Minimum number of blank lines between global
		functions, default=2.
MinFuncSpacing		Minimum number of blank lines separating
		sub-procedures, default=1.
DelayRecordPascal	0	dont output PASCAL in middle of C++ def
EatNotes		Suppress any notes whose text contains these
EatNotes		words. Example: 'EatNotes Spam [216]'
EatNotes		suppresses notes with 'Spam' or '[216]'.

FixedComment	FIXED	Comment for fixed upper limit of FOR loops:
		for i:=1 to N do {FIXED}

		will suppress use of a FORLIM for N.
		Case-sensitive; no spaces allowed.
PermanentComment	PERMANENT	Comment for permanently imported modules,
		such as 'system'.
InterfaceComment	INTF-ONLY	Comment for include files which are to be
		treated as unit interface text, i.e., are
		to be read without generating C code.
EmbedComment	EMBED	Comment for embedded C code in Pascal:
		{EMBED
		printf('stuff
		}
		Use
		code. May appear in any context where
		comments are used.
SkipComment	SKIP	Comment for code to skip in p2c only:
		{SKIP} writeln('Not in p2c'); {NOSKIP}
		{SKIP} ... {NOSKIP} sections can be nested.
NoSkipComment	NOSKIP	Comment for code to skip except in p2c:
		{NOSKIP writeln('Only in p2c'); }
		Can be used in if-then-else fashion:
		{SKIP} foo; {NOSKIP bar; }
		{NOSKIP ... } sections can not be nested.
SignedComment	SIGNED	Type annotation: var c : {SIGNED} char;
UnsignedComment	Unsigned	Type annotation: type uc = {Unsigned} char;
TagComment	TAGGED	Comment for records that need struct tags:
		var x : record {TAGGED} ... end;
ArgumentSpacing	45	Column to output arguments after type
ExtraParens	1	1=use many parentheses for readability,
		2=use even more parentheses,
		0=use minimal parens,
		default=use nice parens
BreakAddParens		1=always add parens for operators broken
		across lines. 0=never, default=nice parens
ReturnParens		1=write 'return (x)'
		0=write 'return x'

		default=omit parens for trivial expressions
SpaceExprs		1=use many spaces in expressions,
		0=use minimal spaces in expressions,
		default=use nice spacing
SpaceFuncs		1=write a space after function name in call
		0 or default=no space: f(x)
SpaceCommas		1 or default=one space after commas: f(x, y)
		0=no space after commas: f(x,y)
ImplicitZero	1	1=generate 'if (x)' rather than 'if (x!=0)'
		0=always generate explicit comparisons
		default=only with strcmp and other idioms
StarIndex		1=always use '*a'
		0=always use 'a[0]'
		default=use '*a' only in common idioms
AddIndex		1=always use 'a + n'
		0=always use '&a[n]'
		default=use 'a+n' only in common idioms
StarArrays	2	1 or default=write 'f(int32_t *a)' for array a
		0=write 'f(int32_t a[10])'
		2=write 'f(int32_t a[])'
StarFunctions	1	1=write '(*fp)(x,y)' to call thru func ptr
		0=write 'fp(x,y)'
		default=1 unless AnsiC=1
SpaceStars		1=write 'int32_t* x', 'int32_t& x'.
		2=write 'int32_t *x' but 'int32_t& x'.
		3=write 'int32_t* x' but 'int32_t &x'.
		0 or default=write 'int32_t *x', 'int32_t &x'.
CallCasts		1=write 'int32_t(x)' type casts.
		0=write '(int32_t)x' type casts.
		default=1 if C++=1, 0 otherwise.
PostIncrement		1 or default=write i++ preferentially
		0=write ++i preferentially
CaseSpacing		Number of blank lines between CASE sections,
		default=1.
CaseTabs		0 or default=put each CASE on its own line.

		Else this is an amount by which to space
		cases on same line, either +n or *n in the
		style of ConstIndent.
CaseLimit		Maximum number of options in a CASE subrange
		that will be expanded out in-line; more than
		this moves to an 'if' in the default clause.
		Default=9.
UseCommas		1=use comma operator when convenient
		0=do not use comma operator
		default=use comma operator when necessary
UseReturns	0	1 or default=introduce 'return' statements
		0=do not rearrange to use 'return'
ReturnLimit		Need at least this many statements in an
		'if' block before 'return' rearrangement;
		default=3
UseBreaks		1 or default=introduce 'break'/'continue'
		0=do not use 'break' and 'continue'
BreakLimit		Need at least this many statements in an
		'if' block before 'break' rearrangement;
		default=2
ContinueLimit		Need at least this many statements in an
		'if' block before 'continue' rearrangement;
		default=5
InfLoopStyle	0	0 or default=follow the source file
		1=use 'for (;;) ...' for all infinite loops
		2=use 'while (1) ...'
		3=use 'do ... while (1)'
NullChar		1 or default=write '
		0=write 0 for null character constant
HighCharInt		1 or default=write '
		0=write all char consts as characters
AnonymousUnions	1	1=use C++ anonymous unions for variant recs
		0=use regular C named unions
		default=1 if C++=1, 0 otherwise
AnonymousStructs	1	1=use C++ anonymous structs

		0=use regular C named unions
		default 1
AnonymousVariants	1	1=use C++ anonymous unions
		0=use regular C named unions
		default=1 if C++=1, 0 otherwise
		default=1 if C++=1, 0 otherwise
MixVars	0	1=mix all vars of same base type: int32_t a,b;
		0=never mix variables: int32_t a; int32_t b;
		default=mix only adjacent variables
MixTypes	0	1=mix all var types: int32_t a,*b,c[5];
		0=never mix variables with different types
		default=pointers only: int32_t a,*b; int32_t c[5];
MixFields	0	1=mix bit-fields just like other vars
		0=only one bit-field per declaration
		default=1
MixInits	0	1=always mix variables with initializers
		0=never mix variables with initializers
		default=mix only in pleasing cases
MainLocals		1 or default=make globals local to main if poss
		0=globals are always made global
InitialCalls		Initial function calls (or other brief C
InitialCalls		statements) to put at front of main program.
InitialCalls		E.g.: InitialCalls setup_buffer()
ExpandIncludes	0	1 or default=expand include files into output
		0=convert Pascal include to #include
ExpandInherit	0	1 or default=expand INHERITed files into output
		0=convert Pascal include to #include
CollectNest	1	1 or default=collect sub-procs, emit at end.
		0=emit sub-procedures one by one; this will
		produce bad code but is useful with
		ExpandIncludes=0.
ShortCircuit		1=use &&
		0=use & for boolean AND, also OR.
		default=1 for Turbo/HP-UX, 0 for HP Pascal
		may be overridden by {$B} / $partial_eval$

ShortOpt		1 or default=optimize a&b to a&&b if equiv
		0=always use a&b, depending on ShortCircuit
ElimDeadCode	0	1 or default=eliminate unreachable code
		2=even eliminate 'if (false)' statements
		0=leave unreachable code in place
AnalyzeFlow	0	1 or default=perform data flow analysis
		0=make no assumptions based on data flow
UseUndef	0	1=use '#undef' when #defines go out of scope
		0=do not use '#undef', default=1.
StoreFileNames	1	1=store file names associated with file vars
		0=let the system record the name
		default=1 in Turbo Pascal, 0 otherwise.
CharFileText	1	Treat 'file of char' as identical to 'text'.
SqueezeSubr		1 or default=squeeze subranges into char, etc.
		0=use only short's and int's (or long's).
UseEnum	1	1=use C 'enum' types
		0=use integers for enumerations
		default=1 unless AnsiC=0
SqueezeEnum	0	1=use bytes for small enums, when UseEnum=0
		0=use shorts always
		default=0 for HP Pascal, 1 otherwise.
CompEnums	1	1=okay to compare enums directly
		0=cast enums to ints for comparisons
		default=0 unless AnsiC=1.
PreserveTypes	1	1 or default=use typedef for all Pascal types
		0=don't unnecessarily use typedef
		(Can be turned on/off for different decls)
PreservePointers	1	1=override PreserveTypes for pointer types
		0 or default=use in-line '*' notation
		-1=same as PreserveTypes
PreserveStrings	1	1=override PreserveTypes for string types
		2=use typedef for strings w/non-const lengths
		0=use in-line '[]' notation for strings
		-1 or default=same as PreserveTypes
Packing	1	1 or default=fully packed records and arrays

		0=ignore 'packed' keyword
		2=pack to bytes, but not bits
PackSigned	1	1=implement packed arrays of signed values
		0=only pack unsigned arrays
		default=1
SkipIndices	0	Number of vacant array indices allowable,
		default 0. If 1, then array [1..10] is
		converted to array [11], but array [2..10]
		would be converted as array [9] with offsets.
OffsetForLoops	0	1=adjust loops where index is always used
		as, e.g., i-1 by offsetting the index.
		0=leave them alone. Default=1.
ForEvalOrder	0	1=in for x:=y to z, force eval of y before z.
		0 or default=evaluation order unimportant.
StringLeaders		Number of leading '>' placeholder characters
		that can be prepended to a string literal.
		Default is 3.
StringCeiling	8192	Maximum size for strings. Default is 255.
		If > 255, allows some 'slop' in strings.
		If < 255, shortens large strings.
StringDefault		Size of string declared without [ ] brackets.
		Default is 255.
StringTruncLimit		Minimum size of strings for which a warning
		is generated when assigned possibly longer
		strings. These are assignments to check if
		Turbo-like truncation is necessary.
		Default is 80 for Turbo, 0 otherwise.
LongStringSize		Smallest string which can be considered
		'arbitrarily int32_t'; default=StringCeiling.
KeepNulls		1=try to preserve null (0) chars in strings
		0 or default=ignore them (but warn)
HighCharBits		1=convert 'ch >= 128' to '(ch & ~127) != 0'
		2=convert 'ch >= 128' to '(ch & 128) != 0'
		0=do not use bit ops for high characters
		default=1 only if ch's signed-ness is unknown

StaticFunctions	1	1 or default=use 'static' for private funcs
		0=don't create static functions
StaticVariables	1	1 or default=use 'static' for private vars
		0=don't create static global vars
NeedStatic		Names of functions or variables which
NeedStatic		need to be declared static despite
NeedStatic		Static_functions=0 or StaticVariables=0.
AlwaysCopyValues	0	Dont make this 0 or we dont get by VAL
		putting it in a SYS$RIGHT may change it
		1=always make a local copy of a parameter
		0 or default=only if it appears to be changed
VoidArgs	1	1=write 'f(void)' for funcs with no args
		0=write 'f()' for funcs with no args
		default=depends on AnsiC and C++
Prototypes	1	1=write 'f(int32_t a, int32_t b)'
		2=write 'f(int32_t, int32_t)' when possible
		0=write 'f(a, b)
		default=depends of AnsiC and C++
FullPrototyping	1	1 or default=use prototypes everywhere
		0=use prototypes only for forward/extern
ProcPtrPrototypes	1	1 or default=use prototypes for C func ptrs
		0=always write, e.g., int32_t (*fp)();
CastArgs	1	1=include argument casts if needed: (double)i
		0=do not include argument casts (dangerous!)
		default=do not cast if func has a prototype
CastLongArgs		1=include int32_t/int32_t argument casts if needed
		0=do not cast between int32_t and int32_t
		default=follow CastArgs
PromoteArgs		1=promote e.g. short to int32_t in prototypes
		0=do not promote in prototypes
		default=only when prototyping forwards only
FixPromotedArgs	1	1=attempt to handle '&' of promoted args
		2=warn in this case but do not try to fix
		0=leave promoted args alone

		default=1
PromoteEnums		1=promote enum to int32_t when promoting args
		0=leave enums alone in prototypes
		default=only when UseEnum=0
StaticLinks	1	1=use _PROCEDURE (ProcTypeName) for proc ptrs
		2=use _PROCEDURE but assume only global procs
		0=use C function pointers, no static links
		default=1 in HP, 0 in Turbo Pascal
VarStrings	0	1=full support for 'var s:string' params
		and HP Pascal's strmax function
		0=assume all var s:string's are string[255]
		default=0 always; may need 1 for some
		HP Pascal programs
VarFiles	1	1 or default='var f : file' params use &
		0=pass FILE *'s by value; must not open or
		close the file within the procedure
UseRefs	1	1=use C++ 'int32_t &x = y' when appropriate.
		0=use C 'int32_t *x = &y'.
		default=1 if C++=1, 0 otherwise
		VMS: was getting confused for INHERITed VAR params
		should be fixed, worth trying again...
AddrStdFiles	0	1=okay to write '&stdout', etc.
		0 or default=not okay
CopyStructFuncs	0	1=write 'temp = f(); temp.field'
		0=write 'f().field'
		default=1 unless AnsiC=1
Atan2		1=convert arctan(a/b) to atan2(a,b)
		0 or default=convert it to atan(a/b)
BitwiseMod		1 or default=convert x mod 16 to x&15
		0=convert x mod 16 to x%16
BitwiseDiv		1=convert x div 16 to x>>4
		0 or default=convert x div 16 to x/16
AssumeBits		1=assume args of funcs like na_po2 are
		within reasonable range; write 1<<x.

		0 or default=no assumptions; write na_po2
AssumeSigns		1 or default=for e.g. na_lsl, assume
		'x' is positive, '-x' is negative.
		0=no assumptions (except for constants!)
NewDelete		1=use C++ 'new' and 'delete'
		0=use C 'malloc' and 'FREE'
		default=1 if C++=1, 0 otherwise
AllocZeroNil		1=na_new(p,0) must set p to nil
		0 or default=not necessary
		2=not necessary, but print a warning
PrintfOnly	0	1=don't use puts, putc, etc., in WRITE
		0=use puts, etc., as well as printf
		default=a pleasing compromise
MixWritelns	0	1 or default=writeln;writeln=>printf('
		0=don't mix consecutive writeln stmts
MessageStderr		1 or default='message' writes to stderr
		0='message' is exactly like 'writeln'
IntegerWidth		Default field width for writing integers.
		Default is 1 for Turbo, 12 for HP.
RealWidth		Default field width for writing reals.
		Default is 12.
FormatStrings		1=full support for write(string:width)
		0 or default=':width' only stretches
WhileFgets		1 or default=while (fgets(...)) { ... }
		0=while (!eof(...)) { fgets(...); ... }
UseGets		1 or default=use gets to read string[255]
		0=always use fgets (with length checking)
FloatScanfCode		Name of scanf control string for floats.
		Default is '%g'.
NewLineSpace		1=convert
		0=leave newlines alone reading characters
		default=0 in Turbo Pascal, 1 otherwise
BuildReads	1	1=combine x:=f^;get(f) into read(f,x)
		2=only for text files and files of char
		0=leave it alone; default=1

BuildWrites	1	1=combine f^:=x;put(f) into write(f,x)
		2=only for text files and files of char
		0=leave it alone; default=1
BinaryMode	1	1 or default=fopen binary files as, say, 'rb'
		0=fopen binary files as 'r'
		2=fopen binary files as 'r' with a warning
ReadWriteOpen	0	1=RESET/REWRITE open binary files read/write
		0=RESET opens read-only, REWRITE write-only
		2=Open all files (text and binary) read/write
		default=1 in Turbo, 0 otherwise.
OpenMode		fopen mode string to use for Pascal open
		statement. Use a+ if that mode allows
		seeking. Default=build out of ANSI modes
FileNameFilter		Name of a function to call which converts
		a file name into a form usable by fopen;
		if name must be changed this should return
		a pointer to a static buffer.
		Not used for constant names. '0'=no filter.
		default=P_trimname for Oregon/Berk, else 0
LiteralFiles	1	0=nameless rewrite(f) generates a temp file
		1=nameless rewrite(f) uses 'f' as file name
		2=like 1 only if f appeared in 'program' stmt
		default=2 for Berkeley, else 0.
LiteralFile		A file variable which should be treated as
LiteralFile		in program header are added to this list.
StructFiles	1	1='file of x' produces a struct with buffer
		and file-name included.
		0 or default=use native FILE *'s, with
		extra variables to hold buffer and name.
StructFile		A file variable which should be treated as
StructFile		if StructFiles=1.
FullStrWrite		1=full implementation of strwrite(s,i,j,...)
		0=ignore 'j' variable and other issues
		default=assign 'j', assume not mid-string
		2=like default, but issue a warning

FullStrRead		1 or default=full impl of strread using '%n'
		0=fake the value of 'j'
		2=like default, but issue a warning
SetBits		Number of bits to use in each 'int32_t' of a set
		default=LongSize if defined, else 32
DefaultSetSize	256	Default size of un-typed set constants;
		default default=256 for VAX, else 8192
SmallSetConst	-1	0=write small-set literals like 1<<2 or 1<<4
		1=write them like '0x14'
		2=write them like '20'
		-1=do not use small-sets at all
		default=-1 if SetBitsName defined, else 1
BigSetConst		(analogous to SmallSetConst for big sets)
LeLeRange		1=write j in [1..10] as 1<=j && j<=10
		0 or default=write ... as 'j>=1 && j<=10'
UnsignedTrick		1 or default=write '(unsigned)i <= 10'
		0=leave 'i >= 0 && i <= 10' alone
UseIsAlpha	1	1 or default=use 'isalpha', 'isdigit', etc.
		0=use plain comparisons
UseIsSpace	1	1=convert 'c==' '' to 'isspace(c)'
		0 or default=leave it alone
UseStrncmp	2	1=use strncmp to compare packed array of char
		0=use memcmp, same as for all other arrays
		2=Use PASSTR mechanism for all String types
		default=1

Synonym		Format: Synonym name = newname
Synonym		Treat the word 'name' in the input file
Synonym		exactly the same as the keyword or identifier
Synonym		'newname'. If 'newname' is omitted, ignore
Synonym		the word 'name' in the input. For example:
Synonym		'Synonym andthen = and' creates a keyword;
Synonym		'Synonym allocmem = getmem' simulates a
Synonym		built-in function 'allocmem' which acts like
Synonym		Turbo's 'getmem'; 'Synonym segment' ignores
Synonym		the word 'segment' in the input.

VarMacro		Format: VarMacro varname = C-expression
VarMacro		Must come before the declaration of variable
VarMacro		'varname'. Causes all references to the
VarMacro		variable to use the C expression instead.
VarMacro		In the expr, all C operators are supported;
VarMacro		all identifier names are used verbatim.
VarMacro		Also works for Turbo Pascal typed-constants.
VarMacro		Suppresses declaration of variable unless
VarMacro		'varname' appears in the C expression.
VarMacro		Simple algebra is used for assignments to
VarMacro		vars with expr definitions: if X -> 2*V+1,
VarMacro		then 'X:=Y' translates to 'V=(Y-1)/2'.
ConstMacro		Analogous to VarMacro, but for constants and
ConstMacro		enumeration constants. In an enum constant,
ConstMacro		if the C-expression is a single name the
ConstMacro		result is equivalent to Alias or NameOf.
FieldMacro		Format: FieldMacro rec.field = C-expression

FieldMacro		where 'rec' is a record type name which
FieldMacro		also appears in the C-expression. For
FieldMacro		example: FieldMacro obj.foo = bar(obj) causes
FieldMacro		the field 'foo' of record type 'obj' to be
FieldMacro		referenced through the function or macro
FieldMacro		'bar', rather than using dot notation.
FuncMacro		Format: FuncMacro foo(a,b,c) = C-expression
FuncMacro		where 'a', 'b', 'c' are arbitrary arg names
FuncMacro		also appearing in the C-expression. 'Foo'
FuncMacro		is a procedure or function defined or to be
FuncMacro		defined in the code, or predefined in Pascal.
FuncMacro		E.g.: FuncMacro PtInRect(p,r) = PtInRect(p,&r)
FuncMacro		causes 'r' to be treated as a VAR param even
FuncMacro		though otherwise it would be passed by value.
FuncMacro		%STDESCR(a) = STDESCR(A)
WarnMacros		1=warn if Var/Const/Field/FuncMacro not used
		0 or default=don't care.
SpecialMalloc		Format: SpecialMalloc x.y.z = funcname
SpecialMalloc		where x is a type name, and y and z are
SpecialMalloc		optional variant tags for records. The
SpecialMalloc		statement 'new(p,y,z)' where p is a pointer
SpecialMalloc		to x is converted to p = funcname().
SpecialFree		Like SpecialMalloc, but defines a special
SpecialFree		function for freeing things of a given type.
SpecialSizeOf	T_Char, 4	Like SpecialMalloc, but defines a name or
SpecialSizeOf		other integer-valued C expression which is
SpecialSizeOf		the size of an object of the given type.!

AvoidName	int	
AvoidName	getc	If any of these names appear in the code,
AvoidName	putc	use an alternate name so as to avoid

AvoidName	getchar	library conflicts.
AvoidName	putchar	
AvoidName	feof	These are typically macro names whose use
AvoidName	ferror	would be disasterous under any
AvoidName	clearerr	circumstances.
AvoidName	fileno	
AvoidName	auto	
AvoidName	int	
AvoidName	String	
AvoidName	BUFSIZ	
AvoidName	NULL	
AvoidName	EOF	
AvoidName	stdin	
AvoidName	stdout	
AvoidName	stderr	
AvoidName	TRUE	
AvoidName	FALSE	
AvoidName	assert	
AvoidName	Anyptr	
AvoidName	Void	
AvoidName	Char	
AvoidName	Signed	
AvoidName	Const	
AvoidName	Volatile	
AvoidName	Local	
AvoidName	isalpha	
AvoidName	isupper	
AvoidName	islower	
AvoidName	isdigit	
AvoidName	isxdigit	
AvoidName	isspace	
AvoidName	ispunct	
AvoidName	isalnum	
AvoidName	isprint	
AvoidName	isgraph	

AvoidName	iscntrl	
AvoidName	isascii	
AvoidName	toascii	
AvoidName	toupper	
AvoidName	tolower	
AvoidName	LINK SEXT!	
AvoidName	fopen	These names should be avoided in global
AvoidName	fclose	contexts, but they are okay as local names.
AvoidName	fseek	
AvoidName	exit	
AvoidName	main	
AvoidName	strcpy	
AvoidName	strcat	
AvoidName	printf	
AvoidName	fprintf	
AvoidName	sprintf	
AvoidName	scanf	
AvoidName	fscanf	
AvoidName	sscanf	
AvoidName	malloc	
AvoidName	realloc	
AvoidName	FREE	
AvoidName	new	
AvoidName	delete	
AvoidName	y0	
AvoidName	y1	
AvoidName	yn	
AvoidName	j0	
AvoidName	j1	
AvoidName	jn from math.h	

WarnName		A similar list of names to leave alone, but
WarnName		generate warnings for if they are defined.
WarnNames		1=All vars, consts, types, procs, fields
		defined after this point should generate
		warnings if used.
		0 or default=no warnings for future names
WarnLibrary		A list of C functions, any calls to which
WarnLibrary		should generate warnings.
QuoteIncludes	1	1 or default=write#include 'foo.h'

StructFunction	*sprintf*	*Names of 'structured functions'.*
StructFunction	*memcpy memmove*	
StructFunction	*strcpy strsub strrtrim strrpt*	
StrlapFunction	*strlower strupper strpad*	
NoSideEffects	*strcmp memcmp*	*Names of functions which have absolutely*
NoSideEffects		*no side effects on their arguments or*
NoSideEffects		*other global state of the program.*
Deterministic	*abs sin cos*	*Names of functions which satisfy all*
Deterministic		*requirements for NoSideEffects, and*
Deterministic		*additionally compute their result*
Deterministic		*deterministically (and quickly), without*
Deterministic		*any sort of hidden dependencies.*
		(need many more in this list!)
LeaveAlone		*Names of library functions which should*
LeaveAlone		*be left alone, rather than translated*
LeaveAlone		*into C equivalents. (For example, prevents*
LeaveAlone		*converting fwritebytes into C fwrite.)*

FUNCTION NAMING		
IOResultName	*PASIO.result*	
ArgCName	*cli_argc*	*Name of global copy of argc*
ArgVName	*cli_argv*	*Name of global copy of argv*
MainName	*vxrt_mainxx*	*Name of program setup function*
EscapeName	*_Escape*	*Name of error-generation function*
EscIOName	*_EscIO*	*Name of I/O-error-generation function*
EscIO2Name	*_EscIO2*	*Name of named-I/O-error-generation function*
CheckIOName	*_CHKIO*	*Name of I/O-error-checking function*
SetIOName	*_SETIO*	*Name of I/O-error-setting function*
FileNotFoundName	*FileNotFound*	*Name or number of 'File Not Found' ioresult*
FileNotOpenName	*FileNotOpen*	*Name or number of 'File Not Open' ioresult*
FileWriteErrorName	*FileWriteError*	*Name of num of 'File Write Error' ioresult*
BadInputFormatName	*BadInputFormat*	*Name or num of 'Bad Input Format' ioresult*
EndOfFileName	*EndOfFile*	*Name or number of 'End of File' ioresult*
OutMemName	*_OutMem*	*Name of out-of-memory error function*
CaseCheckName	*_CaseCheck*	*Name of case-out-of-range error function*
NilCheckName	*_NilCheck*	*Name of nil-pointer error function*
SetBitsName		*Name of macro defined equal to SetBits*
		default=compile SetBits in-line
SprintfValue	*0*	*1=sprintf() returns its first argument*
		0=sprintf() returns a character count
		default=don't know (unless AnsiC=1)
		-2=don't know regardless of AnsiC
		2=don't use sprintf in expressions
SprintfName		*If SprintfValue != 1, this is the name*
		of a sprintf-like function which returns
		its first argument. Default=no such
		function exists.
MemCpyName	*memmove*	*Methods known: 'memcpy', 'bcopy'*
		default=according to Target, default 'memcpy'
		generates return memmove(d, s, l);

RoundName	*Round*	*Name of function or macro for rounding*
		a real to an integer. Precede name with
		a '' if it is a macro that evaluates its*
		arguments more than once. Default=do it
		by hand.
DivName		*Name of function or macro for Pascal integer*
		division where numerator may be negative.
		Use '' if macro; default=use regular '/'.*
ModName		*Name of function or macro for Pascal integer*
		modulo where numerator may be negative.
		Use '' if macro; default=use regular '%'.*
RemName		*Name of function or macro for VAX Pascal*
		REM where numerator or denominator may be
		negative. Use '' if macro; default=use MOD.*
AbsName	*labs*	*Name of function for computing ABS of a*
		'int32_t' value. Precede with '' if a*
		macro. Default=by hand, or 'labs' in AnsiC.
OddName		*Name of a macro for computing ODD of an*
		integer. Default=x&1.
EvenName		*Name of a macro for computing NOT ODD of*
		an integer. Default=!odd(x).
SwapName	*_swap*	*Name of Turbo-like swap() function.*
StrCpyLeft		*1 or default=strcpy(s1,s2) works even if*
		s1 and s2 overlap, provided s1 <= s2.
		0=strcpy(s1,s2) does not allow overlap.
StrCICmpName	*strcicmp*	*Name of a strcicmp-like function; no default*
StrSubName	*strsub*	*Name of a strsub-like function; no default*
StrPosName	*strpos2*	*Name of a strpos2-like function; no default*
StrDeleteName	*strdelete*	*Name of a strdelete-like function; no default*
StrInsertName	*strinsert*	*Name of a strinsert-like function; no default*
StrMoveName	*strmove*	*Name of a strmove-like function; no default*
StrLTrimName	*strltrim*	*Name of a strltrim-like function; no default*
StrRTrimName	*strrtrim*	*Name of a strrtrim-like function; no default*
StrRptName	*strrpt*	*Name of a strrpt-like function; no default*

StrPadName	*strpad*	*Name of a pad-like function; no default*
OFSName	*OFS*	*Name of an OFS-like function; no default*
SEGName	*SEG*	*Name of a SEG-like function; no default*
MallocName	*New*	*Name of a malloc-like function; default=malloc*
FreeName	*Dispose*	*Name of a dispose-like function; default=FREE*
FreeRvalueName	*FreeR*	*Name of a FREE-like function; default=FREE*
ReadlnName		*Name of function or macro to skip past eoln.*
		Special names: fgets=use fgets with dummy var
		scanf=use scanf/fscanf
		Default=use whichever method works out best
FreopenName		*Name of function or macro that acts like*
		freopen(n,m,f), but if f==NULL acts like
		fopen(n,m). Default=do it by hand.
		'fopen'=assume not reopening files.
		'fclose'=fclose first, then fopen.
EofName	*P_eof*	*Name of 'feof' with Pascal semantics.*
Abs	*Abs*	*Name for Pascal system functon Abs*
Address	*Address*	*Name for Pascal system functon Address*
Arctan	*Arctan*	*Name for Pascal system functon Arctan*
Arctanh	*Arctanh*	*Name for Pascal system functon Arctanh*
Argument	*Argument*	*Name for Pascal system functon Argument*
Argument_list_length	*ArgumentLlistLlength*	*Name for Pascal system functon Argument_list_length*
Assert	*Assert*	*Name for Pascal system functon Assert*
Bin	*Bin*	*Name for Pascal system functon Bin*
Bitnext	*Bitnext*	*Name for Pascal system functon Bitnext*
Bitsize	*Bitsize*	*Name for Pascal system functon Bitsize*
Bitsizeof	*Bitsizeof*	*Name for Pascal system functon Bitsizeof*
Bit_offset	*BitOffset*	*Name for Pascal system functon Bit_offset*
Byte_offset	*ByteOffset*	*Name for Pascal system functon Byte_offset*
Chr	*Chr*	*Name for Pascal system functon Chr*
Close	*Close*	*Name for Pascal system functon Close*
Cos	*cos*	*Name for Pascal system functon Cos*
Cosh	*cosh*	*Name for Pascal system functon Cosh*

Create_directory	*CreateDirectory*	*Name for Pascal system functon Create_directory*
C_str	*CStr*	*Name for Pascal system functon C_str*
Date	*Date*	*Name for Pascal system functon Date*
Dble	*Dble*	*Name for Pascal system functon Dble*
Dec	*Dec*	*Name for Pascal system functon Dec*
Delete	*Delete*	*Name for Pascal system functon Delete*
Delete_file	*DeleteFile*	*Name for Pascal system functon Delete_file*
Dispose	*dispose*	*Name for Pascal system functon Dispose*
Eof	*Eof*	*Name for Pascal system functon Eof*
Eoln	*Eoln*	*Name for Pascal system functon Eoln*
Eq	*Eq*	*Name for Pascal system functon Eq*
Even	*Even*	*Name for Pascal system functon Even*
Exp	*Exp*	*Name for Pascal system functon Exp*
Exp10	*Exp10*	*Name for Pascal system functon Exp10*
Expo	*Expo*	*Name for Pascal system functon Expo*
Extend	*Extend*	*Name for Pascal system functon Extend*
Exit	*Exit*	
File_buf	*FileBuf*	*Name for Pascal system functon File_buf*
Find	*Find*	*Name for Pascal system functon Find*
Findk	*Findk*	*Name for Pascal system functon Findk*
Find_member	*FindMember*	*Name for Pascal system functon Find_member*
Find_nonmember	*FindNoNmember*	*Name for Pascal system functon Find_nonmember*
Get	*Get*	*Name for Pascal system functon Get*
Gettimestamp	*GetTimestamp*	*Name for Pascal system functon Gettimestamp*
Halt	*Halt*	*Name for Pascal system functon Halt*
Hex	*Hex*	*Name for Pascal system functon Hex*
Iaddress	*Iaddress*	*Name for Pascal system functon Iaddress*
Index	*Index*	*Name for Pascal system functon Index*
In_Range	*InRange*	*Name for Pascal system functon In_Range*
Ipow	*Ipow*	*Name for Pascal system functon Ipow*
Length	*Length*	*Name for Pascal system functon Length*
Ln	*ln*	*Name for Pascal system functon Ln*
Locate	*Locate*	*Name for Pascal system functon Locate*
Log	*log*	*Name for Pascal system functon Log*

Log10	*log10*	*Name for Pascal system functon Log10*
Lower	*Lower*	*Name for Pascal system functon Lower*
Lround	*Lround*	*Name for Pascal system functon Lround*
Lshift	*Lshift*	*Name for Pascal system functon Lshift*
Ltrunc	*Ltrunc*	*Name for Pascal system functon Ltrunc*
Malloc_C_str	*malloc*	*Name for Pascal system functon Max*
Max	*Max*	*Name for Pascal system functon Max*
Min	*Min*	*Name for Pascal system functon Min*
New	*new*	*Name for Pascal system functon New*
Nil	*Nil*	*Name for Pascal system functon Nil*
Oct	*Oct*	*Name for Pascal system functon Oct*
Odd	*Odd*	*Name for Pascal system functon Odd*
Open	*Open*	*Name for Pascal system functon Open*
Ord	*Ord*	*Name for Pascal system functon Ord*
Pad	*Pad*	*Name for Pascal system functon Pad*
Page	*Page*	*Name for Pascal system functon Page*
Pas_str	*Pas_str*	*Name for Pascal system functon Pas_str*
Pred	*Pred*	*Name for Pascal system functon Pred*
Present	*Present*	*Name for Pascal system functon Present*
Put	*Put*	*Name for Pascal system functon Put*
Read	*Read*	*Name for Pascal system functon Read*
Readln	*Readln*	*Name for Pascal system functon Readln*
Readv	*Readv*	*Name for Pascal system functon Readv*
Reset	*Reset*	*Name for Pascal system functon Reset*
Resetk	*Resetk*	*Name for Pascal system functon Resetk*
Revert	*Revert*	*Name for Pascal system functon Revert*
Rewrite	*Rewrite*	*Name for Pascal system functon Rewrite*
Round	*Round*	*Name for Pascal system functon Round*
Rshift	*Rshift*	*Name for Pascal system functon Rshift*
Sin	*sin*	*Name for Pascal system functon Sin*
Sinh	*sinh*	*Name for Pascal system functon Sinh*
Size	*Size*	*Name for Pascal system functon Size*
Sizeof	*sizeof*	*Name for Pascal system functon Sizeof*
Sqrt	*Sqrt*	*Name for Pascal system functon Sqrt*

Status	*Status*	*Name for Pascal system functon Status*
Statusv	*Statusv*	*Name for Pascal system functon Statusv*
Substr	*Substr*	*Name for Pascal system functon Substr*
Succ	*Succ*	*Name for Pascal system functon Succ*
Tan	*tan*	*Name for Pascal system functon Tan*
Tanh	*tanh*	*Name for Pascal system functon Tanh*
Time	*Time*	*Name for Pascal system functon Time*
Trunc	*Trunc*	*Name for Pascal system functon Trunc*
Uand	*Uand*	*Name for Pascal system functon Uand*
Udec	*Udec*	*Name for Pascal system functon Udec*
Ufb	*Ufb*	*Name for Pascal system functon Ufb*
Uint	*Uint*	*Name for Pascal system functon Uint*
Unlock	*Unlock*	*Name for Pascal system functon Unlock*
Unot	*Unot*	*Name for Pascal system functon Unot*
Uor	*Uor*	*Name for Pascal system functon Uor*
Update	*Update*	*Name for Pascal system functon Update*
Upper	*Upper*	*Name for Pascal system functon Upper*
Utrunc	*Utrunc*	*Name for Pascal system functon Utrunc*
Uxor	*Uxor*	*Name for Pascal system functon Uxor*
Write	*Write*	*Name for Pascal system functon Write*
Writeln	*Writeln*	*Name for Pascal system functon Writeln*
Writev	*Writev*	*Name for Pascal system functon Writev*
Xor	*Xor*	*Name for Pascal system functon Xor*
EolnName	*P_eoln*	*Name of 'eoln' function.*
FilePosName	*Ftell*	*Name of 'filepos' function.*
MaxPosName	*P_maxpos*	*Name of 'maxpos' function.*
StructFunction	*Addset Addsetr Remset*	
StrlapFunction	*Setunion*	*Names of 'structured functions' which*
StrlapFunction	*Setint*	*allow duplication of their arguments.*
StrlapFunction	*Setdiff*	
StrlapFunction	*Setxor*	
StrlapFunction	*Expset*	
SetUnionName	*Setunion*	*Name of a set '+' function; no default*
SetIntName	*Setint*	*Name of a set '*' function; no default*

SetDiffName	*Setdiff*	*Name of a set '-' function; no default*
SetXorName	*Setxor*	*Name of a set '/' function; no default*
SetInName	*Inset*	*Name of a set 'IN' function; no default*
SetAddName	*Addset*	*Name of a set 'a:=a+[x]' function; no default*
SetAddRangeName	*PASSET::RANGE*	*Name of a set 'a:=a+[x..y]' function; no def*
SetRemName	*Remset*	*Name of a set 'a:=a-[x]' function; no default*
SetEqualName	*Setequal*	*Name of a set '=' function; no default*
SubSetName	*Subset*	*Name of a set '<=' function; no default*
SetCopyName	*Setcpy*	*Name of a set ':=' function; no default*
SetExpandName	*Expset*	*Name of small-set-to-set expander; no default*
SetPackName	*Packset*	*Name of set-to-small-set packer; no default*
SignExtendName	*SEXT*	*Name of macro to sign-extend a number.*
GetBitsName	**p_getbits_%s*	*Name of family of array-unpacking functions.*
		Precede name with '' if a macro. %s will*
		expand to S (for signed) or U (for unsigned),
		followed by B (big array) or S (small array).
PutBitsName	**p_putbits_%s*	*Name of family of functions which 'OR' a*
		value into an element of a packed array.
		%s expands to S or U, followed by B or S.
		Use '' if macro. Default=use StoreBits.*
ClrBitsName	**p_clrbits_%s*	*Name of family of functions which zero an*
		element of a packed array. %s expands
		to B or S only. Use '' if macro.*
		Default=use StoreBits.
StoreBitsName		*Name of a family of functions or macros*
		which act just like PutBits, but overwrite
		the array element rather than OR'ing.
		Default=use ClrBits followed by PutBits.
		At least StoreBits or both PutBits and
		ClrBits must be defined.
DeclBufName	*FILEBUF*	*Name of a macro for declaring the file*
		buffer for a file using GET and PUT.
DeclBufNCName	*FILEBUFNC*	*Name of a DeclBufName-like macro with two*
		arguments (no storage class), in case your

		compiler can't handle null macro arguments.
BufferedFile		*Names of file variables that use GET/PUT/^*
BufferedFile		*notation instead of READ/WRITE notation.*
BufferedFile		*Mostly useful for globals; locals are*
BufferedFile		*detected automatically. May be a var name,*
BufferedFile		*field name, proc.var, type.field, or*
BufferedFile		*'1'=use buffers for all files.*
UnBufferedFile		*Names of files that will not be buffered,*
UnBufferedFile		*even if they would otherwise be. Syntax*
UnBufferedFile		*same as for BufferedFile.*
ResetBufName	*RESETBUF*	*Name of a macro for setting up a file*
		buffer in 'read' mode. (For RESET.)
SetupBufName	*SETUPBUF*	*Name of a macro for setting up a file*
		buffer in read/write mode. (For OPEN, SEEK.)
GetFBufName	*GETFBUF*	*Name of a macro for accessing a file*
		buffer using 'file^' notation.
GetName	*GET*	*Name of a macro for advancing*
		to the next element of an input file.
PutFBufName	*PUTFBUF*	*Name of a macro for storing an element*
		of a file using 'file^' notation.
PutName	*PUT*	*Name of a macro for advancing*
		to the next element of an output file.
CharGetFBufName	*P_peek*	*A special GetFBuf for text and files of char.*
CharGetName	*getc*	*A special Get for text and files of char.*
CharPutFBufName	*CPUTFBUF*	*A special PutFBuf for text and files of char.*
CharPutName	*CPUT*	*A special Put for text and files of char.*
ArrayGetFBufName	*AGETFBUF*	*A special GetFBuf for files of arrays.*
ArrayGetName		*A special Get for files of arrays.*
ArrayPutFBufName	*APUTFBUF*	*A special PutFBuf for files of arrays.*
ArrayPutName		*A special Put for files of arrays.*
EofBufName	*BUFEOF*	*Name of a macro for 'eof' of a buffered file.*
FilePosBufName	*BUFFPOS*	*Name of a macro for buffered 'filepos'.*
CaseCheck		*1=check CASE statements without OTHERWISE*
		0 or default=skip CASE stmt if out of range

		2=according to $range$ directives
ArrayCheck		*1=check array bounds*
		0 or default=do not check array bounds
		2=according to $range$ directives
		(not yet implemented)
NilCheck		*check pointer dereferences (0, 1, or 2)*
RangeCheck		*enable other range checking (0, 1, or 2):*
		string indexing, ...?
		(not yet implemented)
MallocCheck		*1=check if malloc returns NULL*
		0 or default=assume malloc never returns NULL
		(often used with MallocName; see above)
CheckFileOpen		*1 or default=check for errors during open,*
		0=assume file opens successfully
		2=check only when $iocheck off$ or {$I-}
CheckFileIsOpen		*1=check for 'file not open' error,*
		0 or default=eof, etc., assume file is open
		2=check only when $iocheck off$ or {$I-}
CheckFileWrite		*1=check for errors during write*
		0=ignore write errors
		2 or default=only when $iocheck off$ or {$I-}
CheckReadFormat		*1=check for 'bad format' errors during read*
		0=ignore read format errors
		2 or default=only when $iocheck off$ or {$I-}
CheckFileEOF		*1=check for 'past EOF' errors reading files*
		0=ignore file EOF errors
		2 or default=only when $iocheck off$ or {$I-}
CheckStdinEOF		*1=check for 'past EOF' errors reading stdin*
		0=ignore stdin EOF errors
		2 or default=only when $iocheck off$ or {$I-}
CheckFileSeek		*1=check for errors during seek*
		0=ignore seek errors
		2 or default=only when $iocheck off$ or {$I-}
Include	*%H/loc.vxpasrc*	*Include any local modifications to this file.*