Neuron Data Elements Environment Element Application Services

Version 4.0

C Reference Summary

Part Number: OI-6000-0796-00

Contents

ARGS_ Class

Function	Returns	Arguments	Descriptions
`GetAll`	ArrayPtr	(void);	Returns list of all arguments (including application name itself).
`GetExecName`	CStr	(void);	Returns the application name.
`GetFirst`	CStr	(void);	Returns the first argument after the application name.
`GetNext`	CStr	(void);	Returns the next argument.
`GetNth`	CStr	(ArgIVal *n*);	Returns the nth argument.
`GetNum`	**ArgIVa**l	(void);	Returns number of arguments (including application name itself).
`Init`	void	(CRTL_int *argc*, CRTL_char** *argv*);	Should be called from main routine before any other initialization.
`InsertNth`	void	(ArgIVal *n*, CStr *arg*);	Inserts the new argument arg into the list at given index n.
`RemoveNth`	void	(ArgIVal *n*);	Extract the nth argument from the list.

ArNum Class

Function	Returns	Arguments	Description
~NDArNum 7	void	**(ArNumPtr** *arnum***);**	Destructor
AppendElt 10	void	**(ArNumPtr** *arnum***);**	
ContainsElt 9	void	**(ArNumPtr** *arnum***);**	
ExtractElt 12	void	**(ArNumPtr** *arnum***);**	
ExtractNthElt 12	void	**(ArNumPtr** *arnum***);**	
FindElt 9	void	**(ArNumPtr** *arnum***);**	
GetLen 8	ArrayIVal	**(ArNumPtr** *arnum***);**	Returns the number of elements in the ARNUM.
GetNthElt 8	void	**(ArNumPtr** *arnum***);**	
InsertNthElt 11	void	**(ArNumPtr** *arnum***);**	
IsEmpty 8	BoolEnum	**(ArNumPtr** *arnum***);**	Returns whether the ARNUM is empty or not.
IsInRange 8	BoolEnum	**(ArNumPtr** *arnum***);**	
IsSorted 13	void	**(ArNumPtr** *arnum***);**	
LookupElt 9	void	**(ArNumPtr** *arnum***);**	

Function	Returns	Arguments	Description
NDArNum 6	void	**(ArNumPtr** *arnum***);**	Default ARNUM construction.
RemoveDupls 13	void	**(ArNumPtr** *arnum***);**	
RemoveElt 11	void	**(ArNumPtr** *arnum***);**	
RemoveNthElt 11	void	**(ArNumPtr** *arnum***);**	
Reset 7	void	**(ArNumPtr** *arnum***);**	Resets the contents of the AR-NUM
SetAlloc 7	void	**(ArNumPtr** *arnum***, Ar-rayIVal** *alloc***);**	Reallocates the contents of the ARNUM
SetLen 7	void	**ArNumPtr** *arnum***, Ar-rayIVal** *len***);**	Sets the number of elements of the ARNUM to 'len' and reallo-cates the contents of the AR-NUM if necessary.
SetNthElt 8	void	**(ArNumPtr** *arnum***);**	
Sort 12	void	**(ArNumPtr** *arnum***);**	
SortedExtract-Elt 12	void	**(ArNumPtr** *arnum***);**	
SortedFindElt 10	void	**(ArNumPtr** *arnum***);**	
SortedInsertElt 11	void	**(ArNumPtr** *arnum***);**	
SortedLooku-pElt 10	void	**(ArNumPtr** *arnum***);**	
SortedRemove-Dupls 13	void	**(ArNumPtr** *arnum***);**	
SortedUniqIn-sertElt 11	void	**(ArNumPtr** *arnum***);**	

Function	Returns	Arguments	Description
UnboundedGet-NthElt 8	void	**(ArNumPtr** *arnum***);**	
UnboundedSet-NthElt 9	void	**(ArNumPtr** *arnum***);**	

AROBJOF*AROBJ_ELT*_ Class

Function	Returns	Arguments	Descriptions
`~NDArObj`	void	ArrayIVal *start*, ArrayIVal *num*);	Destroys the array and all its elements..
`AppendElt`	void	(const *AROBJ_ELT elt*);	Adds `elt' at the end of the array.
`ContainsElt`	BoolEnum	(const *AROBJ_ELT elt*);	Returns BOOL_TRUE if the array contains an object equal to `elt'.
`ExtractElt`	void	(const AROBJ_ELT *elt*);	Same as corresponding NDArObjOfAROBJ_ELT::RemoveElt call but preserves the relative ordering of the elements in the array.
`ExtractNthElt`	void	ArrayIVal *i*);	Removes the element at index `i'.
`FindElt`	ArrayIVal	const *AROBJ_ELT elt*);	Same as AROBJ_Lookup routine but signal a failure if `elt'.
`GetLen`	ArrayIVal	(	Returns the number of elements in the array.
`GetNthElt`			
`InsertNthElt`	AROBJ_ELT	ArrayIVal *i*);	Returns a const reference to the element at index `i'.
`IsEmpty`	void	(ArrayIVal *i*);	Returns whether the array is empty.
`IsInRange`	void	(ArrayIVal *i*);	Returns BOOL_TRUE if `i' is a valid index for the array (in the range [0, len-1] where len is the length of the array).
`IsSorted`	void	(CmpProc *proc*);	Returns BOOL_TRUE if the array is sorted according to `proc'.
`LookupElt`	ArrayIVal	(const *AROBJ_ELT elt*);	Returns the index of the first occurrence of an object which is equal to `elt'.
`NDArObj`	void	(ArrayIVal *start*, ArrayIVal *num*);	Default AROBJ construction.
`RemoveDupls`	void	(void);	Removes duplicate elements in the array.
`RemoveElt`	void	(const AROBJ_ELT *elt*);	Removes the first occurence of `elt' in the array.
`RemoveNthElt`	void	(ArrayIVal *i*);	Removes the element at index `i'.
`Reset`	void	(void);	Resets the contents of the array.

Function	**Returns**	**Arguments**	**Descriptions**
`SetAlloc`	void	(ArrayIVal *alloc*);	Reallocates the capacity of the array for `alloc' elements if necessary but does not change the number of elements in the array.
`SetLen`	void	(ArrayIVal *len*);	Sets the number of elements of the array to `len' .
`SetNthElt`	void	(ArrayIVal *i*, const *AROBJ_ELT elt*);	Sets the element at index `i' to copy to object ’elt’.
`Sort`	void	CmpProc *proc*);	Sorts the array using `proc' to compare the elements.
`SortedExtractElt`	void	CmpProc *cmp*, const AROBJ_ELT *elt*);	Extracts `elt', using `proc' to compare elements of the array.
`SortedFindElt`	ArrayIVal	;(CmpProc *proc*, *AROBJ_KEY key*);	Searches element which matches `key' in the array.
`SortedInsertElt`	void	(CmpProc *proc*, const *AROBJ_ELT elt*);	Insert a copy of `elt', using `proc' to compare addresses the array elements.
`SortedLookupElt`	BoolEnum	(CmpProc *proc*, *AROBJ_KEY key*, ArrayIValPtr *result*);	Searches element which matches key in the array.
`SortedRemoveDupls`	void	void);	Removes duplicates in the array, assumes that it is sorted.
`SortedUniqInsertElt`	ArrayIVal	CmpProc *proc*, const *AROBJ_ELT elt*);	Same as NDArObjOfAROBJ_ELT::SortedInsertElt but does not insert if the element is already in the array.
`UniqAppendElt`	void	(const *AROBJ_ELT elt*);	Appends `elt' to the array if `elt' is not already in the array.

ARPTR_ Class

Function	Returns	Arguments	Description
`AppendElt`	void	(ArPtrPtr *arptr*, ARPTR_ELT *elt*);	Adds \`elt' at the end of the ARPTR.
`Construct`	void	(ArPtrPtr *arptr*);	Default ARPTR constructor.
`ConstructAlloc`	void	(ArPtrPtr *artpr*, ArrayIVal *alloc*);	Constructs the ARPTR with 0 elements but a buffer allocated for \`alloc' elements.
`ConstructArPtr`	void	(ArPtrPtr *arptr*, ArPtrPtr *arptr2*);	Constructs the ARPTR as a copy of \`arptr2'
`ContainsElt`	BoolEnum	(ArPtrCPtr *arptr*, ARPTR_ELT *elt*);	Returns whether or not the ARPTR contains \`elt'.
`ExtractElt`	void	(ArPtrPtr *arptr*, ARPTR_ELT *elt*);	Same as corresponding ARPTR_Remove calls but preserve the relative ordering of the elements in the ARPTR.
`ExtractNthElt`	void	(ArPtrPtr *arptr*, ArrayIVal *i*);	Removes the element at index \`I'.
`FindElt`	ArrayIVal	(ArPtrCPtr *arptr*, ARPTR_ELT *elt*);	Same as ARPTR_Lookup routines but signal a failure if the ARPTR does not contain \`elt'.
`GetLen`	ArrayIVal	(ArPtrCPtr *arptr*);	Returns the number of elements in the ARPTR.
`GetNthElt`	ARPTR_ELT	(ArPtrCPtr *arptr*, ArrayIVal *i*);	Returns the element at index \`I'.
`GetNthEltAddr`	ARPTR_ELT Ptr	(ArPtrCPtr *arptr*, ArrayIVal *i*);	Returns the address of the element at index \`I'.
`InsertNthElt`	void	(ArPtrPtr *arptr*, ArrayIVal *i*, ARPTR_ELT *elt*);	Inserts \`elt' at index \`I'.
`IsEmpty`	BoolEnum	(ArPtrCPtr *arptr*);	Returns whether the ARPTR is empty or not.
`IsInRange`	BoolEnum	(ArPtrCPtr *arptr*, ArrayIVal *i*);	Returns whether \`i' is a valid index for the ARPTR (in the [0, len-1] range, where len is the length of the ARPTR).
`IsSorted`	void	(ArPtrCPtr *arptrc*, CmpProc *proc*);	Returns whether a is sorted or not according to \`proc'.

Function	Returns	Arguments	Description
`LookupElt`	ArrayIVal	(ArPtrCPtr *arptr*, ARPTR_ELT *elt*);	Returns the index of the first occurrence of `elt' in the ARPTR. Returns -1 if the ARPTR does not contain `elt'.
`RemoveDupls`	void	(ArPtrPtr *arptr*);	Removes duplicate elements in the ARPTR.
`RemoveElt`	void	(ArPtrPtr *arptr*, ARPTR_ELT *elt*);	Removes the first occurrence of `elt' in the ARPTR.
`RemoveNthElt`	void	(ArPtrPtr *arptr*, ArrayIVal *i*);	Removes the element at index `I'.
`Reset`	void	(ArPtrPtr *arptr*);	Resets the contents of the ARPTR.
`SetAlloc`	void	(ArPtrPtr *arptr*, ArrayIVal *alloc*);	Reallocates the contents of the ARPTR for `alloc' elements if necessary but does not change the number of elements in the ARPTR.
`SetLen`	void	(ArPtrPtr *arptr*, ArrayIVal *len*);	Sets the number of elements of the ARPTR to `len' and reallocates the contents of the ARPTR if necessary.
`SetNthElt`	void	(ArPtrPtr *arptr*, ArrayIVal *i*, ARPTR_ELT *elt*);	Sets the element at index `I'.
`Sort`	void	(ArPtrPtr *arptr*, CmpProc *proc*);	Sorts the ARPTR.
`SortedExtractElt`	ArrayIVal	(ArPtrPtr *arptr*, CmpProc *cmp*, ARPTR_ELT *elt*);	Extracts `elt', using `proc' to compare elements of the ARPTR.
`SortedFindElt`	ArrayIVal	(ArPtrCPtr *arptr*, CmpProc *proc*, ARPTR_KEY *key*);	Searches element which matches `key' in the ARPTR.
`SortedInsertElt`	ArrayIVal	(ArPtrPtr *arptr*, CmpProc *proc*, ARPTR_ELT *elt*);	Insert `elt', using `proc' to compare elements of the ARPTR.
`SortedLookupElt`	BoolEnum	(ArPtrCPtr *arptr*, **CmpProc** *proc*, **ARPTR_KEY** *key*, **ArrayIValPtr** *result*);	Searches element which matches key in the ARPTR.
`SortedRemoveDupls`	void	(ArPtrPtr *arptr*);	Removes duplicates in the ARPTR, assumes that it is sorted.

Function	Returns	Arguments	Description
`SortedUniqInsertElt`	ArrayIVal	(ArPtrPtr *arptr*, CmpProc *proc*, ARPTR_ELT *elt*);	Same as ARPTR_SortedXXX calls but do not insert if the element is already in the ARPTR.
`UnboundedGetNthElt`	APPTR_ELT	(ArPtrCPtr *arptr*, ArrayIVal *i*);	Same as ARPTR_GetNthElt but returns 0 if `i' is out of range instead of failing.
`UnboundedSetNthElt`	void	(ArPtrPtr *arptr*, ArrayIVal *i*, ARPTR_ELT *elt*);	Same as ARPTR_SetNthElt but extends the array if `i' is out of range and elt is not NULL (`i' must be positive).
`UniqAppendElt`	void	(ArPtrPtr *arptr*, ARPTR_ELT *elt*);	Appends `elt' to the ARPTR if `elt' is not already in the ARPTR.

ARREC_ Class

Function	Returns	Arguments	Description
`AppendElt`	void	(ArRecPtr *arrec*, ARREC_ELTPtr *elt*);	Adds elt at the end of the AR-REC.
`ContainsElt`	BoolEnum	;(ArRecCPtr *arrec*, ARREC_ELTPtr *elt*);	Returns whether or not the ARREC contains elt.
`ExtractElt`	void	(ArRecPtr *arrec*, ARREC_ELTPtr *elt*);	Same as corresponding ARREC_Remove calls but preserve the relative ordering of the elements in the ARREC.
`ExtractNthElt`	void	(ArRecPtr *arrec*, ArrayIVal *i*);	Removes the element at index I.
`FindElt`	ArrayIVal	(ArRecCPtr arrec, ARREC_ELTCPtr elt);	Same as ARREC_Lookup rou-tine but signal a failure if elt.
`GetLen`	ArrayIVal	(ArRecCPtr *arrec*);	Returns the number of ele-ments in the ARREC.
`GetNthElt`	ARREC_EL T	;(ArRecCPtr *arrec*, ArrayIVal *i*);	Returns the address of the ele-ment at index I.
`InsertNthElt`	void	(ArRecPtr *arrec*, ArrayIVal *i*, ARREC_ELTPtr *elt*);	Inserts elt at index I.
`IsEmpty`	BoolEnum	(ArRecCPtr *arrec*);	Returns whether the ARREC is empty or not.
`IsInRange`	BoolEnum	(ArRecCPtr *arrec*, ArrayIVal *i*);	Returns whether i is a valid in-dex for the ARREC (in the [0, len-1] range, where len is the length of the ARREC).
`IsSorted`	BoolEnum	(ArRecCPtr *arrec*, CmpProc *proc*);	
`LookupElt`	ArrayIVal	(ArRecCPtr *arrec*, ARREC_ELTCPtr *elt*);	Returns the index of the first occurrence of elt in the AR-REC.
`RemoveDupls`	void	(ArRecPtr *arrec*);	Removes duplicate elements in the ARREC.
`RemoveElt`	void	(ArRecPtr *arrec*, ARREC_ELTPtr *elt*);	Removes the first occurrence of elt in the ARREC.
`RemoveNthElt`	void	(ArRecPtr *arrec*, ArrayIVal *i*);	Removes the element at index I.
`Reset`	void	(ArRecPtr *arrec*);	Resets the contents of the AR-REC.
`SetAlloc`	void	(ArRecPtr *arrec*, ArrayIVal *alloc*);	Reallocates the contents of the ARREC for alloc elements if necessary but does not change the number of elements in the ARREC.

Function	**Returns**	**Arguments**	**Description**
`SetLen`	void	(ArRecPtr *arrec*, ArrayIVal *len*);	Sets the number of elements of the ARREC to len and reallocates the contents of the ARREC if necessary.
`SetNthElt`	void	(ArRecPtr *arrec*, ArrayIVal *i*, ARREC_ELTPtr *elt*);	Sets the element at index I.
`Sort`	void	(ArRecPtr *arrec*, CmpProc *proc*);	Sorts the ARREC by passing the address of the elements instead of the elements themselves to the comparison routine.
`SortedExtractElt`	ArrayIVal	(ArRecPtr *arrec*, CmpProc *cmp*, ARREC_ELTPtr *elt*);	Extracts elt, using proc to compare elements of the ARREC
`SortedFindElt`	ArrayIVal	(ArRecCPtr *arrec*, CmpProc *proc*, ARREC_KEY *key*);	Searches element which matches key in the ARREC.
`SortedInsertElt`	ArrayIVal	(ArRecPtr *arrec*, CmpProc *proc*, ARREC_ELTPtr *elt*);	Insert elt, using proc to compare addresses of the ARREC elements
`SortedLookupElt`	BoolEnum	(ArRecPtr *arrec*, CmpProc *proc*, ARREC_KEY *key*, ArrayIValPtr *result*);	Searches element which matches key in the ARREC. t
`SortedRemoveDupls`	void	(ArRecPtr *arrec*);	Removes duplicates in the ARREC, assumes that it is sorted
`SortedUniqInsertElt`	ArrayIVal	(ArRecPtr *arrec*, CmpProc *proc*, ARREC_ELTPtr *elt*);	Same as Sorted calls but do not insert if the element is already in the ARREC
`UniqAppendElt`	void	(ArRecPtr *arrec*, ARREC_ELTPtr *elt*);	Appends elt to the ARREC if elt is not already in the ARREC.

BBUF_ Class

Function	Returns	Arguments	Description
`CurPos`	BBufOff-setVal	(BBufCPtr *bbuf*);	Returns current position.
`Flush`	void	(BBufPtr *bbuf*);	Calls the FlushProc method.
`GetClientData`	ClientPtr	(BBufCPtr *bbuf*);	Respectively, returns user-defined data set by BBUF_SetClientData and sets the ClientData.
`GetCurPtr`	BBuf-BytePtr	(BBufCPtr *bbuf*);	Returns a pointer to the byte at current position, and modifies the pointer to the byte at current position.
`GetEndianity`	EndianE-num	(BBufCPtr *bbuf*);	Returns the real order of bytes in inte-gers.
`GetPageBeginPos`	BBufOff-setVal	(BBufCPtr *bbuf*);	Returns the offset to the first byte in current page
`GetPageBeginPtr`	BBuf-BytePtr	(BBufCPtr *bbuf*);	Returns a pointer to the first byte of current page
`GetPageEndPtr`	BBuf-BytePtr	(BBufCPtr *bbuf*);	Returns a pointer to the first byte after current page, and sets the pointer to the first byte after current page.
`GetPagingData`	ClientPtr	(BBufCPtr *bbuf*);	Returns PagingData.
`GetTotalSize`	BBufOff-setVal	(BBufCPtr *bbuf*);	Returns the total size of data
`IsPageModified`	BoolEnum	(BBufCPtr *bbuf*);	Respectively, returns BOOL_TRUE if current page has been modi-fied,BOOL_FALSE otherwise, and sets / unsets the PageModified flag.
`LoadCurPage`	void	(BBufPtr *bbuf*);	Loads the current page (if needed); .
`NDBBuf`	void	(BBufPtr *bbuf*, BBufBytePtr *data*, BBufOffsetVal *len*);	Constructs the bbuf to point to data.
`QueryMethods`	void	(BBufCPtr *bbuf*, BBufMethodsPtr *methods*);	Fills methods with the methods in-stalled in the bbuf.
`ReadInt8`	void	(BBufPtr *bbuf*, Int8Ptr *valptr*);	Reads an Int8 and writes it into valptr.
`ReadInt16`	void	(BBufPtr *bbuf*, Int16Ptr *valptr*);	Reads an Int16 and writes it into valp-tr.
`ReadInt32`	void	(BBufPtr *bbuf*, Int132Ptr *valptr*);	Reads an Int32 and writes it into valp-tr.
`ReadNBytes`	void	(BBufPtr *bbuf*, HugeBytePtr *ptr*, BBufOffsetVal *len*);	Reads len bytes from bbuf and puts the result into to ptr..
`ReadUInt8`	void	(BBufPtr *bbuf*, UInt8Ptr *valptr*);	Reads an UInt8 and writes it into valp-tr.

Function	Returns	Arguments	Description
`ReadUInt16`	void	(BBufPtr *bbuf*, UInt16Ptr *valptr*);	Reads an UInt16 and writes it into valptr.
`ReadUInt32`	void	(BBufPtr *bbuf*, UInt32Ptr *valptr*);	Reads an UInt32 and writes it into valptr.
`SeekBy`	void	(BBufPtr *bbuf*, BBufOffsetVal *pos*);	Sets position to offset relative to current position.
`SeekTo`	void	(BBufPtr *bbuf*, BBufOffsetVal *pos*);	Sets position to absolute offset.
`SetClientData`	void	(BBufPtr *bbuf*, ClientPtr *data*);	Sets the ClientData.
`SetCurPtr`	void	(BBufCPtr *bbuf*, BButBytePtr *cur*);	Modifies the pointer to the byte at current position.
`SetEndianity`	void	(BBufPtr *bbuf*, EndianEnum *endian*);	Sets the real order of bytes in integers for the bbuf.
`SetMethods`	void	(BBufPtr *bbuf*, BBufMethodsPtr *methods*);	Installs the methods in methods in the bbuf.
`SetPageBeginPos`	void	(BBufPtr *bbuf*, BBufOffsetVal *pos*);	Sets the offset to the first byte in current page.
`SetPageBeginPtr`	void	(BBufPtr *bbuf*, BBufBytePtr *pageBeg*);	Respectively, returns a pointer to the first byte of current page, and sets the pointer to the first byte of current page.
`SetPageEndPtr`	void	(BBufPtr *bbuf*, BBufBytePtr *pageEnd*);	Returns a pointer to the first byte after current page.
`SetPageModified`	void	(BBufPtr *bbuf*, BoolEnum *mod*);	Sets / unsets the PageModified flag.
`SetPagingData`	void	(BBufPtr *bbuf*, ClientPtr *data*);	Modifies PagingData.
`SetTotalSize`	void	(BBufPtr *bbuf*, BBufOffsetVal *len*);	Sets the total size of data for the bbuf.
`SkipRead`	void	(BBufPtr *bbuf*, BBufOffsetVal *pos*);	Skips <n> bytes from current position.
`SkipWrite`	void	(BBufPtr *bbuf*, BBufOffsetVal *pos*);	Skips <n> bytes from current position. I
`WriteInt8`	void	(BBufPtr *bbuf*, Int8 *val*);	Writes an Int8 respectively into the bbuf.
`WriteInt16`	void	(BBufPtr *bbuf*, Int16 *val*);	Writes an Int16 into the bbuf.
`WriteInt32`	void	(BBufPtr *bbuf*, Int32 *val*);	Writes an Int32 into the bbuf.
`WriteNBytes`	void	(BBufPtr *bbuf*, HugeByteCPtr *ptr*, BBufOffsetVal *len*);	Writes len bytes of ptr to the bbuf.
`WriteUInt8`	void	(BBufPtr *bbuf*, UInt8 *val*);	Writes an UInt8 into the bbuf.
`WriteUInt16`	void	(BBufPtr *bbuf*,UInt16 *val*);	Writes an UInt8 respectively into the bbuf.
`WriteUInt32`	void	(BBufPtr *bbuf*, Int16 *val*);	Writes an UInt16 into the bbuf.

CHAR_ Class

Function	Returns	Arguments	Description
`AsciiAlphaGetBase`	Int	(Char *ch*);	Returns the base value of an ASCII letter.
`AsciiDigitGetInt`	Int	(Char *ch*);	Returns the integer value of an ASCII digit.
`AsciiGetControl`	Char	(Char *ch*);	Converts a character to a control character.
`AsciiGetEbcdic`	Byte	(Byte *b*);	Converts an ASCII character to an EBCDIC character.
`AsciiGetGraph`	Char	(Char *ch*);	Converts a control character into a character.
`AsciiGetLower`	Char	(Char *ch*);	Converts an ASCII character to lower case.
`AsciiGetUpper`	Char	(Char *ch*);	Converts an ASCII character to upper case.
`AsciiHexDigitGetInt`	Int	(Char *ch*);	Returns the integer value of an ASCII digit.
`AsciiIsAlNum`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII.
`AsciiIsAlpha`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII.
`AsciiIsControl`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII.
`AsciiIsDigit`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII
`AsciiIsGraph`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII.
`AsciiIsHexDigit`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII.
`AsciiIsLower`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII.
`AsciiIsOctDigit`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII.
`AsciiIsPrint`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII
`AsciiIsPunct`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII.

`AsciiIsSpace`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII
`AsciiIsUpper`	BoolEnum	(Char *ch*);	Same as NDChar::IsAscii... macros except that an error is generated if the character is not ASCII
`AsciiOctDigitGetInt`	Int	(Char *ch*);	Returns the integer value of an ASCII digit.
`CodeGetLen`	StrIVal	(ChCode *chcode*);	Returns the length of the character whose first byte contains the specified -bit code.
`FromAscii`	Char	(Byte *b*);	Converts an ASCII code to a native character.
`EbcdicGetAscii`	Byte	(Byte *b*);	Converts an EBCDIC character to an ASCII character.
`GetByte`	Char	(ChCode *chcode*, Int *byte-num*);	Returns the contents the specified byte of a multibyte character
`GetLen`	StrIVal	(Char *ch*);	Returns the length of the character whose first byte contains the specified bit character.
`IsAscii`	BoolEnum	(ChCode *chcode*);	Determines whether the character corresponds to the ASCII set..
`IsAsciiAlNum`	BoolEnum	(ChCode *chcode*);	Determines whether the character corresponds to.a letter or a digit.
`IsAsciiAlpha`	BoolEnum	(ChCode *chcode*);	Determines whether the character corresponds to.a letter.
`IsAsciiControl`	BoolEnum	(ChCode *chcode*);	Determines whether the character corresponds to.an ASCII control character.
`IsAsciiDigit`	BoolEnum	(ChCode *chcode)*	Determines whether the character corresponds to.a digit.
`IsAsciiGraph`	BoolEnum	(ChCode *chcode*);	Determines whether the character corresponds to.a graph character.
`IsAsciiHexDigit`	BoolEnum	(ChCode *chcode*);	Determines whether the character corresponds to.a hexadecimal digit.
`IsAsciiLower`	BoolEnum	(ChCode *chcode)*	Determines whether the character corresponds to.a lower case letter.
`IsAsciiOctDigit`	BoolEnum	(ChCode *chcode*);	Determines whether the character corresponds to.an octal digit.
`IsAsciiPrint`	BoolEnum	(ChCode *chcode*);	Determines whether the character corresponds to.a printable character.
`IsAsciiPunct`	BoolEnum	(ChCode *chcode)*	Determines whether the character corresponds to.an ASCII punctuation.
`IsAsciiSpace`	BoolEnum	(ChCode *chcode*);	Determines whether the character corresponds to.an ASCII octal space.
`IsAsciiUpper`	BoolEnum	(ChCode *chcode*);	Determines whether the character corresponds to an upper case character..

CHAR Class

`NatGetByte`	NatChar	NatCode *natcode*, Int *byte-num*);	Returns the contents the specified byte of a multibyte character.
`NatGetByte`	NatChar	(NatCode *natcode*);	Returns the contents the specified byte of a multibyte character.
`NatGetByte`	NatChar	(NatCode *natcode*);	Returns the contents the specified byte of a multibyte character.
`NatGetByte`	NatChar	(NatCode *natcode*);	Returns the contents the specified byte of a multibyte character.
`NatGetLen`	StrIVal	(NatChar *natCh*);	Returns the length of the native character whose first byte contains the specified -bit character code.
`ToAscii`	Byte	(Char *ch*);	Converts a native character to ASCII.

CHAR Class

CS_ Class

Function	Returns	Arguments	Description
`Construct`	void	(CsPtr *cs*);	Default code set object constructor.
`ConstructId`	void	(CsPtr *cs*, CsIdEnum *csid*);	Constructs the code set object from the `csid' information.
`CvtChar`	BoolEnum	(CsCPtr *cs*, CsCode *in*, CharCvtSet *flags*, LgEnvCPtr *lgenv*, CsCodePtr *out*);	Convert the character in `in' described in `flags' and set the result to `out'. `lgenv' specifies a language environment.
`Destruct`	void	(CsPtr *cs*);	Default code set object destructor.
`FromUni`	BoolEnum	(CsCPtr *cs*, CsCode *cscode*, CsCodePtr *uni*);	Converts unicode to cscode.
`GetCharInfo`	CharInfoVal	(CsCPtr *cs*, CsCode *code*);	Get the 'charinfo' value of the character `code'.
`GetCharLen`	StrIVal	(CsCPtr *cs*);	Get the character length for the code set.
`GetCsGlobal`	CsPtr	(void);	Returns a pointer to the global code set.
`GetCsId`	CsIdEnum	(CsCPtr *cs*);	Get the code set's id.
`GetCsNative`	CsPtr	(void);	Returns a pointer to the native code set.
`GetCsUnicode`	CsPtr	(void);	Returns a pointer to the Unicode code set.
`TransChar`	BoolEnum	(CsCPtr *cs*, CsCode *code*, CharCvtSet *flags*, CsCPtr *incs*, CsCodePtr *chcodeptr*);	Translates the character of specified code set to the character within this code set.
`ToUni`	BoolEnum	(CsCPtr *cs*, CsCode *cscode*, UniCodePtr *uni*);	Converts cscode to unicode. If it cannot be converted, returns false; otherwise, sets the unicode to uni and return true.

CT_ Class

Function	Returns	Arguments	Function
`CvtChar`	BoolEnum	(CtCPtr *ct*, ChCode *in*, CharCvtSet *flags*, LgEnvCPtr *lgenv*,ChCodePtr *out*);	Converts a character and sets the result.
`CvtCsToCt`	ChCode	(CtCPtr *ct*, CsCode *code*, CsCPtr *cs*);	Converts a character code from its code set form to its code type form.
`CvtCtToCs`	CsCode	(CtCPtr *ct*, ChCode *ch*, CsPtr* *cs*);	Converts a character code from its code type form to its code set form.
`FromUni`	BoolEnum	(CtCPtr *ct*, UniCode *uni*, ChCodePtr *ch*);	Converts unicode to chcode.
`GetBwrd`	ChCode	(CtCPtr *ct*, CStr *str*, StrIVal *pos*, StrIValPtr *lenp*);	Returns the character code for the character found in front of a given index in a string.
`GetCharLen`	StrIVal	(CtCPtr *ct*, Char *ch*);	Returns the length of a global character of a specified code type.
`GetCtId`	CtIdEnum	(CtCPtr *ct*);	Returns the code type id from the code type data record structure.
`GetFwrd`	ChCode	(CtCPtr *ct*, CStr *str*, StrIValPtr *lenp*);	Returns the value of the character found at the beginning of a string.
`GetInfo`	CharInfoVal	(CtCPtr *ct*, ChCode *ch*);	Returns the CharInfoVal for a character.
`GetLower`	ChCode	(CtCPtr *ct*, ChCode *ch*);	Returns the lower case form of a character.
`GetMaxCharLen`	StrIVal	(CtCPtr *ct*);	Returns the maximum character length supported by a code type.
`GetUpper`	ChCode	(CtCPtr *ct*, ChCode *ch*);	Returns the upper case form of a character.
`IsSingleOnly`	BoolEnum	(CtCPtr *ct*);	Determines whether a code type defines single-byte characters only.
`ToUni`	BoolEnum	(CtCPtr *ct*, ChCode *ch*, UniCodePtr *uni*);	Converts chcode to unicode.

DS_ Class

Function	Returns	Arguments	Description
`AddContDs`	void	(DsPtr *ds*, DsPtr *contDs*);	Adds contDs as a contained data source to the data source.
`Class`	RClasPtr	(void);	Returns a pointer to the DataSource Class.
`Create`	DsPtr	(RClasPtr *rclas*);	Creates a datasource object..
`GetViewOption`	CStr	(DsCPtr *ds*, ResCPtr *view*, CStr *option*);	Returns the string corresponding to option for the view registered with the data source.
`RegisterView`	void	(DsPtr *ds*, ResPtr *view*);	Register the resource view with the data source.
`RemoveContDs`	void	(DsPtr *ds*, DsPtr *contDs*);	Removes contDs from the data source.
`SetViewOption`	void	(DsPtr *ds*, ResPtr *view*, CStr *option*, CStr *info*);	Set info as the option for the view registered in the data source.
`StartEdit`	DsEditPtr	(DsPtr *dsEdit*);	Opens an edition on the whole data source.
`StartUpdateEdit`	DsUpdateEditPtr	(DsPtr *ds*);	Opens an update on the whole data source.
`UnregisterView`	void	(DsPtr *ds*, ResPtr *view*);	Unregisters view from the data source.
`ViewGetDs`	DsPtr	(ResPtr *view*);	Returns the data source, if any, associated to the view.

DSEDIT_ Class

Function	Returns	Arguments	Description
`Abort`	void	(DsEditPtr *dsEdit*);	Abort the edition on the data source.
`AddOperation`	void	(DsEditPtr *dsEdit*);	Add an operation to the edition
`End`	DsEditCompletionEnum	(DsEditPtr *dsEdit*);	Commit the edition on the whole of the data source.
`GetOwner`	ResPtr	(DsEditPtr *dsEdit*);	Retrieve owner (if any) of the edition

Function	Returns	Arguments	Description
`SetOwner`	void	(DsEditPtr *dsEdit*, ResPtr *owner*);	Set owner of the edition.

DSUPDATEEDIT_ Class

Function	Returns	Arguments	Description
`Abort`	void	(DsUpdateEditPtr *dsEdit*);	Abort the update on the data source.
`End`	void	(DsUpdateEditPtr *dsEdit*);	Commit the update on the whole of the data source.

FILE_ Class

Function	Returns	Arguments	Description
`Backup`	void	(FilePtr *fileobj*);	Creates a backup of the fileobj. The file must not be open at the time of the call.
`Close`	void	(FilePtr *fileobj*);	Close a file.
`CreateOpen`	void	(FilePtr *fileobj*, FMgrCreateFileCPtr *create,* FileFmtEnum *format*);	Creates a new file and opens it in the given format.
`CurBinaryOffset`	FileOffsetVal	(FileCPtr *fileobj*);	Return the current position offset in a binary file.
`CurLineNumber`	FileLineNbVal	(FileCPtr *fileobj*);	Return the current line number in a file.
`CurSize`	FileOffsetVal	(FileCPtr *fileobj*);	Return the current size of a file.
`CurTextOffset`	FileOffsetVal	(FileCPtr *fileobj*);	Return the current position offset in a text file.
`Find`	BoolEnum	(FilePtr *fileobj*);	Searches for a file specified by its spec name.
`Flush`	void	(FilePtr *fileobj*);	Flushes file output buffer.
`GetAutoBackup`	BoolEnum	(FileCPtr *fileobj*);	Get the value of a file object's auto backup flag.
`GetClientData`	ClientPtr	(FileCPtr *fileobj*);	Gets the client data attached to a file object.
`GetDefSearchPath`	CStr	(void);	Set the default search path used by NDFile::Open.
`GetDefSearchPathName`	CStr	(void);	Get the name of the environment variable containing the default search path.
`GetError`	FileErrEnum	(FileCPtr *fileobj*);	Return the last error generated.
`GetFailIfNotFound`	BoolEnum	(FileCPtr *fileobj*);	Get the value of a file structure's FailIfNotFound flag.
`GetFailOnEof`	BoolEnum N	(FileCPtr *fileobj*);	Get the value of a file structure's FailOnEOF flag.
`GetNodeType`	FMgrNodeEnum	(FileCPtr *fileobj*);	Determines the type of a node.

`GetOpenFormat`	FileFmtEnum	(FileCPtr *fileobj*);	Determines the file format and I/O mode in which a file is open.
`GetOpenMode`	FileFmtEnum	(FileCPtr *fileobj*);	Determines the I/O mode in which a file has been opened.
`GetRealName`	CStr	(FileCPtr *fileobj*);	Get the file name used by Open.
`GetSearchPath`	CStr	(FileCPtr *fileobj*);	Get the search path used by Open.
`GetSpecName`	CStr	(FileCPtr *fileobj*);	Get the file name used by Open.
`GotoBeg`	void	(FilePtr *fileobj*);	Seek to the beginning of a file.
`GotoEnd`	void	(FilePtr *fileobj*);	Seek to the end of a file.
`IsAtEnd`	BoolEnum	(FileCPtr *fileobj*);	Return whether the current position is the end of the file.
`IsOpen`	BoolEnum	(FileCPtr *fileobj*);	Determine if a file is open, and in which modes it is open.
`IsOpenBinary`	BoolEnum	(FileCPtr *fileobj*);	Determine if a file is open, and in which modes it is open.
`IsOpenLine`	BoolEnum	(FileCPtr *fileobj*);	Determine if a file is open, and in which modes it is open.
`IsOpenRead`	BoolEnum	(FileCPtr *fileobj*);	Determine if a file is open, and in which modes it is open.
`IsOpenText`	BoolEnum	(FileCPtr *fileobj*);	Determine if a file is open, and in which modes it is open.
`IsOpenWrite`	BoolEnum	(FileCPtr *fileobj*);	Determine if a file is open, and in which modes it is open.
`IsReadable`	BoolEnum	(FileCPtr *fileobj*);	Determines whether the given file has read access.
`IsWritable`	BoolEnum	(FileCPtr *fileobj*);	Determines whether the given file has write access.
`Open`	void	(FileCPtr *file*, FileIOEnum *iomode*, FileFmtEnum *format*);	Open a file with the specified I/O mode and format.

`QueryLinePos`	void	(FileCPtr *file*, FileLinePosPtr *posptr*);	Query the current position for a file opened in line format.
`QueryNatRef`	void	(FileCPtr *file*, FileNatRefPtr *nat*);	Attaches a different native file to an existing file. It does not close the old native file.
`QueryTextPos`	void	(FileCPtr *fileobj*, FileTextPosPtr *posptr*);	Query the current position structure for a text file.
`ReadByte`	Int	(FilePtr *fileobj*);	Return the next byte in a binary file.
`ReadChar`	Int	(FilePtr *fileobj*);	Return the next character in a text file.
`ReadLine`	CStr	(FilePtr *fileobj*);	Read one line of text from a line format file.
`ReadNBytes`	FileOffsetVal	(FilePtr *file*, VoidPtr *buffer*, FileOffsetVal *n*);	Read a number of bytes from a binary file.
`ReadNChars`	FileOffsetVal	(FilePtr *fileobj*, Str *fileobj*, FileOffsetVal *n*);	Read and return a string of characters from a text file.
`ReadStr`	CStr	(FilePtr *fileobj*);	Read and return a string of characters from a text file.
`ReadTextLine`	CStr	(FilePtr *fileobj*);	Read and return a line of characters from a text file.
`SeekBinaryBy`	void	(FilePtr *fileobj*, FileOffsetVal *position*);	Set the file position relative to the current position for a binary file.
`SeekBinaryTo`	void	(FilePtr *fileobj*, FileOffsetVal *position*);	Set the current absolute position in a binary file.
`SetAutoBackup`	void	(FilePtr *fileob*,BoolEnum *autobackup*);	Set the value of a file object's auto backup flag.
`SetClientData`	void	(FilePtr *file*, ClientPtr *clientdata*);	Sets the client data attached to a file object.
`SetDefSearchPath`	void	(CStr *path*);	Set the default search path used by NDFile::Open.
`SetDefSearchPathName`	void	(CStr *path*);	Get the name of the environment variable containing the default search path.
`SetError`	void	(FilePtr *file*, FileErrEnum *fileerr*);	Set the error for a file.

`SetFailIfNotFound`	void	(BoolEnum *flag*);(FilePtr *file-obj*,BoolEnum *fail*);	Set the value of a file structure's FailIfNot-Found flag.
`SetFailOnEof`	void	(BoolEnum *flag*);(FilePtr *file-obj*,BoolEnum *fail*);	Set the value of a file structure's FailOnEOF flag.
`SetLinePos`	void	(FilePtr *fileobj*, FileLinePosCPtr *posptr*);	Set the current position in a file opened in line format.
`SetNatRef`	void	(FilePtr *file*, FileNatRefCPtr *nat*);	Attaches a different native file to an existing file. It does not close the old native file.
`SetSearchPath`	void	(FilePtr *fileobj*, CStr *path*);	Set the search path used by NDFile::Open.
`SetSpecName`	void	(FilePtr *fileobj*, CStr *specname*);	Set the file name used by NDFile::Open.
`SetTextPos`	void	(FilePtr *fileobj*, FileTextPosCPtr *posptr*);	Set the current position in a text file.
`Truncate`	void	(FilePtr *fileobj*)	Truncate a file at the current position.
`TryClose`	BoolEnum	(FilePtr *fileobj*);	Non-asserting version of the file close function.
`TryCreateOpen`	BoolEnum	(FilePtr *file*, FMgrCreateFileCPtr *createptr*, FileFmtEnum *format*);	Non-asserting version of the file create function.
`TryOpen`	BoolEnum	(FilePtr *fileobj*, FileIOEnum *iomode*, FileFmtEnum *format*);	Non-asserting version of the file open function.
`WriteByte`	void	(FilePtr *file*, Int *byte*);	Writes a byte to a binary file.
`WriteChar`	void	FilePtr *fileobj*, Int *char*);	Writes a character to a text file.
`WriteLine`	void	(FilePtr *fileobj*, CStr *string*);	Write a string and line terminator to a text file.
`WriteNBytes`	void	(FilePtr *file*, VoidCPtr *buffer*, File-OffsetVal*n*);	Writes a number of bytes to a binary file.
`WriteNChars`	void	(FilePtr *file*, CStr *buffer*, FileOff-setVal *n*);	Writes a number of characters to a text file.
`WriteStr`	void	(FilePtr *fileobj*, CStr *string*);	Write a null terminated string to a text file.
`WriteTextLine`	void	(FilePtr *fileobj*, CStr *string*);	Write a string and line terminator to a text file.

FMGR_ Class

Function	Returns	Arguments	Description
`AddFileType`	void	(FMgrFileTypeCPtr *fmgrfile-type*)	Adds a file type.
`AllFilesWildCard`	CStr	(void);	Return wildcard pattern that matches all the files in a directory.
`CheckDir`	BoolEnum	(CStr *name*);	Checks the type of a node without performing I/O operations on VMS.
`CheckFile`	BoolEnum	(CStr *name*);	Checks the type of a node without performing I/O operations on VMS.
`CopyDir`	void	(CStr *orname*, CStr *destname*);	Copy a directory and all of its content.
`CopyFile`	void	(CStr *orname*, CStr *destname*);	Copy a file.
`CopyNode`	void	(CStr *orname*, CStr *destname*);	Copy a node
`CreateDir`	void	(CStr *name*, FMgrCreate-DirCPtr *createInfo*);	Create a directory with the specified permission rights.
`CreateFile`	void	(CStr *name*, FMgrCreateFi-leCPtr *createInfo*);	Creates a file called name with the specified permission rights and Macintosh signatures.
`DeleteDir`	void	(CStr *name*)	Delete a directory and all its content.
`DeleteDirContent`	void	(CStr *name*)	Delete the contents of a directory.
`DeleteFile`	void	(CStr *name*)	Delete a file
`DeleteNode`	void	(CStr *name*)	Delete a node
`DirWildCard`	CStr	(void);	Return wildcard pattern that matches only directories.
`Exists`	BoolEnum	(CStr *name*);	Determines whether the specified node exists.
`FindFileTypeId`	FMgrFile-TypeEnum	(CStr *name*);	Returns the FileTypeId for the given file.
`FindFileTypeInfo`	FMgrFile-TypeCPtr	(CStr *name*);	Same as FindFileTypeId but returns the full type description instead of just the FileTypeId.
`GetMacCreator`	FMgr-MacIdVal	(CStr *name*);	Returns the Macintosh signatures of a file.
`GetMacType`	FMgr-MacIdVal	(CStr *name*);	Returns the Macintosh signatures of a file.
`GetNodeType`	FMgrN-odeEnum	(CStr *name*);	Determines the type of the specified node.
`GetNthFileType`	FMgrFile-TypeCPtr	(ArrayIVal *n*);	Returns the nth register file type description.
`GetNumFileTypes`	ArrayIVal	(void);	Returns the number of registered file types.

`IsDevConcealed`	BoolEnum	(CStr *name*)	Checks whether a concealed device is present for the file or directory passed.
`IsDir`	BoolEnum	(CStr *name*)	Macros for checking the type of a node.
`IsExecutable`	BoolEnum	(CStr *name*)	Functions for checking the access permissions of a node.
`IsFile`	BoolEnum	(CStr *name*)	Macros for checking the type of a node.
`IsReadable`	BoolEnum	(CStr *name*)	Functions for checking the access permissions of a node.
`IsVolume`	BoolEnum	(CStr *name*)	Macros for checking the type of a node.
`IsWritable`	BoolEnum	(CStr *name*)	Functions for checking the access permissions of a node.
`MoveDir`	void	(CStr *orname*, CStr *orname*);	Rename and / or move a directory
`MoveFile`	void	(CStr *orname*, CStr *destname*);	Rename and / or move a file.
`MoveNode`	void	(CStr *orname*, CStr *destname*);	Rename and / or move a node.
`PerfDirFiles`	PerfEnum	(Str *dir*, Str *pattern*, FMgrPerfFileProc *func*, ClientPtr *data*);	Performs an action on all the entries of a directory dir which match a given pattern.
`PerfVolumes`	void	(FMgrPerfVolProc *func*, ClientPtr *data*);	Call a user function for each volume in the system.
`PurgeDir`	void	(CStr *dir*, CStr *pattern*);	Purge specified files from a directory.
`QueryNodeInfo`	BoolEnum	(CStr *name*, FMgrNodePtr *fmgrnode*);	Queries all the information for a node.
`RemoveFileType`	void	(FMgrFileTypeCPtr *fmgrfiletype*)	Removes a file type.
`TryCopyDir`	BoolEnum	(CStr *orname*, CStr *destname*);	Non-asserting versions of the copy, create, delete, and move functions.
`TryCopyFile`	BoolEnum	(CStr *orname*, CStr *destname*);	Non-asserting versions of the copy, create, delete, and move functions.
`TryCopyNode`	BoolEnum	(CStr *orname*, CStr *destname*);	Non-asserting versions of the copy, create, delete, and move functions.
`TryCreateDir`	BoolEnum	(CStr *name*, FMgrCreateDirPtr createInfo);	Non-asserting versions of the copy, create, delete, and move functions.
`TryCreateFile`	BoolEnum	(CStr *orname*, FMgrCreateFile *destname*);	Non-asserting versions of the copy, create, delete, and move functions.
`TryDeleteDirContent`	BoolEnum	(CStr *name*, FMgrCreateDirPtr *createInfo*);	Non-asserting versions of the copy, create, delete, and move functions.
`TryDeleteDir`	BoolEnum	(CStr *name*);	Non-asserting versions of the copy, create, delete, and move functions.
`TryDeleteFile`	BoolEnum	(CStr *name*);	Non-asserting versions of the copy, create, delete, and move functions.
`TryDeleteNode`	BoolEnum	(CStr *name*);	Non-asserting versions of the copy, create, delete, and move functions.

FMGR Class

`TryMoveDir`	BoolEnum	(CStr *orname*, CStr *dest-name*);	Non-asserting versions of the copy, create, delete, and move functions.
`TryMoveFile`	BoolEnum	(CStr *orname*, CStr *dest-name*);	Non-asserting versions of the copy, create, delete, and move functions.
`TryMoveNode`	BoolEnum	(CStr *orname*, CStr *destname*);	Non-asserting versions of the copy, create, delete, and move functions.

FNAME_ Class

Function	Returns	Arguments	Description
Cmp	CmpEnum	(CStr *name*, CStr *name*);	Compare two file names.
Convert	void	(CStr *source*, FNameBuf *dest*);	Determine the syntax of a name and convert it to the current syntax.
ConvertFromTo	void	(CStr *source*, FNameBuf *dest*, FNameStxEnum *syntax*);	Convert a name from one given syntax to another.
ConvertInPlace	void	(FNameBuf *name*);	Determine the syntax of a name and convert it to the current syntax.
CurDirStr	CStr	(void);	Returns the string representation of the current directory.
CvtDirFileToPath	void	(CStr *path*, FNameBuf *file*);	Convert a directory string from path syntax to file syntax.
CvtDirPathToFile	void	(CStr *file*, FNameBuf *path*);	Convert a directory string from file syntax to path syntax.
CvtToAbsolute	void	(CStr *name*, FNameBuf *name*);	Convert a file name to an absolute file name.
QueryParent Dir	void	(CStr *dir*, FNameBuf *parent*);	Returns the string representation of the specified parent directory.
Equal	BoolEnum	(CStr *name*, CStr *name*);	Compare two file names.
Evaluate	void	(FNameBuf *name*)	Replace each variable expression by its value, using the specified syntax.
EvaluateIn	void	(FNameBuf *name*, FNameStxEnum *syntax*)	Replace each variable expression by its value, using the current syntax.
FindSyntax	FNameStxEnum	(CStr *name*);	Returns the most likely syntax for the given name.
GetCompSet	FNameCompSet	(CStr *name*);	Return the set of components which are present in a file name.
GetCurSyntax	FNameStxEnum	(void);	Determine the current syntax.
GetStatus	FNameStatusEnum	(void);	Get the status of the most recent conversion.

GetSysSyntax	FNameStxE-num	(void);	Determine the syntax of the native system.
GetTmpPath	CStr	(void);	Return the path where temporary file names are created.
HomeDirStr	CStr	(void);	Returns the string representation of the top directory of the system.
IsAbsolute	BoolEnum	**(**CStr *name***);**	Determine if a file name is specified as absolute or as relative.
IsConvertible	BoolEnum	**(**CStr *name***);**	Determine if a name can be completely converted.
IsDirAsFile	BoolEnum	(CStr *directory*);	Determine if a directory is specified as a directory name or as a file name.
IsPortable	BoolEnum	**(**CStr *name***);**	Determine if a name can be completely converted to all target syntaxes.
IsTopDir	BoolEnum	(CStr *name*);	Returns whether a directory path name is at the top level.
IsValid	BoolEnum	(CStr *name*);	Determine whether a file name is valid in the current syntax.
IsValidIn	BoolEnum	(CStr *filename*, FNameStxEnum *syntax*);	Determine whether a file name is valid in a given file system syntax.
MakeBackupName	void	(CStr *filename*, FNameBuf *backup-name*);	Generate a name for a backup file.
MakeTmpFileName	void	(CStr *prefix*, FNameBuf *buffer*);	Generate a temporary file name.
MakeValid	void	(FNameBuf *name*);	Modify a name to make it valid in the current syntax
MakeValidIn	void	(FNameBuf *name*, FNameStxE-num *syntax*);	Modify a name to make it valid in a specified syntax
MergeFile	void	(CStr *path*, CStr *file*, FNameBuf *name*);	Merge a path component and a file component into a full file name.
MergePath	void	(CStr *path*, CStr *subdirectory*, FNameBuf *name*);	Merge a path and a subdirectory into a full path for the subdirectory.
ParentDirStr	CStr	(void);	Returns the string representation of the parent directory.

FNAME Class

QueryComps	void	(CStr *name*, FNameCompSet *components*, Str *str*);	Extract specified file name components and copy to a string.
QueryCurDir	void	*dir*(FNameBuf *dir*);	Returns the string representation of the current directory.
QueryCurParams	void	(FNameParamsPtr *params*);	Determines the current syntax conversion parameters.
QueryCurVolume	void	(FNameBuf *volume*);	Determines the current volume string.
QueryHomeDir	void	(FNameBuf *home*);	Determines the home directory of the current user.
QueryParent Dir	void	(FNameBuf *parent*);	Returns the parent directory of the current directory.
QueryTopDir	void	(FNameBuf *topDir*);	Determines the current top directory.
ReduceComps	void	(Str *name*, FNameCompSet *components*);	Reduce a file name to a specified set of components.
ResetCurParams	void	(void);	Reset the current syntax conversion parameters to the default parameters.
SetCurDir	void	(CStr *directory*);	Returns the string representation of the current directory.
SetCurParams	void	(FNameParamsCPtr *params*);	Set the current syntax conversion parameters.
SetCurSyntax	void	(FNameStxEnum *syntax*);	Set a particular syntax as the current syntax.
SetStatus	void	(FNameStatusEnum *status*);	Set the status flag to the given value.
SetTmpPath	void	(CStr *path*);	Change the path where temporary file names will be created.
SplitFile	void	(CStr *name*, FNameBuf *path*, FNameBuf *file*);	Split a file name into path and file components.
SplitPath	BoolEnum	(CStr *name*, FNameBuf *path*, FNameBuf *subdirectory*);	Split a path into parent path and child subdirectory components.
StatusGetMsg	CStr	(FNameStatusEnum *status*);	Get the text description of the given status value.

FNAME Class

StxGetName	CStr	(FNameStxEnum *syntax*);	Returns the name of the specified syntax.
SysTmpPath	CStr	(void);	Return the native system's default path for temporary files.
TopDirStr	CStr	(void);	Returns the string representation of the top directory of the current volume.
VolumeQueryCurDir	void	(CStr *volume*, CStr *directory*);	Query the current directory of a given volume.
VolumeSetCurDir	void	(CStr *volume*, CStr *directory*);	Set the current directory of a given volume.

HASH_ Class

Function	Returns	Arguments	Descriptions
`AddGetEntry`	HashEntryPtr	(HashPtr *hash*, HashKeyVal *key*, HashDataVal *value*);	Adds an entry corresponding to 'key' without checking its previous existence.
`CompareProc`	BoolEnum	(HashKeyVal *key1*, HashKeyVal *key2*);	Procedure that compares two entries.
`DataCloneProc`	HashDataVal	(HashDataVal);	Procedure returning a clone of the data of entry.
`DataDisposeProc`	void	(HashDataVal);	Procedure used to dispose data stored in the hash table for an entry (created through cloning).
`DefCompareInt`	BoolEnum	(HashKeyVal *key1*, HashKeyVal *key2*);	Returns the result of the default Open Interface comparison of key1 and key2.
`DefCompareIStr`	BoolEnum	(HashKeyVal *key1*, HashKeyVal *key2*);	Returns the result of the default Open Interface comparison of key1 and key2.
`DefComparePtr`	BoolEnum	(HashKeyVal *key1*, HashKeyVal *key2*);	Returns the result of the default Open Interface comparison of key1 and key2.
`DefCompareStr`	BoolEnum	(HashKeyVal *key1*, HashKeyVal *key2*);	Returns the result of the default Open Interface comparison of key1 and key2.
`DefHashInt`	HashLenVal	(HashKeyVal *key*, HashLenVal *mod*);	Returns the bin index computed by the default hashing procedure provided by Open Interface.
`DefHashIStr`	HashLenVal	(HashKeyVal *key*, HashLenVal *mod*);	Returns the bin index computed by the default hashing procedure provided by Open Interface.
`DefHashPtr`	HashLenVal	(HashKeyVal *key*, HashLenVal *mod*);	Returns the bin index computed by the default hashing procedure provided by Open Interface.
`DefHashStr`	HashLenVal	(HashKeyVal *key*, HashLenVal *mod*);	Returns the bin index computed by the default hashing procedure provided by Open Interface.

Function	**Returns**	**Arguments**	**Descriptions**
`DefStrKeyClone`	HashKeyVal	(HashKeyVal *key*);	Default string cloning
`DefStrKeyDispose`	void	(HashKeyVal *key*);	Default string cloning
`EntryGetKey`	HashKeyVal	(HashEntryCPtr *entry*);	Returns the key stored in the entry.
`EntryGetValue`	HashDataVal	(HashKeyCPtr *key*);	Returns the value stored in the entry.
`EntrySetValue`	void	(HashEntryPtr *entry*, HashDataVal *val-ue*);	Changes the value stored in the entry.
`Extract`	BoolEnum	(HashPtr *hash*, HashKeyVal *key*, Hash-DataValPtr *valPtr*);	Looks for an entry corre-sponding to 'key'.
`GetDefIntInfo`	HashInfoCPtr	(HashInfoPtr *hashInfo*);	Returns the default set-tings for a hash table with integer keys.
`GetDefIStrInfo`	HashInfoCPtr	(void);	Returns the default set-tings for a hash table with string keys.
`GetDefPtrInfo`	HashInfoCPtr	(void);	Returns the default set-tings for a hash table with pointer keys.
`GetDefStrInfo`	HashInfoCPtr	(void);	Returns the default set-tings for a hash table with string keys.
`GetDefStrKeyClonedInfo`	HashInfoCPtr	(void);	Returns the default set-tings for a hash table with cloned string keys.
`GetEntry`	HashEntryPtr	(HashPtr *hash*, HashKeyVal *key*);	Returns the pointer to the actual entry corre-sponding to 'key'. I
`Insert`	void	(HashPtr *hash*, HashKeyVal *key*, Hash-DataVal *value*);	Looks for an entry corre-sponding to 'key'.
`InsertGetEntry`	HashEntryPtr	(HashPtr *hash*, HashKeyVal *key*, Hash-DataVal *value*);	Same as above but tests whether there was an en-try there before or not.
`Lookup`	BoolEnum	(HashCPtr *hash*, HashKeyVal *key*, Hash-DataValPtr *valPtr*);	Looks for an entry corre-sponding to 'key'
`Perf`	PerfEnum	(HashCPtr *hash*, HashPerfProc *perfProc*, ClientPtr *data*);	Triggers the iteration of 'perfProc' for all the en-tries in the table.
`QueryDefInfo`	void	(HashInfoPtr *hashInfo*);	Fills 'hashInfo' with the default settings for a hash table.
`QueryInfo`	void	(HashCPtr *hashc*, HashInfoPtr *hashInfo*);	Fills 'hashInfo' with the values that were used to define the hash table.

Function	Returns	Arguments	Descriptions
`QueryStats`	void	(HashCPtr *hash*, HashStatsInfoPtr *stats*);	Fills 'stats' with the statistical information corresponding to hash.
`RemoveEntry`	void	(HashPtr *hash*, HashEntryPtr *entry*);	Removes the entry from the hash table.
`Reset`	void	(HashPtr *hash*);	Resets the contents of the hash table to the defaults as when created.

HEAP_ Class

Function	Returns	Arguments	Description
`Add`	void	(HeapPtr *heap*, HeapKeyVal *key*, ClientPtr *client*);	Insertion with no reorder: the heap structure is temporarily incorrect, a call to HEAP_Correct will be necessary before the heap can actually be used.
`Alloc`	HeapPtr	(void);	Returns a pointer to an allocated but not yet constructed heap.
`Correct`	void	(HeapPtr *heap*);	The structure of the key is corrected.
`Dealloc`	void	(HeapPtr *heap*);	Deallocates the heap.
`Dispose`	void	(HeapPtr *heap*);	Disposes a heap.
`GetSize`	HeapIndexVal	(HeapPtr *heap*);	Returns the number of entries in the heap.
`Insert`	void	(HeapPtr *heap*, HeapKeyVal *key*, ClientPtr *client*);	Insertion with reorder.
`New`	void	(HeapPtr *heap*);	Creates and constructs a heap.
`Perf`	PerfEnum	(HeapPtr *heap*, HeapPerfProc *proc*, ClientPtr *clientData*);	Performs `proc' on each entry of the heap. `proc' gets called with `clientData' as last argument.
`QueryFirst`	BoolEnum	(HeapPtr *heap*, BoolEnum *extract*, HeapKeyValPtr *keyPtr*, ClientPtrPtr *dataPtr*);	Extraction of the top-most entry: if it can find one, it returns BOOL_TRUE and sets `keyPtr' and `dataPtr'; if not it returns BOOL_FALSE.

ISET_ Class

Function	Returns	Arguments	Description
`AddIntervals`	void	(ISetPtr *iset*, ISetLenVal *n*, ISetIntervalPtr *intervals*);	Same as ISET_AddInterval but for 'n' intervals.
`ContainsElt`	BoolEnum	(ISetPtr *iset*, ISetEltVal *elt*);	Returns BOOL_TRUE if the set contains 'elt'.
`ContainsIntervals`	BoolEnum	(ISetPtr *iset*, ISetLenVal *numIntervals*, ISetIntervalPtr *intervals*);	Same as ISET_ContainsInterval but for 'n' intervals.
`GetMaxElt`	ISetEltVal	(ISetPtr *iset*);	Returns the biggest element in the set.
`GetMinElt`	ISetEltVal	(ISetPtr *iset*);	Returns the smallest element in the set.
`GetNumIntervals`	ISetLenVal	(ISetPtr *iset*);	Returns the number of intervals in the set.
`IsAll`	BoolEnum	(ISetPtr *iset*);	Returns BOOL_TRUE if the set contains all possible elements, i.e. is the interval [ISET_ELTVALINF, ISET_ELTVALSUP].
`MixGetPartSet`	ISetMixPartSet	(ISetPtr *A*, ISetPtr *B*);	Same as the equivalent calls in the Set package, but for ISet objects.
`MixQueryParts`	void	(ISetPtr A, ISetPtr *B*, ISetMixPartSet *part*, ISetPtr *C*);	Same as the equivalent calls in the Set package, but for ISet objects.
`QueryComplement`	void	(ISetPtr *iset*, ISetPtr *compl*);	Computes the complement of 'iset' and puts the result into 'compl'.
`QueryIntervals`	void	(ISetPtr *iset*, ISetLenVal *numIntervals*,ISetIntervalPtr *intervals*);	Fills intervals with the intervals in the iset.
`RemoveIntervals`	void	(ISetPtr *iset*, ISetLenVal *numIntervals*, ISetIntervalPtr *intervals*);	Same as ISET_RemoveInterval but for 'n'intervals.
`SetIntervals`	void	(ISetPtr *iset*, ISetLenVal *numIntervals*, ISetIntervalPtr *intervals*);	Defines the intervals in the iset to be those specified by intervals.
`Universal Set`	ISetPtr	(void);	Returns a pointer to a shared "universal" set (i.e. a set which contains all possible values).

NFIER_ Class

Function	Returns	Arguments	Description
`Alloc`	NfierPtr	(void);	Returns a pointer to an allocated but not yet constructed notifier.
`Broadcast`	void	(NfierCPtr *nfier*, ClientPtr *clientData*);	Broadcasts a notification (`clientData' contains the notification information) to all the clients which have registered.
`ClientDealloc`	void	(NfierClientPtr *nfier*);	Deallocates the notifier client.
`ClientGetClientData`	ClientPtr	(NfierPtr *nfier*);	Retrieves the client data previously associated with `nfierClient'.
`ClientNewRegister`	NfierClientPtr	(NfierPtr *nfier*, NfierClientPtr *client*);	Unregisters, destructs and deallocates a notifier client from a notifier..
`ClientSetClientData`	void	(NfierClientPtr *nfierClient*, ClientPtr *clientData*);	Associates `clientData' with `nfierClient'.
`ClientUnregisterDispose`	void	(NfierPtr *nfier*, NfierClientPtr *client*);	Allocates, constructs with `proc', registers with the notifier a notifier client that is returned..
`RegisterNfierClient`	void	(NfierPtr *nfier*, NfierClientPtr *client*);	Adds a notifier client to the list of clients of a given notifier.
`UnregisterNfierClient`	void	(NfierPtr *nfier*, NfierClientPtr *client*);	Removes a notifier client from the list of clients of a notifier.

PACK_ Class

Function	Returns	Arguments	Description
`Alloc`	PackPtr	(void);	Allocates the pack object.
`CcittDecode`	void	(PackPtr *pack*, PackSizeVal *width*, PackCcittFlags *ccittFlags*);	Decodes using the CCITT compression mechanisms.
`CcittEncode`	void	(PackPtr *pack*, PackSizeVal *width*, PackCcittFlags *ccittFlags*);	Encodes using the CCITT compression mechanisms.
`Dealloc`	void	(PackPtr *pack*);	Deallocates the pack object.
`Decode`	void	(PackPtr *pack*, PackMethodEnum *method*);	Decodes with the method specified by `method'.
`Encode`	void	(PackPtr *pack*, PackMethodEnum *method*);	Encodes with the method specified by `method'.
`LzwDecode`	void	(PackPtr *pack*, PackDepthVal *depth*, PackLzwFlags *flags*);	Decodes with LZW.
`LzwEncode`	void	(PackPtr *pack*, PackDepthVal *depth*, PackLzwFlags *flags*);	Encodes with LZW.
`PkbDecode`	void	(PackPtr *pack*);	Decodes with PackBits.
`PkbEncode`	void	(PackPtr *pack*);	Encodes with PackBits.
`RleDecode`	void	(PackPtr *pack*);	Decodes with RLE algorithm.
`RleEncode`	void	(PackPtr *pack*);	Encodes with RLE algorithm.

PACK Class

POOL_ Class

Function	Returns	Arguments	Description
`Alloc`	PoolPtr	(void);	Returns a pointer to an allocated but not yet constructed memory pool.
`Dealloc`	void	(PoolPtr *pool*);	Deallocates the memory pool.
`DisposePtr`	void	(PoolPtr *pool*, HugePtr *ptr*);	Deallocates ptr. ptr must have been allocated in the pool
`NDPool`	void	(PoolPtr *pool*);	Default memory pool construction.
`NewPtr`	HugePtr	(PoolPtr *pool*);	Returns a pointer to a cell allocated in the pool.
`QueryInfo`	void	(PoolCPtr *pool*, PoolInfoPtr *poolInfo*);	Fills poolInfo with the information with which the memory pool was created.
`QueryStats`	void	(PoolCPtr *pool*, PoolStatsInfoPtr *stats*);	Fills stats with the statistics information on the pool.
`ResetStats`	void	(PoolStatsInfoPtr *stats*);	Resets stats.
`SetInfo`	void	(PoolPtr *pool*, PoolInfoCPtr *poolInfo*);	Updates the memory pool with the information from poolInfo.

PTR_ Class

Function	Returns	Arguments	Description
`AlignCheck`	void	(HugeCPtr *ptr*);	Checks whether the pointer is aligned for the worst case scenario.
`Clear`	void	(VoidPtr *ptr*, PtrSizeVal *number*);	Sets the set of bytes specified in a buffer to zero.
`Cmp`	CmpEnum	(VoidCPtr *p1*, VoidCPtr *p2*, PtrSizeVal *size*);	Compares the `size' first bytes of `p1' and `p2'. The bytes are compared as their unsigned values bytes.
`Copy`	void	(VoidPtr *dest*, VoidCPtr *src*, PtrSizeVal *number*);	Copies the number of bytes specified to a disjoint location in a buffer.
`CopyByte`	void	(VoidPtr *dst*, PtrSizeVal *dstOffset*, VoidCPtr *src*, PtrSizeVal *srcOffset*);	Copies value stored at 'src'+'srcOffset' to dst+'dstOffset'
`DefFailProc`	HugePtr	(PtrFailEnum *fail*, PtrHugeSizeVal *size*);	Default method to trap memory management failures.
`Dispose`	void	(void);	Frees memory allocated for a given pointer.
`GetAlignedSize`	PtrSizeVal	(PtrSizeVal *size*);	Returns the smallest aligned size for the size passed.
`GetByte`	Byte	(VoidCPtr *ptr*, PtrSizeVal *offset*);	Reads value stored at address ptr+'offset'.
`GetFailProc`	PtrFailProc	(void);	Returns the custom failure callback procedure installed.
`GetSize`	PtrSizeVal	(void);	Returns the size of which the buffer 'ptr' is allocated.
`HugeClear`	void	(HugePtr *hugeptr*, PtrHugeSizeVal *number*);	Sets the set of bytes specified in a buffer to zero.
`HugeCmp`	CmpEnum	(HugeCPtr *p1*, HugeCPtr *p2*, PtrHugeSizeVal *size*);	Compares the `size' first bytes of `p1' and `p2'. The bytes are compared as their unsigned values bytes.
`HugeCopy`	void	(HugePtr *dest*, HugeCPtr *src*, PtrHugeSizeVal *number*);	Copies the number of bytes specified to a disjoint location in a buffer.
`HugeCopyByte`	void	(HugePtr *dst*, PtrHugeSizeVal *dstOffset*, HugeCPtr *src*,PtrHugeSizeVal *srcOffset*);	Copies value stored at 'src'+'srcOffset' to dst+'dstOffset'.
`HugeDispose`	void	(HugePtr *hugeptr*);	Frees memory allocated for a given pointer.
`HugeGetByte`	Byte	(HugeCPtr *ptr*, PtrHugeSizeVal *offset*);	Reads value stored at address ptr+'offset'.
`HugeGetSize`	PtrHugeSizeVal	(HugePtr *ptr*);	Same as before except that the sizes can be larger than HUGELIMIT.
`HugeMatches`	BoolEnum	(HugeCPtr *p1*, HugeCPtr *p2*, PtrHugeSizeVal *size*);	Returns whether `p1' and `p2' match exactly on their first `size'.

Function	Returns	Arguments	Description
HugeMove	void	(HugePtr *dest*, HugeCPtr *src*);	Copies the number of bytes specified to an overlapping location.
HugeNew	HugePtr	(PtrHugeSizeVal *size*);	Allocates a new huge pointer of the size specified.
HugeSet	void	(HugePtr *ptr*, Byte *byte*, PtrHugeSizeVal *size*);	Sets the specified number of bytes of a buffer to a value specified.
HugeSetByte	void	(HugePtr *ptr*, PtrHugeSizeVal *offset*, Byte *byte*);	Sets new value at address ptr+'offset' to 'byte'.
HugeSetSize	HugePtr	(HugePtr *ptr*, PtrHugeSizeVal *size*);	Changes the size of the pointer specified.
HugeSwap	void	(HugePtr *p1*, HugePtr *p2*, PtrHugeSizeVal *size*);	Copies `size' first bytes of `p1' to `p2' and vice versa.`p1' and `p2' must NOT overlap.
HugeSwapByte	void	(HugePtr *p1*, PtrHugeSizeVal *offset1*, HugePtr *p2*, PtrHugeSizeVal *offset2*);	Exchanges values at ptr1+'offset1' and 'ptr2'+'offset2'.
Int16ToMch	void	(Int16Ptr *valp*);	Converts in place an integer from standard format into the machine-dependent format.
Int16ToStd	void	(Int16Ptr *valp*);	Converts in place an integer from the machine-dependent format to the standard format.
Int32ToMch	void	(Int32Ptr *valp*);	Converts in place an integer from standard format into the machine-dependent format
Int32ToStd	void	(Int32Ptr *valp*);	Converts in place an integer from the machine-dependent format to the standard format.
Int8ToMch	void	(Int8Ptr *valp*);	Converts in place an integer from standard format into the machine-dependent format
Int8ToStd	void	(Int8Ptr *valp*);	Converts in place an integer from the machine-dependent format to the standard format.
Matches	BoolEnum	(VoidCPtr *p1*, VoidCPtr *p2*, PtrSizeVal *size*);	Returns whether `p1' and `p2' match exactly on their first `size'.
Move	void	(VoidPtr *dest*, VoidCPtr *src*, PtrSizeVal *number*); (HugePtr *dest*, HugeCPtr *src*);	Copies the number of bytes specified to an overlapping location.
New	VoidPtr	(PtrSizeVal *size*);	Allocates a new pointer of the size specified or for named structure.
QueryStats	void	(PtrStatsPtr *ptr*);	Determines the current memory manager statistics.
ReadInt16	void	(VoidCPtr *ptr*, Int16Ptr *valp*);	Reads a machine-dependent integer from a memory buffer 'ptr' where integers are in standard format.

Function	Returns	Arguments	Description
`ReadInt32`	void	(VoidCPtr *ptr*, Int32Ptr *valp*);	Reads a machine-dependent integer from a memory buffer ‘ptr’ where integers are in standard format.
`ReadInt8`	void	(VoidCPtr *ptr*, Int8Ptr *valp*);	Reads a machine-dependent integer from a memory buffer ‘ptr’ where integers are in standard format.
`ReadStr`	void	(HugeCPtr *ptr*, Str *str*, StrIVal *len*);	Reads a machine-dependent string from a memory buffer where strings are stored in standard format.
`Set`	void	(VoidPtr *dest*, Byte *byte*, PtrSizeVal *number*);	Sets the specified number of bytes of a buffer to a value specified.
`SetByte`	void	(VoidPtr *ptr*, PtrSizeVal *offset*, Byte *byte*);	Sets new value at address ptr+’offset’ to ‘byte’.
`SetFailProc`	void	(PtrFailProc *failProc*);	Sets a custom failure callback procedure.
`SetSize`	VoidPtr	(VoidPtr *ptr*, PtrSizeVal *size*);	Changes the size of the pointer specified.
`StatsOutput`	void	(void);	Outputs memory manager statistics to standard output.
`StrToMch`	void	(Str *str*, StrIVal *len*);	Converts strings to and from standard format.
`StrToStd`	void	(Str *str*, StrIVal *len*);	Converts strings to and from standard format.
`Swap`	void	(VoidPtr *p1*, VoidPtr *p2*, PtrSizeVal *size*);	Copies `size' first bytes of `p1' to `p2' and vice versa.`p1' and `p2' must NOT overlap.
`SwapByte`	void	(VoidPtr *p1*, PtrSizeVal *offset1*, VoidPtr *p2*, PtrSizeVal *offset2*);	Exchanges values at ptr1+’offset1’ and ‘ptr’+’offset’
`WriteInt16`	void	(VoidPtr *ptr*, Int16CPtr *valp*);	Writes a machine-dependent integer into a memory buffer ‘ptr’ where integers are in standard format.
`WriteInt32`	void	(VoidPtr *ptr*, Int3CPtr *valp*);	Writes a machine-dependent integer into a memory buffer ‘ptr’ where integers are in standard format.
`WriteInt8`	void	(VoidPtr *ptr*, Int8CPtr *valp*);	Writes a machine-dependent integer into a memory buffer ‘ptr’ where integers are in standard format.
`WriteStr`	void	(HugeCPtr *ptr*, Str *str*, StrIVal *len*);	Writes a machine-dependent string into a memory buffer where strings are stored in standard format.

RCLAS_ Class

Function	Returns	Arguments	Description
`Add`	void	(RClasPtr *rclas*, ResNfyProc *nfyProc*);	Registers the class and its default notification procedure with the resource manager.
`CPlusRegister`	RClasPtr	(CStr *name*, RClasNewProc *nProc*, RClasDeleteProc *dProc*, ResNfyProc *nfy*, RClasPtr *pClass*, PFldPtr *oiFields*);	Registration of a C++ subclass to the resource manager.
`FindByName`	RClasPtr	(CStr *name*);	Returns a class by name.
`GetDefNfy`	ResNfyProc	(RClasCPtr *rclas*);	Get the default notification handler of an RClas.
`GetFields`	PFldPtr	(RClasCPtr *rclas*);	Functions that get the various fields of an RClas structure.
`GetFirst`	RClasPtr	(void);	Returns the first alphabetical resource class.
`GetFlags`	RClasFlagsSet	(RClasCPtr *rclas*);	Functions that get the various fields of an RClas structure.
`GetModName`	CStr	(RClasCPtr *rclas*);	Functions that get the various fields of an RClas structure.
`GetName`	CStr	(RClasCPtr *rclas*);	Functions that get the various fields of an RClas structure.
`GetNext`	RClasPtr	(RClasPtr *rclas*);	Returns the class that is alphabetically after the class specified.
`GetParentClass`	RClasPtr	(RClasCPtr *rclas*);	Functions that get the various fields of an RClas structure.
`GetSizeOfRes`	PtrSizeVal	(RClasCPtr *rclas*);	Functions that get the various fields of an RClas structure.
`GetTemplate`	ResPtr	(RClasPtr *rclas*);	Functions that get the various fields of an RClas structure.
`GetVersion`	RClasVersionVal	(RClasCPtr *rclas*);	Functions that get the various fields of an RClas structure.
`IsSubClassOf`	BoolEnum	(RClasPtr *childclass*, RClasCPtr *parentclass*);	Determines whether one class is a subclass of another.
`OperatorDelete`	void	(RClasPtr *rclas*, VoidPtr *obj*);	Default allocation method for all Open Interface classes
`OperatorNew`	VoidPtr	RClasPtr *rclas*, PtrSizeVal *size*);	Default deallocation method for all Open Interface classes.
`ProcessDefNfy`	void	(RClasCPtr *rclas*, ResPtr *res*, ResNfyEnum *code*);	Trigger the default notification procedure on an instance.
`ProcessParentDefNfy`	void	RClasCPtr *rclas*, ResPtr *res*, ResNfyEnum *code*);	Trigger the parent default notification procedure on an instance.

`Register`	void	(RClasPtr rclas);	Registers the class and its default notification procedure with the resource manager.
`SetDefNfy`	void	(RClasPtr *rclas*, ResNfyProc *defnfy*);	Set the default notification handler for an RClass.
`SetFields`	void	(RClasPtr *rclas*, PFldPtr *fields*);	Functions that set the various fields of an RClas structure.
`SetFlags`	void	(RClasPtr *rclas*, RClasFlags-Set *flags*);	Functions that set the various fields of an RClas structure.
`SetModName`	void	(RClasPtr *rclas*, CStr *mod-name*);	Functions that set the various fields of an RClas structure.
`SetName`	void	(RClasPtr *rclas*, CStr *name*);	Functions that set the various fields of an RClas structure.
`SetParentClass`	void	(RClasPtr *rclas*, RClasPtr *parent*);	Functions that set the various fields of an RClas structure.
`SetSizeOfRes`	void	RClasPtr *rclas*, PtrSizeVal *size*);	Functions that set the various fields of an RClas structure.
`SetVersion`	void	(RClasPtr *rclas*, RClasVer-sionVal *version*);	Functions that set the various fields of an RClas structure.

RECT16_ Class

Function	Returns	Arguments	Description
`AbsDist`	Int16	(Point16Ptr *p1*, Point16Ptr *p2*);	Returns the absolute distance between two points.
`ContainsPoint`	void	(Rect16CPtr *r*, Point16CPtr *p*);	Determines whether a rectangle contains the point specified.
`Copy`	void	(Rect16Ptr *dst*, Rect16CPtr *src*);	Copies a rectangle.
`CopyResetOri`	void	*src*(Rect16Ptr *dst*, Rect16CPtr *src*);	Copies ‘src’ into ‘dst’, but then sets dst->Ori to (0,0).
`Equals`	BoolEnum	(Rect16CPtr *r1*, Rect16CPtr *r2*);	Returns BOOL_TRUE if ‘r1’ and ‘r2’ are identical.
`GetBegX`	Int16	(Rect16CPtr *r*);	Gets one Origin / Extent coordinate.
`GetBegY`	Int16	(Rect16CPtr *r*);	Gets one Origin / Extent coordinate.
`GetEndX`	Int16	(Rect16CPtr *r*);	Gets one Origin / Extent coordinate.
`GetEndY`	Int16	(Rect16CPtr *r*);	Gets one Origin / Extent coordinate.
`GetExtX`	Int16	(Rect16CPtr *r*);	Gets one Origin / Extent coordinate.
`GetExtY`	Int16	(Rect16CPtr *r*);	Gets one Origin / Extent coordinate.
`GetOriX`	Int16	(Rect16CPtr *r*);	Gets one Origin / Extent coordinate.
`GetOriY`	Int16	(Rect16CPtr *r*);	Gets one Origin / Extent coordinate.
`IncludesNonEmptyRect`	BoolEnum	(Rect16CPtr *r2*, Rect16CPtr *r1*);	Returns BOOL_TRUE if ‘r1’ is included in ‘r2’ (assuming ‘r1’ is not empty).
`IncludesRect`	BoolEnum	(Rect16CPtr *r2*, Rect16CPtr *r1*);	Determines whether a rectangle contains the rectangle specified.
`IncOriExtXY`	void	(Int16*orix*,Int16*oriy*,Int16*extx*,Int16*exty*);	Increments all Origin / Extent coordinates.
`Intersects`	BoolEnum	(Rect16CPtr *r1*, Rect16CPtr *r2*);	Determines whether two rectangles intersect.
`Intersection`	void	;(Rect16Ptr *dst*, Rect16CPtr *src*);	Sets ‘dst’ to the insersection of ‘dst’ and ‘src’.
`IsEmpty`	BoolEnum	(Rect16CPtr *r*);	Determines whether a point is empty.
`IsValid`	BoolEnum	(Rect16CPtr *r*);	Determines whether a rectangle has valid coordinates.
`MakeFit`	void	(Rect16Ptr *in*, Rect16CPtr *out*);	Repositions a rectangle so that it is contained within the rectangle specified.
`MakeValid`	BoolEnum	(Rect16CPtr *r*);	Changes the coordinates of the rectangle specified to make them valid.
`MoveInside`	void	(Rect16Ptr *in*, Rect16CPtr *out*);	Moves one rectangle inside another.

Function	**Returns**	**Arguments**	**Description**
`Reset`	void	(Rect16CPtr *r*);	Resets the coordinates of a rectangle to 0.
`SetByPoints`	void	(Rect16Ptr *r*, Point16CPtr *beg*, Point16CPtr *end*);	Sets the coordinates of a rectangle to the points specified.
`SetBegX`	void	(Rect16Ptr *r*, Int16 *val*);	Sets one Origin / Extent coordinate.
`SetBegY`	void	(Rect16Ptr *r*, Int16 *val*);	Sets one Origin / Extent coordinate.
`SetEndX`	void	(Rect16Ptr *r*, Int16 *val*);	Sets one Origin / Extent coordinate.
`SetEndY`	void	(Rect16Ptr *r*, Int16 *val*);	Sets one Origin / Extent coordinate.
`SetExtX`	void	(Rect16Ptr *r*, Int16 *val*);	Sets all Origin / Extent coordinates.
`SetExtY`	void	(Rect16Ptr *r*, Int16 *val*);	Sets all Origin / Extent coordinates.
`SetOriExtXY`	void	(Rect16Ptr *r*, Int16 *orix*, Int16 *oriy*, Int16 *extx*, Int16 *exty*);	Sets all Origin / Extent coordinates.
`SetOriX`	void	(Rect16Ptr *r*, Int16 *val*);	Sets all Origin / Extent coordinates.
`SetOriY`	void	(Rect16Ptr *r*, Int16 *val*);	Sets all Origin / Extent coordinates.
`Union`	void	(Rect16Ptr *dst*, Rect16CPtr *src*);	Determines the union of two rectangles.

RECT32_ Class

Function	Returns	Arguments	Description
`AbsDist`	Int32	(Point32Ptr *p1*, Point32Ptr *p2*);	Returns the absolute distance between two p
`ContainsPoint`	void	(Rect32CPtr *r*, Point32CPtr *p*);	Determines whether a rectangle contains the
`Copy`	void	(Rect32Ptr *dst*, Rect32CPtr *src*);	Copies a rectangle.
`CopyResetOri`	void	*src*(Rect32Ptr *dst*, Rect32CPtr *src*);	Copies ‘src’ into ‘dst’, but then sets dst->Ori
`Equals`	BoolEnum	(Rect32CPtr *r1*, Rect32CPtr *r2*);	Returns BOOL_TRUE if ‘r1’ and ‘r2’ are ider
`GetBegX`	Int32	(Rect32CPtr *r*);	Gets one Origin / Extent coordinate.
`GetBegY`	Int32	(Rect32CPtr *r*);	Gets one Origin / Extent coordinate.
`GetEndX`	Int32	(Rect32CPtr *r*);	Gets one Origin / Extent coordinate.
`GetEndY`	Int32	(Rect32CPtr *r*);	Gets one Origin / Extent coordinate.
`GetExtX`	Int32	(Rect32CPtr *r*);	Gets one Origin / Extent coordinate.
`GetExtY`	Int32	(Rect32CPtr *r*);	Gets one Origin / Extent coordinate.
`GetOriX`	Int32	(Rect32CPtr *r*);	Gets one Origin / Extent coordinate.
`GetOriY`	Int32	(Rect32CPtr *r*);	Gets one Origin / Extent coordinate.
`IncludesNonEmptyRect`	BoolEnum	(Rect32CPtr *r2*, Rect32CPtr *r1*);	Returns BOOL_TRUE if ‘r1’ is included in ‘r empty).
`IncludesRect`	BoolEnum	(Rect32CPtr *r2*, Rect32CPtr *r1*);	Determines whether a rectangle contains the
`IncOriExtXY`	void	(Int32*orix*, Int32*oriy*, Int32*extx*, Int32*exty*);	Increments all Origin / Extent coordinates.
`Intersects`	BoolEnum	(Rect32CPtr *r1*, Rect32CPtr *r2*);	Determines whether two rectangles intersec
`Intersection`	void	;(Rect32Ptr *dst*, Rect32CPtr *src*);	Sets ‘dst’ to the insersection of ‘dst’ and ‘src’
`IsEmpty`	BoolEnum	(Rect32CPtr *r*);	Determines whether a point is empty.
`IsValid`	BoolEnum	(Rect32CPtr *r*);	Determines whether a rectangle has valid co
`MakeFit`	void	(Rect32Ptr *in*, Rect32CPtr *out*);	Repositions a rectangle so that it is containe specified.
`MakeValid`	BoolEnum	(Rect32CPtr *r*);	Changes the coordinates of the rectangle spe
`MoveInside`	void	(Rect32Ptr *in*, Rect32CPtr *out*);	Moves one rectangle inside another.
`Reset`	void	(Rect32CPtr *r*);	Resets the coordinates of a rectangle to 0.
`SetByPoints`	void	(Rect32Ptr *r*, Point32CPtr *beg*, Point32CPtr *end*);	Sets the coordinates of a rectangle to the poi
`SetBegX`	void	(Rect32Ptr *r*, Int32*val*);	Sets one Origin / Extent coordinate.
`SetBegY`	void	(Rect32Ptr *r*, Int32*val*);	Sets one Origin / Extent coordinate.
`SetEndX`	void	(Rect32Ptr*r*, Int32*val*);	Sets one Origin / Extent coordinate.
`SetEndY`	void	(Rect32Ptr*r*, Int32*val*);	Sets one Origin / Extent coordinate.
`SetExtX`	void	(Rect32Ptr *r*, Int32*val*);	Sets all Origin / Extent coordinates.

Function	Returns	Arguments	Description
`SetExtY`	void	(Rect32Ptr *r*, Int32*val*);	Sets all Origin / Extent coordinates.
`SetOriExtXY`	void	(Rect32Ptr *r*, Int32*orix*, Int32*oriy*, Int32*extx*, Int32*exty*);	Sets all Origin / Extent coordinates.
`SetOriX`	void	(Rect32Ptr *r*, Int32*val*);	Sets all Origin / Extent coordinates.
`SetOriY`	void	Rect32Ptr *r*, Int32*val*);	Sets all Origin / Extent coordinates.
`Union`	void	(Rect32Ptr *dst*, Rect32CPtr *src*);	Determines the union of two rectangles.

RES_ Class

Function	Returns	Arguments	Description
`CheckClass`	void	(ResPtr *res*, RClasCPtr *class*);	Provides a way for recovering from the specification of an invalid class.
`Class`	RClasPtr	(void);	Returns a pointer to the Res class.
`ClassDefNfy`	void	(ResPtr *res*, ResNfyEnum *code*);	Trigger the default notification procedure on an instance.
`Clone`	ResPtr	(ResCPtr *sourceRes*, BoolE-num *deep*);	Creates a resource with all the persistent fields copied from ‘sourceRes’.
`CmdIssue`	void	(ResPtr *res*);	Issue the resource’s command for execution.
`CmdSend`	void	(ResPtr *start*, CmdCtlEnum *ctl*);	Start routing of the resource’s command at given ’start’ point. (start is sent a NfyCommandRoute notification with ‘ctl’ as argument).
`CmdTableHandle`	void	(ResPtr *res*, CmdTablePtr *table*);	To be used upon NfyCommand notifications: fetches the command object and searches the given command table, using the default RES_TableHandle method.
`CmdUpdate`	void	(ResPtr *res*);	Issue the resource’s command as command query for self updating.
`DefNfy`	void	(ResPtr *res*, ResNfyEnum *notif*);	Default notification handler for a resource class.
`ExecuteScript`	BoolEnum	(ResPtr *res*, ResNfyEnum *code*);	This function causes the script for the event ‘code’ to be executed, if such a script is attached to the resource.
`FilenameOutputRc`	void	(ResPtr *res*, CStr *filename*);	Outputs resource to the text file specified by filename.
`Find`	ResPtr	(CStr *modname*, CStr *resname*);	Loads a resource by class name and resource name, returning NULL if the resource does not exist.
`FindByFullName`	ResPtr	(CStr *Mod.Res*);	Loads the attached resource specified, returning NULL if the resource does not exist.
`GetClass`	RClasCPtr	(ResCPtr *res*);	Returns the class of the resource specified.
`GetClientData`	ClientPtr	(ResCPtr *res*);	Returns the client data of a resource.
`GetName`	CStr	(ResCPtr *res*);	Returns the name of a resource as a NULL-terminated string.

Function	Returns	Arguments	Description
`GetNfyCmd`	CmdPtr	(ResCPtr *res*);	Returns the CmdPtr associated to a command notification.
`GetNfyData`	ClientPt	(ResPtr *res*);	Returns the notify data of a resource.
`GetNfyHandlerClientData`	ClientPtr	(ResPtr *res*, ResNfyEnum *nfy*);	Returns the ClientData installed for 'res' to process the 'nfy' message.
`GetNfyHandlerProc`	ResNfy-Handler-Proc	(ResPtr *res*, ResNfyEnum *nfy*);	Returns the handler procedure installed for 'res' to process the 'nfy' message.
`GetNfyProc`	ResN-fyProc	(ResPtr *res*);	Returns the currently installed notification routine for a resource.
`GetNthChild`	ResPtr	(ResPtr *resource*, ArrayIVal *child*);	Returns the child resource specified for a resource.
`GetNumChildren`	ArrayIVal	(ResPtr *res*);	Returns the total number of child resources belonging to a resource.
`InheritsFrom`	BoolEnum	(ResCPtr *childclass*, RClas-CPtr *parentclass*);	Determines whether one class inherits from another class.
`IsCmdSource`	BoolEnum	(ResCPtr *res*);	Returns whether the resource has the RES_FLAGISCOMMANDSOURC E flag set.
`IsInitialized`	BoolEnum	(ResCPtr *res*);	Determines whether a resource has been initialized already.
`IsNamed`	BoolEnum	(ResCPtr *res*);	Determines whether a resource has a name or not.
`LibExit`	void	(void);	Exits the Res library, making all Open Interface libraries unavailable.
`LibInit`	void	(void);	Installs, loads, and initializes the Res library, making all Open Interface libraries available, except Genwin.
`LibInstall`	void	(void);	Installs the Res library, making all Open Interface libraries available, except Genwin.
`LibLoadInit`	void	(void);	Loads and initializes the Res library, making all Open Interface libraries available, except Genwin.
`Load`	ResPtr	(CStr *module*, CStr *resource*);	Loads a resource using two parameters.
`LoadByFullName`	ResPtr	(CStr *fullName*);	Loads a resource using a single parameter.
`LoadChildren`	void	(ResPtr *resource*);	Loads all of the children resources for a resource.

Function	**Returns**	**Arguments**	**Description**
`LoadDetach`	ResPtr	(CStr *module*, CStr *resource*);	Loads a detached resource.
`LoadInit`	ResPtr	(CStr *module*, CStr *resource*);	Loads and initializes an attached resource.
`LoadInitDetach`	void	(CStr *module*, CStr *resource*);	Loads and initializes a detached resource.
`LockedSendNfyData`	void	(ResPtr *res*, ResNfyEnum *code*, ClientPtr *nfyData*);	Notifies resource clients and sends data specified.
`ParentClassDefNfy`	void	(ResPtr *res*, ResNfyEnum *code*);	Trigger the parent default notification procedure on an instance.
`QueryFullName`	void	(ResCPtr *res*, Str *name*, StrIVal *length*);	Determines the full name of a resource.
`Release`	void	(ResPtr *res*);	Deallocates persistent resource fields.
`RemoveNfyHandler`	void	(ResPtr *res*, ResNfyEnum *nfy*);	Remove the handler installed to process the ‘nfy’ message to ‘res’.
`SaveDat`	void	(ResPtr *resource*);	Saves attached resources to library database (.DAT) file.
`SendCtrlNfyData`	void	(ResPtr *res*, ResNfyEnum *code*, ResCtrlNfyPtr *resCtrlNfy*);	Send a CtrlNfyData notification.
`SendNfy`	void	(ResPtr *resource*, ResNfyEnum *notif*);	Sends a notification to the notification procedure of a resource.
`SendNfyData`	void	(ResPtr *resource*, ResNfyEnum *notif*, ClientPtr *data*);	Notifies resource clients and sends data specified.
`SendNfyEnd`	void	(ResPtr *res*);	Sends a NDRes::NFYEND to the resource specified.
`SendNfyInit`	void	(ResPtr *res*);	Sends a NDRes::NFYINIT to the resource specified.
`SendNfyReset`	void	(ResPtr *res*);	Sends a NDRes::NFYRESET to the resource specified.
`SetClientData`	void	(ResPtr *res*, ClientPtr *data*);	Sets the specified client data of a resource.
`SetNfyHandler`	void	(ResPtr *res*, ResNfyEnum *nfy*, ResNfyHandlerProc *proc*);	Installs "proc" to process the "nfy" notification messages sent to "res". "proc" will be called with the resource, the notification code "nfy" and the nfyData corresponding to the notification ‘nfy’ as arguments.
`SetNfyHandlerClientData`	void	(ResPtr *res*, ResNfyEnum *nfy*, ClientPtr *data*);	Associates ‘data’ with the call-back defined for the resource and the notification ‘code’. ‘data’ can later be retrieved using RES_GetNfyHandlerClientData if needed.

Function	**Returns**	**Arguments**	**Description**
`SetNfyProc`	void	(ResPtr *res*, ResNfyProc *nfyproc*);	Sets a client notification routine for a resource.
`Use`	void	(ResPtr *res*);	Increments the reference count of a resource.
`VERIFY`	void	(ResCPtr *res*, RClasCPtr *class*);	Provides a handler for recovering from the specification of an invalid class.

RGN_ Class

Function	Returns	Arguments	Description
`IsEqual`	BoolEnum	(RegionCPtr *region1*, RegionCPtr *region2*);	Determines whether two regions are equal.
`IsEmpty`	BoolEnum	(RegionCPtr *region*)	Determines whether a region is empty.
`IsPointInside`	void	(RegionCPtr *region*, Point16CPtr *point*);	Determines whether a point is inside a region.
`PropagateAction`	PerfEnum	(RegionPtr *rgn*, RgnPerfProc *proc*, ClientPtr *clientdata*);	Performs an action on all rectangles belonging to region.
`QueryBounds`	void	(RegionCPtr *region*, Point16Ptr *point*);	Determines the boundaries of a region.
`RectIntersect`	void	(RegionPtr *region*, Rect16CPtr *rectangle*);	Creates an intersection of a region and a rectangle.
`RectPos`	RgnPosEnum	(RegionCPtr *region*, Rect16CPtr *rectangle*);	Returns a code indicating the position of a rectangle relative to a region.
`RectSet`	void	(RegionPtr *region*, Rect16CPtr *rectangle*);	Associates a region with a rectangle.
`RectSubtract`	void	(RegionPtr *region*, Rect16Ptr *rectangle*);	Subtracts a rectangle from a region.
`RectUnion`	void	(RegionPtr *region*, Rect16CPtr *rectangle*);	Creates a union of a rectangle and a region.
`RectXOr`	void	(RegionPtr *region*, Rect16CPtr *rectangle*);	Performs an exclusive Or between a region and a rectangle.
`Reset`	void	(RegionPtr *region*);	Resets a region to empty.
`RgnIntersect`	void	(RegionPtr *region1*, RegionCPtr *region2*);	Creates an intersection of two regions.
`RgnSet`	void	(RegionPtr *region1*, RegionCPtr *region2*);	Sets the coordinates of one region as specified by another.
`RgnSubtract`	void	(RegionPtr *region1*, RegionCPtr *region2*);	;Subtracts one region from another.
`RgnUnion`	void	(RegionPtr *region1*, RegionCPtr *region2*);	Creates a union of two regions.
`RgnXOr`	void	(RegionPtr *region1*, RegionCPtr *region2*);	Performs an exclusive Or on two regions.
`Translate`	void	(RegionPtr *region*, Point16CPtr *point*);	Translates a region by the offset specified.

RLIB_ Class

Function	Returns	Arguments	Description
`Close`	void	(RCLibPtr *lib*);	Closes a file.
`Dispose`	void	(RCLibPtr *lib*);	Unloads library ‘lib’ and all the resources it contains and closes the file.
`Find`	RLibPtr	(CStr *libname*);	Returns a pointer to a library.
`GetFirst`	RLibPtr	(RCLibPtr *lib*);	Returns the first library in the list.
`GetLibName`	CStr	(RCLibPtr *lib*);	Returns the name of a library.
`GetNext`	RLibPtr	(RClasPtr *lib*);	Returns the next library in the list.
`LoadEdit`	RLibPtr	(CStr *libname*, CStr *filename*);	Loads a library database file by full name in read-write mode and returns a pointer to the library.
`LoadFile`	RLibPtr	(CStr *fullname*);	Loads a library database file by full name and returns it.
`LoadLibFile`	RLibPtr	(RCLibPtr *lib*);	Loads a library.
`Open`	void	(RCLibPtr *lib*);	Opens a file.
`Unload`	void	(RCLibPtr *lib*);	Unloads library ‘lib’ from memory and closes the file.

SBUF_ Class

Function	Returns	Arguments	Description
AppendSBuf	void	(SBufPtr *sbuf1*, SBufCPtr *sbuf2*);	Append string, variable string or string buffer.
AppendStr	void	(SBufPtr *sbuf*, CStr *str*);	Append string, variable string or string buffer.
AppendChar8	void	(SBufPtr *sbuf*, ChCode *chcode*);	Append string, variable string or string buffer.
AppendStrSub	void	(SBufCPtr *sbuf2*, CStr *str*, StrIVal *slen*);	Append string, variable string or string buffer.
AppendVStr	void	(SBufPtr *sbuf*, VStrCPtr *vstr*);	Append string, variable string or string buffer.
Clear8	void	SBufPtr *sbuf*);	Clear context of string buffer specified.
CountToIndex	StrIVal	(SBufCPtr *sbuf*, StrIVal *n*);	Convert character count to index.
DownCase	void	(SBufPtr *sbuf*);	Convert string to lower case.
DownCaseSub	void	(SBufPtr *sbuf*, StrIVal *i1*, StrIVal *i2*);	Convert string to lower case.
GetBwrd	ChCode	(SBufCPtr *sbuf*, StrIVal *i*, StrIValPtr *wp*);	Return character code before or after index specified.
GetByte	Byte	(SBufCPtr *sbuf*, StrIVal *i*);	Returns the byte at index specified.
GetFwrd	ChCode	(SBufCPtr *sbuf*, StrIVal *i*, StrIValPtr *wp*);	Return character code before or after index specified.
GetLen	StrIVal	(SBufCPtr *sbuf*);	Returns the length of the string buffer.
GetStr	CStr	(SBufCPtr *sbuf*);	Returns the contents of the string buffer.
GetSubStr	CStr	(SBufPtr *sbuf*, StrIVal *i1*, StrIVal *i2*);	Return the string specified by index range.
IMatchesChar	BoolEnum	(SBufPtr *sbuf*, StrIVal i, ChCode *ch*, StrIValPtr *endp*);	Returns whether a case insensitive string match is found.
IMatchesSBuf	BoolEnum	(SBufPtr *sbuf*, StrIVal i, SBufPtr *sbuf2*, StrIValPtr *endp*);	Returns whether a case insensitive string match is found.
IMatchesStr	BoolEnum	(SBufPtr *sbuf*, StrIVal i, CStr *str*, StrIValPtr *endp*);	Returns whether a case insensitive string match is found.

Function	Returns	Arguments	Description
IMatchesStrSub	BoolEnum	(SBufPtr *sbuf*, StrIVal i, CStr *str*, StrIVal *slen*, StrIValPtr *endp*);	Returns whether a case insensitive string match is found.
IndexToCount	StrIVal	(SBufCPtr *sbuf*, StrIVal *i*);	Convert byte offset to character count.
InsertChar	StrIVal	(SBufPtr *sbuf*, StrIVal *i1*, ChCode *chcode*);	Insert character, string, variable string or string buffer at index specified.
InsertSBuf	StrIVal	(SBufPtr *sbuf*, StrIVal *i1*, SBufCPtr *sbuf2*);	Insert character, string, variable string or string buffer at index specified.
InsertStr	StrIVal	(SBufPtr *sbuf*, StrIVal *i1*, CStr *str*);	Insert character, string, variable string or string buffer at index specified.
InsertStrSub	StrIVal	(SBufPtr *sbuf*, StrIVal *i1*, CStr *str*, StrIVal *slen*);	Insert character, string, variable string or string buffer at index specified.
InsertVStr	StrIVal	(SBufPtr *sbuf*, StrIVal *i1*, VStrCPtr *vstr*);	Insert character, string, variable string or string buffer at index specified.
MatchesChar	BoolEnum	(SBufPtr *sbuf*, StrIVal *index*, ChCode *ch*, StrIValPtr *endp*);	Returns whether a case sensitive string match is found.
MatchesIChar	BoolEnum	(SBufPtr *sbuf*, StrIVal *index*, ChCode *ch*, BoolEnum *icase*, StrIValPtr *endp*);	Returns whether a case insensitive string match is found.
MatchesISBuf	BoolEnum	(SBufPtr *sbuf*, StrIVal i, SBufPtr *sbuf2*, BoolEnum *icase*, StrIValPtr *endp);*	Returns whether a case sensitive string match is found.
MatchesIStr	BoolEnum	(SBufPtr *sbuf*, StrIVal i, CStr *str*, BoolE-num *icase*, StrIValPtr *endp*);	Returns whether a case sensitive string match is found.
MatchesIStrSub	BoolEnum	(SBufPtr *sbuf*, StrIVal i, CStr *str*, StrIVal *slen*, BoolEnum *icase*, StrIValPtr *endp*);	Returns whether a case sensitive string match is found.
MatchesSBuf	BoolEnum	(SBufPtr *sbuf*, StrIVal i, SBufPtr *sbuf2*, StrIValPtr *endp);*	Returns whether a case sensitive string match is found.
MatchesStr	BoolEnum	(SBufPtr *sbuf*, StrIVal i, CStr *str*, StrIValPtr *endp*);	Returns whether a case sensitive string match is found.
MatchesStrSub	BoolEnum	(SBufPtr *sbuf*, StrIVal i, CStr *str*, StrIVal *slen*, StrIValPtr *endp*);	Returns whether a case sensitive string match is found.
RemoveChar	void	(SBufPtr *sbuf*, StrIVal *i1*);	Remove character at in-dex specified.

Function	Returns	Arguments	Description
`RemoveRange`	void	(SBufPtr *sbuf*, StrIVal *index1*, StrIVal *index2*);	Remove range of characters specified.
`ReplaceChar`	StrIVal	(SBufPtr *sbuf*, StrIVal *i*, ChCode *ch*);	Replace string character.
`SetSBuf`	void	(SBufPtr *sbuf*, SBufCPtr *sbuf2*);	Replace the contents of the string buffer.
`SetStr`	void	(SBufPtr *sbuf*, CStr *str*);	Replace the contents of the string buffer.
`SetStrSub`	void	(SBufPtr *sbuf*, CStr *str*, StrIVal *slen*);	Replace the contents of the string buffer.
`SetVStr`	void	(SBufPtr *sbuf*, VStrCPtr *vstr*);	Replace the contents of the string buffer.
`Truncate`	void	(SBufPtr *sbuf*, StrIVal *i*);	Truncate string at index specified.
`UpCase`	void	(SBufPtr *sbuf*);	Convert string to upper case.
`UpCaseSub`	void	(SBufPtr *sbuf*, StrIVal *i1*, StrIVal *i2*);	Convert string to upper case.

SCRPT_ Class

Function	Returns	Arguments	Description
`Compile`	ScrptPtr	(Str *sourceCode*);	Compiles the bare script passed to it as a string, and returns a pointer to the compiled form of that script.
`CompileFile`	ScrptPtr	(Str *fileName*);	Compiles the bare script contained in the file specified by `fileName' and returns a pointer to the compiled form of the script.
`CompileResource`	ScrptPtr	(Str *resName*);	Compiles the bare script contained in the string resource `resName' and returns a pointer to the compiled form of the script.
`Dispose`	void	(ScrptPtr *scrpt*);	This function disposes the compiled bare script `scrpt', freeing all of the memory used by the compiled form.
`Execute`	BoolEnum	(ScrptPtr *scrpt*);	This function executes the compiled script `scrpt'.
`ExecuteApp`	void	(void);	Loads the script from the file identified by fileName, compiles it and executes it.
`GetReturnType`	Int32	(ScrptPtr *scrpt*);	Obtains the type of the value which was returned by execution of the script `scrpt'.
`LibExit`	void	(void);	Unloading and uninstalling the script library.
`LibInit`	void	(void);	Convenience: Installing and initializing the script library.
`LibInstall`	void	(void);	Installing the script library.
`LibLoadInit`	void	(void);	Initialization and loading the script library.

Function	Returns	Arguments	Description
`QueryReturnValue`	void	(ScrptPtr *scrpt*, ClientPtr *value*);	Copies the value which was returned by execution of the script `scrpt' into the buffer pointed to by `value'.
`RegisterConstants`	void	(SrptRegisterConstPtr *constants*, BoolEnum *checkDup*);	Registers the constant identified by constants.
`RegisterEvents`	void	(ScrptRegisterEventPtr *event*, BoolEnum *checkDup*);	Registers the event identified by event.
`RegisterVerbs`	void	(ScrptRegisterVerbPtr *verb*, BoolEnum *checkDup*);	Registers the verb identified by verb.
`RunApp`	void	**(CStr** *fileName***);**	Loads the script from the file identified by fileName, compiles it and executes it.
`SetStringReturnValue`	void	**(Str** *val***);**	Sets the string to return to the script engine for newly registered verbs.

SET_ Class

Function	Returns	Arguments	Description
`AddElt`	void	(SetPtr *set*, SetEltVal *elt*);	Adds ’elt’ to the set (unless it is already in).
`AddElts`	void	(SetPtr *set*, SetLenVal *n*, SetEltValPtr *elts*);	Adds elts[0], elts[1], ..,elts[n-1] to the set (unless they are already in).
`Alloc`	SetPtr	(void);	Returns a pointer to an allocated but not yet constructed set.
`AreEqual`	void	(SetCPtr *A*, SetCPtr *B*);	Returns BOOL_TRUE if sets A and B are equal.
`ContainsElt`	BoolEnum	(SetCPtr *set*, SetEltVal *elt*);	Returns BOOL_TRUE if ’elt’ is in the set.
`Copy`	void	(SetPtr *dst*, SetCPtr *src*);	Empties ’dst’, then copies the contents of ’src’ into ’dst’.
`Dealloc`	void	(SetPtr *set*);	Deallocates the set.
`EmptySet`	SetPtr	(void);	Returns a pointer to a shared empty set.
`GetNumElts`	SetLenVal	(SetCPtr *set*);	Returns the number of elements in set.
`MixGetPartSet`	SetMixPartSet	(SetCPtr *A*, SetCPtr *B*);	Compares two sets A and B and returns the set of parts which are not empty.
`MixQueryParts`	void	(SetCPtr *A*, SetCPtr *B*, SetMixPartSet *parts*, SetPtr *C*);	Combines A and B and extracts the specified parts.
`QueryElts`	void	(SetCPtr *set*, SetLenVal *n*, SetEltValPtr *elts*);	Queries the first ’n’ elements of the set and put them into ’elts’.
`RemoveElt`	void	(SetPtr *set*, SetEltVal *elt*);	Removes ’elt’ from the set (unless it is not in).
`RemoveElts`	void	(SetPtr *set*, SetLenVal *n*, SetEltValPtr *elts*);	Removes elts[0], elts[1], ..,elts[n-1] from the set (unless they are not in the set).
`Reset`	void	(SetPtr *set*);	Empties the set.
`SetElts`	void	(SetPtr *set*, SetLenVal *n*, SetEltValPtr *elts*);	Sets the first ’n’ elements of the set to the values taken from ’elts’.

SET Class

STR_ Class

Function	Returns	Arguments	Description
`Append`	void	(StrPtr *sptr*, CStr *cstr*);	Appends a string.
`AppendSub`	void	(StrPtr *str*, CStr *cstr*, StrIVal *strival*);	Appends a substring.
`AsciiDownCase`	void	(Str *s*);	Converts a string to lower case.
`AsciiDownCaseSub`	void	(Str *s*, StrIVal *l)*;	Converts a substring to lower case.
`AsciiUpCase`	void	(Str *s)*;	Converts a substring to upper case.
`AsciiUpCaseSub`	void	(Str *s*, StrIVal *l*);	Converts a string to upper case.
`Clone`	Str	(CStr *s*);	Returns a new copy of a string.
`Cmp`	CmpEnum	(CStr *s1*, CStr *s2*);	Compares two strings.
`CmpSub`	CmpEnum	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*);	Compares two substrings.
`CtGetBwrd`	NatCode	(NatCStr *nStr*, CtCPtr *ct*, StrIVal *l*);	Returns the code found in front of a location in a string.
`CtGetCode`	NatCode	(NatCStr *nStr*, CtCPtr *ct*);	Returns the character code found at the beginning of a string and sets the length.
`CtGetFwrd`	NatCode	(NatCStr *nStr*, CtCPtr *ct*, StrIValPtr *lenp*);	Returns the character code found at the beginning of a string and sets the length.
`Dispose`	void	(Str *str*);	Disposes of a string buffer.
`Dispose0`	void	(Str *s*);	Disposes of a string buffer if the buffer is not NULL.
`Equals`	BoolEnum	(CStr *s1*, CStr *s2*);	Compares two strings for equality.
`EqualsSub`	BoolEnum	(CStr *s1*, StrIVal *l1*, CStr c2, StrIVal *l2*);	Compares two substrings for equality.
`FindFirst`	StrIVal	(CStr *s1*, CStr *s2*);	Finds the first occurrence of a string.
`FindFirstChar`	StrIVal	(CStr *s*, ChCode *chcode*);	Finds the first occurrence of a character.
`FindFirstCharSub`	StrIVal	(CStr *s*, StrIVal *l1*, ChCode *chcode*);	Find the first occurrence of a character in a substring.
`FindFirstSub`	StrIVal	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*);	Switchable case-independent search for the first occurrence of a substring.
`FindIFirst`	StrIVal	(CStr *s1*, CStr *s2*, BoolEnum *caseI*);	Switchable case-independent search for the first occurrence of a string.
`FindIFirstSub`	StrIVal	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*, BoolEnum *caseI*);	Switchable case-independent search for the first occurrence of a substring.
`FindILast`	StrIVal	(CStr *s1*, CStr *s2*, BoolEnum *caseI*);	Switchable case-independent search for the last occurrence of a string.

Function	Returns	Arguments	Description
`FindILastSub`	StrIVal	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*, BoolEnum *caseI*);	Switchable case-independent search for the last occurrence of a substring.
`FindLast`	StrIVal	(CStr *s1*, CStr *s2*);	Finds the last occurrence of a string.
`FindLastChar`	StrIVal	(CStr *s*, ChCode *chcode*);	Finds the last occurrence of a character within a string
`FindLastCharSub`	StrIVal	(CStr *s*, StrIVal *l1*, ChCode *chcode*);	Finds the last occurrence of a character in a substring.
`FindLastSub`	StrIVal	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*);	Switchable case-independent search for the last occurrence of a substring.
`GetBwrd`	ChCode	(CStr *str*, StrIVal *i*, StrIValPtr *lenp*);	Returns the code found in front of the specified location.
`GetCode`	ChCode	(CStr *str*);	Returns the character code found at the beginning of a string.
`GetDecInt`	Int	(CStr *s*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer string.
`GetDecInt16`	Int16	(CStr *s*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer string.
`GetDecInt32`	Int32	(CStr *s*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer string.
`GetDecUInt`	UInt	(CStr *s*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer string.
`GetDecUInt16`	UInt16	(CStr *s*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer string.
`GetDecUInt32`	UInt32	(CStr *s*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer string.
`GetDouble`	Double	(CStr *s*, StrIValPtr *lenp*);	Returns the double real numeric value found at the beginning of a double real string.
`GetFwrd`	ChCode	(CStr *s*, StrIValPtr *lenp*);	Returns the character code at the beginning of a string and sets its length.
`GetHexInt`	Int	(CStr *s*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a hexadecimal integer string.
`GetHexInt16`	Int18	(CStr *s*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a hexadecimal integer string.
`GetHexUInt`	UInt	(CStr *s*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a hexadecimal integer string.
`GetHexUInt16`	UInt16	(CStr *s*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a hexadecimal integer string.
`GetLen`	StrIVal	(CStr *s*);	Returns the length of a string.

Function	Returns	Arguments	Description
`GetRadixInt`	Int	(CStr *s*, Int *radix*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of an integer string.
`GetRadixInt16`	Int16	(CStr *string*, Int *radix*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of an integer string.
`GetRadixUInt32`	Int32	(CStr *string*, Int *radix*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of an integer string.
`GetTruncLen`	StrIVal	(CStr *string*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of an integer string.
`ICmp`	CmpEnum	(CStr *s1*, CStr *s2*);	Compares two strings.
`ICmpSub`	CmpEnum	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*);	Compares two substrings.
`IEquals`	BoolEnum	(CStr *s1*, CStr *s2*);	Compares two strings for equality.
`IEqualsSub`	BoolEnum	(CStr *s1*, StrIVal *l1*, CStr *c2*, StrIVal *l2*);	Compares two substrings for equality.
`IFindFirstSub`	StrIVal	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*);	Find the first occurrence of a character in a substring.
`IMatchesPat`	BoolEnum	(CStr *s1*, CStr *s2*, StrIValPtr *lenp*);	Tests whether one string matches another string containing a pattern.
`IMatchesPatSub`	BoolEnum	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*, StrIValPtr *lenp*);	Tests whether one string matches another string containing a pattern.
`IMatchesSub`	BoolEnum	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*, StrIValPtr *lenp*);	Tests whether one string matches another string containing a pattern.
`Matches`	BoolEnum	(CStr *s1*, CStr *s2*, StrIValPtr *lenp*);	Tests whether one string matches another string containing a pattern.
`MatchesChar`	BoolEnum	(CStr *s1*, CStr *s2*, StrIValPtr *lenp*);	Tests whether one string matches another string containing a pattern.
`MatchesPat`	BoolEnum	(CStr *s1*, CStr *s2*, StrIValPtr *lenp*);	Tests whether one string matches another string containing a pattern.
`MatchesPatSub`	BoolEnum	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*, StrIValPtr *lenp*);	Tests whether one string matches another string containing a pattern.
`MatchesSub`	BoolEnum	(CStr *s1*, StrIVal *l1*, CStr *s2*, StrIVal *l2*, StrIValPtr *lenp*);	Tests whether one string matches another string containing a pattern.
`NatPutAscii`	StrIVal	(NatStr *nStr*, StrIVal *size*, NatCode *ncode*);	Writes an ASCII character into a native string.
`NatPutCode`	StrIVal	(NatStr *nStr*, StrIVal *size*, NatCode *ncode)*;	Writes a character code into a string.
`NatWriteAscii`	StrIVal	(NatStr *nStr*, StrIVal *size*, NatCode *ncode*);	
`NatWriteCode`	StrIVal	(NatStr *nStr*, StrIVal *size*, NatCode *ncode)*;	Writes a native character code into a native string without terminating the string with NULL.
`NewSet`	void	(CStr *s*);	Returns a new copy of a string.

Function	Returns	Arguments	Description
`Put`	StrIVal	(Str *buf*, StrIVal *size*, CStr *str*, StrIValPtr *endp*);	Writes a string into a buffer.
`PutAscii`	StrIVal	(Str *buf*, StrIVal *size*, Char *ch*);	Writes an ASCII character into a string.
`PutAsciiLower`	StrIVal	(Str *buf*, StrIVal *len*, CStr *str*, StrIValPtr *lenp*);	Same as NDStr::Put, but also converts the ASCII characters in the string to lower case.
`PutAsciiLowerSub`	StrIVal	(Str *buf*, StrIVal *len*, CStr *str*, StrIVal *slen*, StrIValPtr *lenp*);	Same as NDStr::PutSub, but also converts a substring to lower case.
`PutAsciiUpper`	StrIVal	(Str *buf*, StrIVal *len*, CStr *str*, StrIValPtr *lenp*);	Same as NDStr::PutSub, but also converts a substring to lower case.
`PutAsciiUpperSub`	StrIVal	Str *buf*, StrIVal *len*, CStr *str*, StrIVal *slen*, StrIValPtr *lenp*);	Same as NDStr::PutSub, but also converts the ASCII characters in the substring to upper case.
`PutCode`	StrIVal	(Str *buf*, StrIVal *size*, ChCode *chcode*);	Writes a character code into a string.
`PutDecInt`	StrIVal	(Str *buf*, StrIVal *size*, Int *i*);	Converts a decimal integer into its textual representation in a string buffer.
`PutDecInt16`	StrIVal	(Str *buf*, StrIVal *size*, Int16 *i*);	Converts a decimal integer into its textual representation in a string buffer.
`PutDecInt32`	StrIVal	(Str *buf*, StrIVal *size*, Int32 *i)*;	Converts a decimal integer into its textual representation in a string buffer.
`PutDecUInt`	StrIVal	(Str *buf*, StrIVal *size*, UInt *i*);	Converts a decimal integer into its textual representation in a string buffer.
`PutDecUInt16`	StrIVal	(Str *buf*, StrIVal *size*, UInt16 *int*);	Converts a decimal integer into its textual representation in a string buffer.
`PutDecUInt32`	StrIVal	(Str *buf*, StrIVal *size*, UInt32 *int*);	Converts a decimal integer into its textual representation in a string buffer.
`PutDouble`	StrIVal	(Str *buf*, StrIVal *size*, Double *d*);	Converts a double precision value into its textual representation in the string buffer.
`PutHexInt`	StrIVal		Converts a hexadecimal integer into its textual representation in a string buffer.
`PutHexInt16`	StrIVal	(Str *buf*, StrIVal *size*, Int16 *int*);	Converts a hexadecimal integer into its textual representation in a string buffer.
`PutHexInt32`	StrIVal	(Str *buf*, StrIVal *size*, Int32 *int*);	Converts a hexadecimal integer into its textual representation in a string buffer.
`PutHexUInt`	StrIVal	(Str *buf*, StrIVal *size*, UInt *int*);	Converts a hexadecimal integer into its textual representation in a string buffer.
`PutHexUInt16`	StrIVal	(Str *buf*, StrIVal *size*, UInt16 *int*);	Converts a hexadecimal integer into its textual representation in a string buffer.
`PutHexUInt32`	StrIVal	(Str *buf*, StrIVal *size*, UInt32 *int)*;	Converts a hexadecimal integer into its textual representation in a string buffer.

Function	Returns	Arguments	Description
`PutRadixInt16`	StrIVal	(Str *buf*, StrIVal *size*, Int *radix*, Int16 *int*);	Using the radix, converts an integer into its textual representation in the string buffer.
`PutRadixInt32`	StrIVal	(Str *buf*, StrIVal *size*, Int *radix*, Int32 *int*);	Using the radix, converts an integer into its textual representation in the string buffer.
`PutRadixUInt`	StrIVal	(Str *buf*, StrIVal *size*, Int *radix*, UInt *i)*;	Using the radix, converts an integer into its textual representation in the string buffer.
`PutRadixUInt16`	StrIVal	(Str *buf*, StrIVal *size*, Int *radix*, UInt16 *i)*;	Using the radix, converts an integer into its textual representation in the string buffer.
`PutRadixUInt32`	StrIVal	(Str *buf*, StrIVal *size*, Int *radix*, UInt32 *i)*;	Using the radix, converts an integer into its textual representation in the string buffer.
`PutSub`	StrIVal	(Str *buffer*, StrIVal *size*, CStr *s*, StrIVal *slength*, StrIValPtr *endptr*);	Writes a substring into a string buffer.
`ResFind`	CStr	(CStr *mod*, CStr *res*, StrIValP-tr *lenp*);	Finds and returns a string from a StrR or StrL resource.
`ResFindNth`	CStr	(CStr *mod*, CStr *res*, Ar-rayIVal *n*, StrIValPtr *lenp*);	Finds and returns a string from a StrR or StrL resource.
`ResLoad`	CStr	(CStr *mod*, CStr *res*, StrIValP-tr *lenp*);	Returns a string from a StrR or StrL re-source.
`ResLoadNth`	CStr	(CStr *mod*, CStr *resource*, Ar-rayIVal *n*, StrIValPtr *lenp*);	Returns a string from a StrR or StrL re-source.
`Set`	void	(StrPtr *sPtr*, CStr *cstr*);	Setting a new string to contain the con-tents of an existing string.
`SetSub`	void	(StrPtr *sPtr*, CStr *cstr*, StrIVal *length*);	Assigns a substring as the contents of an existing string.
`SubGetDecInt`	Int	(CStr *s*, StrIVal l, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer sub-string.
`SubGetDecInt16`	Int16	(CStr *s*, StrIVal l, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer sub-string.
`SubGetDecInt32`	Int32	(CStr *s*, StrIVal l, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer sub-string.
`SubGetDecUInt`	UInt	(CStr *s*, StrIVal l, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer sub-string.
`SubGetDecUInt16`	UInt16	(CStr *s*, StrIVal l, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer sub-string.

Function	**Returns**	**Arguments**	**Description**
`SubGetDecUInt32`	UInt32	(CStr *s*, StrIVal l, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a decimal integer sub-string.
`SubGetDouble`	Double	(CStr *s*, StrIVal *l*, StrIValPtr *lenp*);	Returns the double real numeric value found at the beginning of a double real substring.
`SubGetHexInt`	Int	(CStr *s*, StrIVal l, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a hexadecimal integer substring.
`SubGetHexInt16`	Int16	(CStr *s*, StrIVal l, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a hexadecimal integer substring.
`SubGetHexInt32`	Int32	(CStr *s*, StrIVal *l*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a hexadecimal integer substring.
`SubGetHexUInt`	UInt	(CStr *s*, StrIVal l, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a hexadecimal integer substring.
`SubGetHexUInt16`	UInt16	(CStr *s*, StrIVal l, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a hexadecimal integer substring.
`SubGetHexUInt32`	UInt32	(CStr *s*, StrIVal *l*, StrIValPtr *lenp*);	Returns the integer value found at the beginning of a hexadecimal integer substring.
`SubGetRadixInt`	Int	`;`(CStr *s*, StrIVal *l*, Int *radix*, StrIValPtr *lenp*);	Returns the integer value found at the *beg*inning of an integer substring.
`SubGetRadixInt16`	Int16	`;`(CStr *s*, StrIVal *l*, Int *radix*, StrIValPtr *lenp*);	Returns the integer value found at the *beg*inning of an integer substring.
`SubGetRadixInt32`	Int32	`;`(CStr *s*, StrIVal *l*, Int *radix*, StrIValPtr *lenp*);	Returns the integer value found at the *beg*inning of an integer substring.
`SubGetRadixUInt`	UInt	`;`(CStr *s*, StrIVal *l*, Int *radix*, StrIValPtr *lenp*);	Returns the integer value found at the *beg*inning of an integer substring.
`SubGetRadixUInt16`	UInt16	`;`(CStr *s*, StrIVal *l*, Int *radix*, StrIValPtr *lenp*);	Returns the integer value found at the *beg*inning of an integer substring.
`SubGetRadixUInt32`	UInt32	`;`(CStr *s*, StrIVal *l*, Int *radix*, StrIValPtr *lenp*);	Returns the integer value found at the *beg*inning of an integer substring.
`WriteAscii`	StrIVal	(Str *str*, StrIVal *size*, Char *ch*);	Writes an ASCII character into a string without terminating the string with NULL.
`WriteCode`	StrIVal	(Str *str*, StrIVal *size*, ChCode *chcode*);	Writes a character code into a string without terminating the string with NULL.

STRL_ Class

Function	Returns	Arguments	Description
`AddStr`	void	(StrLPtr *stringList*, CStr *string*);	Adds a new string to a string list resource.
`AddStrAtIndex`	void	(StrLPtr *stringList*, ArrayIVal *stringIndex*, CStr *string*);	Inserts a new string in a string list resource at the position specified.
`Class`	RClasPtr	(void);	Returns a pointer to the string list class.
`FindNthStr`	CStr	(StrLCPtr *stringList*, CStr *string*, CStr *string*, ArrayIVal *stringIndex*);	Returns a string specified by index from a string list resource.
`GetLen`	ArrayIVal	(StrLCPtr *stringList*);	Returns the number of strings stored in a string list resource.
`GetNthStr`	CStr	(StrLCPtr *stringList*, ArrayIVal *stringIndex*);	Returns a string specified by index from a string list resource.
`LoadNthStr`	CStr	(CStr *mod*, CStr *res*, ArrayIVal *stringIndex*);	Returns a string from a string list resource by resource name and index.
`RemoveIndex`	void	(StrLCPtr *stringList*, ArrayIVal *stringIndex*);	Removes a string from a string list resource at the position specified.
`SetNthStr`	void	(StrLPtr *stringList*, ArrayIVal *stringIndex*, CStr *string*);	Replaces an existing string in a string list resource with a new string.

STRR_ Class

Function	Returns	Arguments	Description
`Class`	RClasPtr	(void);	Returns a pointer to the string resource class.
`FindStr`	CStr	(CStr *modName*, CStr *res*);	Returns the string contained in a string resource.
`GetID`	StrRIdVal	(StrRCPtr *stringRes*);	Returns the id of a string resource.
`GetStr`	CStr	(StrRCPtr *stringRes*);	Returns the string contained in a string resource.
`LoadStr`	CStr	(CStr *modName*, CStr *res*);	Loads the string resource by resource name.
`SetId`	void	(StrRPtr *stringRes*, StrRIdVal *id*);	Changes the id value of a string re-source.
`SetStr`	void	(StrRPtr *stringRes*, CStr *string*);	Replaces the string in a string resource with a new string.

VAR_ Class

Function	Returns	Arguments	Description
`Clear`	void	(VarRef *var*);	Empties this variant.
`ContainsRef`	BoolEnum	(VarCRef *var*);	Returns BOOL_TRUE if this variant contains a reference
`Convert`	void	(VarRef *var*, VarTypeEnum *destType*);	Converts the type of this any to the type specified by `destType'.
`ConvertToValue`	void	(VarRef *var*);	If this any contains a reference the method converts this any to a value obtained by dereferencing the reference.
`CopyToBoolean`	BoolEnum	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToByte`	Byte	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToChar`	Char	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToChCode`	ChCode	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToClientPtr`	ClientPtr	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToDouble`	Double	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.

Function	Returns	Arguments	Description
`CopyToFloat`	Float	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToInt`	Int	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToInt8`	Int8	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToInt16`	Int16	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToInt32`	Int32	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToInt64`	Int64	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToLong`	Long	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToStr`	Str	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToType`	VarPtr	(VarCRef *var*, VarTypeEnum *destType*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.

Function	Returns	Arguments	Description
`CopyToUInt`	UInt	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToUInt8`	UInt8	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToUInt16`	UInt16	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToUInt32`	UInt32	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToUInt64`	UInt64	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToULong`	ULong	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToValue`	VarPtr	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToVARWChar`	VARW-Char	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`CopyToVARWStr`	VARWStr	(VarCRef *var*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.

Function	Returns	Arguments	Description
`GetType`	VarTypeEnum	(VarCRef *var*);	Add a row at index `index' in the list by internally creating and committing an edition object.
`IsEmpty`	BoolEnum	(VarCRef *var*);	Returns BOOL_TRUE if the type of this variant is VAR_TYPE_NONE. Returns BOOL_FALSE otherwise**.**
`IsNULL`	BoolEnum	(VarCRef *var*);	Returns BOOL_TRUE if the type of this variant is VAR_TYPE_NULL. Returns BOOL_FALSE otherwise**.**
`IsNULLObj`	BoolEnum	(VarCRef *var*);	Returns BOOL_TRUE if the type of this variant is VAR_TYPE_NULLOBJ. Returns BOOL_FALSE otherwise**.**
`TryConvert`	BoolEnum	(VarRef *var*, VarTypeEnum *destType*);	Converts the type of this any to the type specified by `destType'.
`TryConvertToValue`	BoolEnum	(VarRef *var*);	If this object contains a reference, converts this to a value obtained by dereferencing the reference.
`TryCopyToBoolEnum`	BoolEnum	(VarCRef *var*, BoolEnumPtr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToByte`	BoolEnum	(VarCRef *var*, BytePtr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToChar`	BoolEnum	((VarCRef *var*, CharPtr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`TryCopyToChCode`	BoolEnum	(VarCRef *var*, ChCodePtr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**

Function	Returns	Arguments	Description
`TryCopyToClientPtr`	BoolEnum	(VarCRef *var*, ClientPtr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToDouble`	BoolEnum	(VarCRef *var*, DoublePtr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToFloat`	BoolEnum	(VarCRef *var*, Float Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToInt`	BoolEnum	**(V**arCRef *var*, IntPtr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToInt8`	BoolEnum	(VarCRef *var*, Int8Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToInt16`	BoolEnum	(VarCRef *var*, Int16 Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToInt32`	BoolEnum	(VarCRef *var*, Int32 Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToInt64`	BoolEnum	(VarCRef *var*, Int64Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToLong`	BoolEnum	(VarCRef *var*, Long Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**

Function	Returns	Arguments	Description
`TryCopyToStr`	BoolEnum	(VarCRef *var*, Str Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`TryCopyToType`	BoolEnum	(VarCRef *var*, VarTypeEnum *destType*, VarPtrPtr *valuePtr*);	Returns an NDVar which contains this any converted to the type specified by `destType'.
`TryCopyToUInt`	BoolEnum	(VarCRef *var*, UIntPtr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`TryCopyToUInt8`	BoolEnum	(VarCRef *var*,UInt8Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`TryCopyToUInt16`	BoolEnum	(VarCRef *var*, UInt16Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`TryCopyToUInt32`	BoolEnum	(VarCRef *var*, UInt32Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`TryCopyToUInt64`	BoolEnum	(VarCRef *var*, UInt64Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`TryCopyToULong`	BoolEnum	(VarCRef *var*, UInt8Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated.
`TryCopyToValue`	BoolEnum	(VarRef *var*);	If this contains a reference, the method returns an NDVar containing a value obtained by dereferencing the reference.

Function	Returns	Arguments	Description
`TryCopyToVARWChar`	BoolEnum	(VarCRef *var*, VARWChar Ptr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryCopyToVARWStr`	BoolEnum	(VarCRef *var*,VARWStrPtr *valuePtr*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**
`TryUpdate`	BoolEnum	(VarRef *var,* VarCRef *ori*);	If this object contains a reference, writes the value of `ori' into the reference, otherwise an exception is generated**.**

VARDS_ Class

Function	Returns	Arguments	Description
`DefNfy`	void	(VarDsPtr *varDs*, VarDsNfyEnum *code*);	Default notification procedure for the VarDs class
`GetValue`	VarPtr	(VarDsPtr *varDs*);	Sets the value of the data source by internally creating an edition object and committing the changes created through it**.**
`QueryValue`	void	VarDsPtr *varDs*, VarPtr *var*);	Returns the value associated to the variant data source in `var'**.**
`SetValue`	BoolEnum	(VarDsPtr *varDs*, VarPtr *var*);	Sets the value of the data source by internally creating an edition object and committing the changes created through it**.**

VARDSEDIT_ Class

Function	Returns	Arguments	Description
`SetValue`	void	(VarDsEditPtr *editDs*, VarPtr *var*);	Sets the value of the data source edition object.

VARLS_Class

Function	Returns	Arguments	Description
AddRow	BoolEnum	(VarLsPtr *varLs*, VarLsIndexVal *index*);	Add a row at index `index' in the list by internally creating and committing an edition object.
Class	void	(VarLsPtr *varLs*);	Adds a row at index `index' in the edition object.
DefNfy	void	(VarLsPtr *varLs*, VarLsNfyEnum *code*);	Default notification procedure for the VarLs class**.**
GetCursorRow	**VarLsIndexVa**l	(VarLsPtr *varLs*);	Returns the current position of the cursor of the list**.**
GetMaxRowTitleStrLen	**StrIVa**l	(VarLsPtr *varLs*);	Returns the length of the longest string for a row title in the list, if the source can provide it.
GetMaxStrLen	**StrIVa**l	(VarLsPtr *varLs*);	Returns the length of the longest string for any row in the list, if the source can provide it.
GetMods	VarLs-ModsCPtr	(VarLsPtr *varLs*);	Get a description of the last modifications made on the list through an edition object**.**
GetNumRows	VarLsIndexVal	(VarLsPtr *varLs*);	Returns the number of rows in the list data source.
GetRowTitle	CStr	(VarLsPtr *varLs*, VarLsIndexVal *index*);	Returns the title for the row `index' in the list, if any**.**
GetRowValue	VarPtr	(VarLsPtr *varLs*, VarLsIndexVal *index*);	Returns the value of the row `index' in the list. T
GetTitle	CStr	(VarLsPtr *varLs*);	Returns the title for the list, if any**.**
QueryRowValue	void	(VarLsPtr *varLs*, VarLsIndexVal *index*, VarPtr *value*)	Returns the value of the row `index' in the list**.**
RemoveRow	BoolEnum	(VarLsPtr *varLs*, VarLsIndexVal *index*);	Removes a row from the list by internally creating and committing an edition. Returns BOOL_TRUE if the internal edit succeeded, BOOL_FALSE in any other case**.**

Function	Returns	Arguments	Description
`SetCursorRow`	BoolEnum	(VarLsPtr *varLs*, VarLsIndexVal *index*);	Sets the current list cursor to `row' by internally creating and committing an edition object.
`SetNumRows`	BoolEnum	(VarLsPtr *edit*, VarLsIndexVal *index*);	Sets the number of rows in the list internally creating and committing a edition object.
`SetRowTitle`	BoolEnum	(VarLsPtr *varLs*, VarLsIndexVal *index*, CStr *title*);	Sets the title of the row identified by `index' to `title' by internally creating and committing an edition.
`SetRowValue`	BoolEnum	(VarLsPtr *varLs*, VarLsIndexVal *index*, VarPtr *value*);	Sets the value corresponding to the row `index' of the edition object to `value'.
`SetTitle`	BoolEnum	(VarLsPtr *varLs*, CStr *title*);	Sets the title of the list to `title' by internally creating and committing an edition.
`StartRowEdit`	VarLsEditPtr	(VarLsPtr *varLs*, VarLsIndexVal *index*);	Open a edit (for modifying the `index' row).

VARLSEDIT_ Class

Function	Returns	Arguments	Description
`AddRow`	void	(VarLsEditPtr *varLs*, VarLsIndexVal *index*);	Adds a row at index `index' in the edit
`RemoveRow`	void	(VarLsEditPtr *edit*, VarLsIndexVal *index*);	Removes the row at index `index' in th
`SetCursorRow`	void	(VarLsEditPtr *edit*, VarLsIndexVal *index*);	Sets the current edition object cursor to
`SetNumRows`	void	(VarLsEditPtr *edit*, VarLsIndexVal *index*);	Sets the number of rows in an edition
`SetRowTitle`	void	(VarLsEditPtr *edit*, VarLsIndexVal *index*, CStr *title*);	Sets the title of the row'index' of the ed the title will be copied onto the row'inc
`SetRowValue`	void	(VarLsEditPtr *edit*, VarLsIndexVal *index*, VarPtr *value*);	Sets the value corresponding to the rov ject to `value'.
`SetTitle`	void	(VarLsEditPtr *edit*, CStr *str*);	Sets the title of the list edition object to

VARTB_ Class

Function	Returns	Arguments	Description
`AddColumn`	void	(VarTbPtr *varTb*, VarTbIndexVal *index*);	Add a column at index `index' to the edition object.
`AddRow`	BoolEnum	(VarTbPtr *varTb*, VarTbIndexVal *i);*	Adds a row at index `index' to the table, by internally creating and committing a edition.
`Class`	RClasPtr	(void);	Returns a pointer to the resource class for VarTb**.**
`GetCellValue`	VarPtr	(VarTbPtr *varTb*, VarTbIndexVal *row*, VarTbIndexVal *col*);	Returns the value of the cell at row `row' and column `col' in the list.
`GetColumnTitle`	CStr	(VarTbPtr *varTb*, VarTbIndexVal *index*);	Returns the title for the column `index' in the table, if any**.**
`GetCursorColumn`	VarTbIndexVal	(VarTbPtr *varTb*);	Returns the current position of the column cursor
`GetCursorRow`	VarTbIndexVal	(VarTbPtr *varTb*);	Returns the current position of the row cursor
`GetMaxColStrLen`	StrIVal	(VarTbPtr *varTb*, VarTbIndexVal *col*);	Returns the length of the longest string for a column in the table (including the title), if the source can provide it.
`GetMods`	VarTbModsCPtr	(VarTbPtr *varTb*);	Get a description of the last modifications committed on the table data source**.**
`GetNumColumns`	VarTbIndexVal	(VarTbPtr *varTb*);	Returns the number of columns in the table**.**
`GetNumRows`	VarTbIndexVal	(VarTbPtr *varTb*);	Returns the number of rows in the table**.**
`GetRowTitle`	CStr	(VarTbPtr *varTb*, VarTbIndexVal *index*);	Returns the title for the row `index' in the table, if any**.**
`GetTitle`	**CS**tr	(VarTbPtr *varTb*);	Returns the title for the table, if any**.**
`QueryCellValue`	void	(VarTbPtr *varTb*, VarTbIndexVal *row*, VarTbIndexVal *col*, VarPtr *value*);	Returns the value of the cell at row `row' and column `col' the list**.**
`RemoveColumn`	BoolEnum	(VarTbPtr *varTb*, VarTbIndexVal *index*);	Remove a column through a edition**.**

Function	**Returns**	**Arguments**	**Description**
`RemoveRow`	BoolEnum	(VarTbPtr *varTb*, VarTbIndexVal *index*);	Removes the row at index `index' in the table.
`SetCellValue`	BoolEnum	(VarTbPtr *varTb*, VarTbIndexVal *row*, VarTbIndexVal *col*, VarPtr *value*);	Sets the value corresponding to the cell identified by `row' and `col' of the edition object to `value'.
`SetColumnTitle`	BoolEnum	(VarTbPtr *varTb*, VarTbIndexVal *col*, CStr *title*);	Sets the title of the column identified by `col' to `title' by internally creating and committing a edition
`SetCursorColumn`	BoolEnum	(VarTbPtr *varTb*, VarTbIndexVal *index*);	Sets the table cursor column of the edition object to `index'
`SetCursorRow`	BoolEnum	(VarTbPtr *varTb*, VarTbIndexVal *index*);	Sets the table cursor row in the edition object to `index'.
`SetNumRowColumns`	BoolEnum	VarTbPtr *varTb*, VarTbIndexVal *numRows*, VarTbIndexVal *numCols*);	Set up the number of rows and columns of the table by internally creating and committing a edition.
`SetRowTitle`	BoolEnum	(VarTbPtr *varTb*, VarTbIndexVal *row*, CStr *title*);	Sets the title of the row identified by `row' to `title' by internally creating and committing a edition.
`SetTitle`	BoolEnum	(VarTbPtr *varTb*, CStr *str*);	Sets the title of the table by internally creating and committing a edition. R
`StartCellEdit`	VarTbEditPtr	(VarTbPtr *varTb*, VarTbIndexVal *row*, VarTbIndexVal *col*);	Open a edition for modifying the cell at (`row', `col').
`StartEdit`	void	(VarTbPtr *varTb*);	Opens an edition on the whole table data source.
`StartRowEdit`	VarTbEditPtr	(VarTbPtr *varTb*, VarTbIndexVal *index*);	Open a edition (for modifying the `index' row).

VARTBEDIT_ Class

Function	Returns	Arguments	Description
`AddColumn`	void	(VarTbEditPtr *edit*, VarTbIndexVal *index*);	Add a column by internally creating a Returns BOOL_TRUE if the internal ec
`AddRow`	void	(VarTbEditPtr *edit*, VarTbIndexVal *index*);	Add a row through edition.
`RemoveColumn`	void	(VarTbEditPtr *edit*, VarTbIndexVal *index*);	Remove a column through a edition**.**
`RemoveRow`	void	(VarTbEditPtr *edit*, VarTbIndexVal *index*);	Removes the row at index `index' in th
`SetCellValue`	void	(VarTbEditPtr *edit*, VarTbIndexVal *row*, VarTbIndexVal *col*, VarPtr *value*);	Sets the value corresponding to the cel `col' of the edition object to `value'.
`SetColumnTitle`	void	(VarTbEditPtr *edit*, VarTbIndexVal *index*, CStr *title*);	Sets the title corresponding to the colu through a edition.
`SetCursorColumn`	void	(VarTbEditPtr *edit*, VarTbIndexVal *index*);	Sets the table cursor column of the edi
`SetCursorRow`	void	(VarTbEditPtr *edit*, VarTbIndexVal *index*	Sets the table cursor row in the edition
`SetNumRowColumns`	void	(VarTbEditPtr *edit*, VarTbIndexVal *numRows*, VarTbIndexVal *numCols*);	Sets up the number of rows and colum
`SetRowTitle`	void	(VarTbEditPtr *edit*, VarTbIndexVal *row*, CStr *title*);	Sets the title of the row identified by `r creating and committing a edition.
`SetTitle`	void	(VarTbEditPtr *edit*, CStr *str*);	Sets the title of the table through a edi

VSTR_ Class

Function	Returns	Arguments	Description
`Append`	void	(VStrPtr *vstr*, VStrCPtr *vstr2*);	Appends one variable string to another.
`AppendChar`	void	(VStrPtr *vstr*, ChCode *ch*);	Appends a character to a variable string.
`AppendStr`	void	(VStrPtr *vstr*, CStr *str*);	Appends a string to a variable string.
`AppendStrSub`	void	(VStrPtr *vstr*, CStr *str*, StrIVal *slen*);	Appends a substring to a variable string.
`ArrayAlloc`	VStrArrayPtr	(void);	Allocates a VStr array.
`Clear`	void	(VStrPtr *vstr*);	Resets a variable string.
`Cmp`	CmpEnum	(VStrCPtr *vstr*, VStrCPtr *vstr2*);	Compares two variable strings, ignoring case differences in the ASCII range,
`CmpStr`	CmpEnum	(VStrCPtr *vstr*, CStr *vstr2*);	Compares a variable strings with another string by comparing the characters in each string by code value.
`Copy`	void	(VStrPtr *vstr*, VStrCPtr *vstr2*);	Copies one variable string to another.
`GetLen`	StrIVal	(VStrCPtr *vstr*);	Returns the length of a variable string.
`GetStr`	CStr	(VStrCPtr *vstr*);	Returns the string equiv-alent of a variable string.
`ICmp`	CmpEnum	(VStrCPtr *vstr*, VStrCPtr *vstr2*);	Compares two variable strings, ignoring case differences in the ASCII range,
`ICmpStr`	CmpEnum	(VStrCPtr *vstr*, CStr *str2*);	Compares a variable strings with another string by comparing the characters in each string by code value.
`QueryStrSub`	void	(VStrCPtr *vstr*, CStrPtr *strp*, StrIValPtr *lenp*);	Finds a substring within a variable string.
`Set`	void	(VStrPtr *vstr*, VStrCPtr *vstr2*);	Copies the contents of `vstr2’ in the vstr
`SetCtStr`	void	(VStrPtr *vstr*, CtCPtr *ct*, NatCStr *str*);	Replaces the contents of a variable string with a copy of an encoded na-tive string.

Function	Returns	Arguments	Description
`SetCtStrSub`	void	(VStrPtr *vstr*, CtCPtr *ct*, NatCStr *str*, StrIVal *slen*);	Replaces the contents of the vstr by a copy of `str’.
`SetNatStr`	void	(VStrPtr *vstr*, NatCStr *str*);	Replaces the contents of a variable string with a copy of a native string.
`SetNatStrSub`	void	(VStrPtr *vstr*, NatCStr *str*, StrIVal *slen*);	Replaces the contents of a variable string with a copy of a native sub-string.
`SetRes`	void	(VCStrPtr *vstr*, CStr *mod*, CStr *res*);	Sets the given string re-source as the contents of a variable string.
`SetStr`	void	(VStrPtr *vstr*, CStr *str*);	Replaces the contents of a variable string with a copy of a string.
`SetStrSub`	void	(VStrPtr *vstr*, CStr *str*, StrIVal *slen*);	Replaces the contents of a variable string with a copy of a substring.
`TruncAt`	void	(VStrPtr *vstr*, StrIVal *pos*);	Truncates a variable string exactly to the length specified.
`Truncate`	void	(VStrPtr *vstr*, StrIVal *pos*);	Truncates a variable string at or before the length specified.