

NAS Software Limited
Incorporating InfoSAR

VS IPL

Quick Reference Guide

VS IPL/Brief [3.20.14]

Release 3.20.14
February 2021

Contents

1	VSIPL Introduction	1
1.1	Introduction to VSIPL	1
1.1.1	Platform Requirements	1
1.1.2	VSIPL Functionality	1
1.1.3	VSIPL Objects	2
1.1.4	Other Features	3
1.2	Basic VSIPL Concepts	3
1.2.1	General Library Design Principles	3
1.2.2	Memory Management	3
1.2.3	Structure of a VSIPL application	6
1.2.4	VSIPL Naming Conventions	7
1.2.5	Non-standard Scalar Data Types	8
1.2.6	Data Array Layout	9
1.2.7	Errors and Restrictions	10
1.3	Implementation-specific Details	10
1.3.1	Types	10
1.3.2	Symbols and Flags	11
1.3.3	Complex Variables	12
1.3.4	Hints	12
1.3.5	Notation	12
2	Getting the Best Performance	13
2.1	Version Information	13
2.2	Memory Alignment	13
2.3	Vector/Matrix Format	13
2.4	Complex Number Format	15
2.5	Error Checking and Debugging	15
2.6	Support Functions	16
2.7	Scalar Functions	16
2.8	Random Number Generation	16
2.9	Vector and Elementwise Operations	16
2.10	Signal Processing Functions	16
2.11	FFT Functions	17
2.12	FIR Filter, Convolution and Correlation Functions	17

2.13	Linear Algebra Functions	17
2.14	Matrix and Vector Operations	17
2.15	LU Decomposition, Cholesky and QRD Functions	18
2.16	Special Linear System Solvers	18
2.17	Controlling the Number of Threads	19
3	Support Functions	20
3.1	Initialization and Finalization	20
	vsip_init	20
	vsip_finalize	20
	Thread_SetParams	20
3.2	Array and Block Functions	20
	vsip_Dblockadmit_P	20
	vsip_blockbind_P	20
	vsip_cblockbind_P	20
	vsip_Dblockcreate_P	21
	vsip_Dblockdestroy_P	21
	vsip_blockfind_P	21
	vsip_cblockfind_P	21
	vsip_blockrebind_P	21
	vsip_cblockrebind_P	22
	vsip_blockrelease_P	22
	vsip_cblockrelease_P	22
	vsip_complete	22
	vsip_cstorage	22
3.3	Vector View Functions	22
	vsip_Dvalldestroy_P	22
	vsip_Dvbind_P	23
	vsip_Dvcloneview_P	23
	vsip_Dvcreate_P	23
	vsip_Dvdestroy_P	23
	vsip_Dvget_P	24
	vsip_Dvgetattrib_P	24
	vsip_Dvgetblock_P	24
	vsip_Dvgetlength_P	24
	vsip_Dvgetoffset_P	25
	vsip_Dvgetstride_P	25
	vsip_vimagview_P	25

vsip_Dvput_P	25
vsip_Dvputattrib_P	25
vsip_Dvputlength_P	26
vsip_Dvputoffset_P	26
vsip_Dvputstride_P	26
vsip_vrealview_P	26
vsip_Dvsubview_P	26
3.4 Matrix View Functions	27
vsip_Dmalldestroy_P	27
vsip_Dmbind_P	27
vsip_Dmcloneview_P	27
vsip_Dmcolview_P	27
vsip_Dmcreate_P	27
vsip_Dmdestroy_P	28
vsip_Dmdiagview_P	28
vsip_Dmget_P	28
vsip_Dmgetattrib_P	28
vsip_Dmgetblock_P	28
vsip_Dmgetcollength_P	29
vsip_Dmgetcolstride_P	29
vsip_Dmgetoffset_P	29
vsip_Dmgetrowlength_P	29
vsip_Dmgetrowstride_P	29
vsip_mimagview_P	30
vsip_Dmput_P	30
vsip_Dmputattrib_P	30
vsip_Dmputcollength_P	30
vsip_Dmputcolstride_P	30
vsip_Dmputoffset_P	31
vsip_Dmputrowlength_P	31
vsip_Dmputrowstride_P	31
vsip_mrealview_P	31
vsip_Dmrowview_P	31
vsip_Dmsubview_P	32
vsip_Dmtransview_P	32
4 Scalar Functions	33
4.1 Real Scalar Functions	33

vsip_acos_f	33
vsip_asin_f	33
vsip_atan_f	33
vsip_atan2_f	33
vsip_ceil_f	33
vsip_cos_f	33
vsip_cosh_f	33
vsip_exp_f	33
vsip_exp10_f	33
vsip_floor_f	33
vsip_fmod_f	33
vsip_hypot_f	33
vsip_log_f	33
vsip_log10_f	33
vsip_mag_f	33
vsip_max_P	33
vsip_min_P	34
vsip_pow_f	34
vsip_rsqrt_f	34
vsip_sin_f	34
vsip_sinh_f	34
vsip_sqrt_f	34
vsip_tan_f	34
vsip_tanh_f	34
4.2 Complex Scalar Functions	34
vsip_arg_f	34
vsip_CADD_f	34
vsip_cadd_f	34
vsip_RCADD_f	34
vsip_rcadd_f	34
vsip_CDIV_f	34
vsip_cdiv_f	35
vsip_CRDIV_f	35
vsip_crdiv_f	35
vsip_CEXP_f	35
vsip_cexp_f	35
vsip_CJMUL_f	35
vsip_cjmul_f	35

vsip_clog_f	35
vsip_CLOG_f	35
vsip_cmag_f	35
vsip_cmagsq_f	35
vsip_CMPLX_f	35
vsip_cmplx_f	35
vsip_CMUL_f	35
vsip_cmul_f	35
vsip_RCMUL_f	36
vsip_rcmul_f	36
vsip_CNEG_f	36
vsip_cneg_f	36
vsip_CONJ_f	36
vsip_conj_f	36
vsip_CRECIP_f	36
vsip_crecip_f	36
vsip_CSQRT_f	36
vsip_csqrt_f	36
vsip_CSUB_f	36
vsip_csub_f	36
vsip_RCSUB_f	36
vsip_rcsub_f	36
vsip_CRSUB_f	36
vsip_crsub_f	37
vsip_imag_f	37
vsip_polar_f	37
vsip_real_f	37
vsip_RECT_f	37
vsip_rect_f	37
4.3 Index Scalar Functions	38
vsip_MATINDEX	38
vsip_matindex	38
vsip_mcolindex	38
vsip_mrowindex	38
5 Random Number Generation	39
5.1 Random Number Functions	39
vsip_randcreate	39

vsip_randdestroy	39
vsip_randu_f	39
vsip_crandu_f	39
vsip_vrandu_f	39
vsip_cvrandu_f	39
vsip_randn_f	39
vsip_crandn_f	39
vsip_vrandn_f	40
vsip_cvrandn_f	40

6 Elementwise Functions 41

6.1 Elementary Mathematical Functions	41
vsip_vacos_f	41
vsip_macos_f	41
vsip_vasin_f	41
vsip_masin_f	41
vsip_vatan_f	41
vsip_matan_f	41
vsip_vatan2_f	41
vsip_matan2_f	41
vsip_vcos_f	41
vsip_mcos_f	41
vsip_vcosh_f	41
vsip_mcosh_f	41
vsip_vexp_f	41
vsip_cvexp_f	42
vsip_mexp_f	42
vsip_cmexp_f	42
vsip_vexp10_f	42
vsip_mexp10_f	42
vsip_vlog_f	42
vsip_cvlog_f	42
vsip_mlog_f	42
vsip_cmlog_f	42
vsip_vlog10_f	42
vsip_mlog10_f	42
vsip_vsin_f	42
vsip_msin_f	42

vsip_vsinh_f	42
vsip_msinh_f	42
vsip_Dvsqrt_P	43
vsip_Dmsqrt_P	43
vsip_vtan_f	43
vsip_mtanh_f	43
vsip_vtanh_f	43
vsip_mtanh_f	43
6.2 Unary Operations	43
vsip_varg_f	43
vsip_marg_f	43
vsip_vceil_f	43
vsip_cvconj_f	43
vsip_cmconj_f	43
vsip_Dvcumsum_P	44
vsip_Dmcumsum_P	44
vsip_veuler_f	44
vsip_meuler_f	44
vsip_vfloor_f	44
vsip_Dvmag_P	44
vsip_Dmmag_P	44
vsip_vcmagsq_f	44
vsip_mcmagsq_f	44
vsip_Dvmeanval_P	44
vsip_Dmmeanval_P	45
vsip_Dvmeansqval_P	45
vsip_Dmmeansqval_P	45
vsip_Dvmodulate_P	45
vsip_Dvneg_P	45
vsip_Dmneg_P	45
vsip_Dvrecip_P	45
vsip_Dmrecip_P	45
vsip_vrsqrt_f	45
vsip_mrsqrt_f	46
vsip_vsq_f	46
vsip_msq_f	46
vsip_Dvsumval_P	46
vsip_Dmsumval_P	46

vsip_vsumsqval_f	46
vsip_msumsqval_f	46
6.3 Binary Operations	46
vsip_Dvadd_P	46
vsip_Dmadd_P	46
vsip_rcvadd_f	47
vsip_rcmadd_f	47
vsip_Dsvadd_P	47
vsip_Dsmadd_P	47
vsip_rscvadd_f	47
vsip_rscmadd_f	47
vsip_Dvdiv_P	47
vsip_Dmdiv_P	47
vsip_rcvdiv_f	47
vsip_rcmdiv_f	47
vsip_crvdiv_f	48
vsip_crmdiv_f	48
vsip_Dsvdiv_P	48
vsip_Dsmdiv_P	48
vsip_rscvdiv_f	48
vsip_rscvsub_f	48
vsip_rscmdiv_f	48
vsip_rscmsub_f	48
vsip_Dvsdiv_P	48
vsip_Dmsdiv_P	48
vsip_Dvexpoavg_P	48
vsip_Dmexpoavg_P	49
vsip_vhypot_f	49
vsip_mhypot_f	49
vsip_cvjmul_f	49
vsip_cmjmul_f	49
vsip_Dvmul_P	49
vsip_Dmmul_P	49
vsip_rcvmul_f	49
vsip_remmul_f	49
vsip_Dsvmul_P	49
vsip_Dsmmul_P	50
vsip_rscvmul_f	50

vsip_rscmmul_f	50
vsip_DvDmmul_P	50
vsip_rvcmmul_f	50
vsip_Dvsub_P	50
vsip_Dmsub_P	50
vsip_crvsub_f	50
vsip_crmsub_f	50
vsip_rcvsub_f	50
vsip_rcmsub_f	51
vsip_Dsvsub_P	51
vsip_Dsmsub_P	51
6.4 Ternary Operations	51
vsip_Dvam_P	51
vsip_Dvma_P	51
vsip_Dvmsa_P	51
vsip_Dvmsb_P	51
vsip_Dvsam_P	51
vsip_Dvsbm_P	52
vsip_Dvsma_P	52
vsip_Dvsmsa_P	52
6.5 Logical Operations	52
vsip_valltrue_bl	52
vsip_malltrue_bl	52
vsip_vanytrue_bl	52
vsip_manytrue_bl	52
vsip_Dvleq_P	52
vsip_Dmleq_P	52
vsip_vlge_P	53
vsip_mlge_P	53
vsip_vlgt_P	53
vsip_mlgt_P	53
vsip_vlle_P	53
vsip_mlle_P	53
vsip_vllt_P	53
vsip_mllt_P	53
vsip_Dvlne_P	54
vsip_Dmlne_P	54
6.6 Selection Operations	54

vsip_vclip_P	54
vsip_mclip_f	54
vsip_vinvclip_P	54
vsip_minvclip_f	54
vsip_vindexbool	54
vsip_vmax_f	55
vsip_mmax_f	55
vsip_vmaxmg_f	55
vsip_mmaxmg_f	55
vsip_vcmaxmgsq_f	55
vsip_mcmmaxmgsq_f	55
vsip_vcmaxmgsqval_f	55
vsip_mcmmaxmgsqval_f	55
vsip_vmaxmgval_f	55
vsip_mmaxmgval_f	55
vsip_vmaxval_f	55
vsip_mmaxval_f	55
vsip_vmin_f	56
vsip_mmin_f	56
vsip_vminmg_f	56
vsip_mminmg_f	56
vsip_vcminmgsq_f	56
vsip_mcmminmgsq_f	56
vsip_vcminmgsqval_f	56
vsip_mcmminmgsqval_f	56
vsip_vminmgval_f	56
vsip_mminmgval_f	56
vsip_vminval_f	56
vsip_mminval_f	56
6.7 Bitwise and Boolean Logical Operators	56
vsip_vand_P	57
vsip_mand_P	57
vsip_vnot_P	57
vsip_mnot_P	57
vsip_vor_P	57
vsip_mor_P	57
vsip_vxor_P	57
vsip_mxor_P	57

6.8	Element Generation and Copy	58
	vsip_Dvcopy_P_P	58
	vsip_Dmcopy_P_P	58
	vsip_Dvfill_P	58
	vsip_Dmfill_P	58
	vsip_vramp_P	59
6.9	Manipulation Operations	59
	vsip_vcplx_f	59
	vsip_mcplx_f	59
	vsip_Dvgather_P	59
	vsip_Dmgather_P	59
	vsip_vimag_f	59
	vsip_mimag_f	59
	vsip_vpolar_f	60
	vsip_mpolar_f	60
	vsip_vreal_f	60
	vsip_mreal_f	60
	vsip_vrect_f	60
	vsip_mrect_f	60
	vsip_Dvscatter_P	60
	vsip_Dmscatter_P	60
	vsip_Dvswap_P	61
	vsip_Dmswap_P	61
6.10	Extensions	61
	vsip_vcsummgval_f	61
7	Signal Processing Functions	62
7.1	FFT Functions	62
	vsip_ccfftip_create_f	62
	vsip_ccfftop_create_f	62
	vsip_crfftop_create_f	62
	vsip_rcfftop_create_f	62
	vsip_fft_destroy_f	62
	vsip_fft_getattr_f	62
	vsip_ccfftip_f	62
	vsip_ccfftop_f	62
	vsip_crfftop_f	62
	vsip_rcfftop_f	62

vsip_ccfftmip_create_f	63
vsip_ccfftmop_create_f	63
vsip_crfftmop_create_f	63
vsip_rcfftmop_create_f	63
vsip_fftm_destroy_f	63
vsip_fftm_getattr_f	63
vsip_ccfftmip_f	63
vsip_ccfftmop_f	63
vsip_crfftmop_f	63
vsip_rcfftmop_f	63
vsip_ccfft2dip_create_f	64
vsip_ccfft2dop_create_f	64
vsip_crfft2dop_create_f	64
vsip_rcfft2dop_create_f	64
vsip_ffft2d_destroy_f	64
vsip_ffft2d_getattr_f	64
vsip_ccfft2dip_f	64
vsip_ccfft2dop_f	64
vsip_crfft2dop_f	64
vsip_rcfft2dop_f	64
7.2 Convolution/Correlation Functions	65
vsip_conv1d_create_f	65
vsip_conv1d_destroy_f	65
vsip_conv1d_getattr_f	65
vsip_convolve1d_f	65
vsip_conv2d_create_f	65
vsip_conv2d_destroy_f	65
vsip_conv2d_getattr_f	65
vsip_convolve2d_f	65
vsip_Dcorr1d_create_P	65
vsip_Dcorr1d_destroy_P	65
vsip_Dcorr1d_getattr_P	66
vsip_Dcorrelate1d_P	66
vsip_Dcorr2d_create_P	66
vsip_Dcorr2d_destroy_P	66
vsip_Dcorr2d_getattr_P	66
vsip_Dcorrelate2d_P	66
7.3 Window Functions	66

	vsip_vcreate_blackman_f	66
	vsip_vcreate_cheby_f	66
	vsip_vcreate_hanning_f	66
	vsip_vcreate_kaiser_f	67
7.4	Filter Functions	67
	vsip_Dfir_create_P	67
	vsip_Dfir_destroy_P	67
	vsip_Dfirfft_P	67
	vsip_Dfir_getattr_P	67
	vsip_Dfir_reset_P	67
7.5	Miscellaneous Signal Processing Functions	67
	vsip_vhisto_f	67
8	Linear Algebra	68
8.1	Matrix and Vector Operations	68
	vsip_cmherm_f	68
	vsip_cvjdot_f	68
	vsip_gemp_f	68
	vsip_cgemp_f	68
	vsip_gems_f	68
	vsip_cgems_f	68
	vsip_Dmprod_P	68
	vsip_cmprodh_P	69
	vsip_cmprodj_P	69
	vsip_Dmprodt_P	69
	vsip_Dmvprod_P	69
	vsip_Dmtrans_P	69
	vsip_Dvdot_P	69
	vsip_Dvmprod_P	70
	vsip_vouter_f	70
	vsip_cvouter_f	70
8.2	Special Linear System Solvers	70
	vsip_covsol_f	70
	vsip_ccovsol_f	70
	vsip_llsqsol_f	70
	vsip_cllsqsol_f	70
	vsip_toepsol_f	70
	vsip_ctoepsol_f	70

8.3	General Square Linear System Solver	71
	vsip_Dlud_P	71
	vsip_Dlud_create_P	71
	vsip_Dlud_destroy_P	71
	vsip_Dlud_getattr_P	71
	vsip_lusol_f	71
	vsip_clusol_f	71
8.4	Symmetric Positive Definite Linear System Solver	71
	vsip_chold_f	71
	vsip_cchold_f	71
	vsip_chold_create_f	71
	vsip_cchold_create_f	71
	vsip_Dchold_destroy_P	72
	vsip_Dchold_getattr_P	72
	vsip_cholsol_f	72
	vsip_ccholsol_f	72
8.5	Overdetermined Linear System Solver	72
	vsip_qrd_f	72
	vsip_cqrd_f	72
	vsip_qrd_create_f	72
	vsip_cqrd_create_f	72
	vsip_Dqrd_destroy_P	72
	vsip_Dqrd_getattr_P	72
	vsip_qrdprodq_f	73
	vsip_cqrdprodq_f	73
	vsip_qrdsolr_f	73
	vsip_cqrdsolr_f	73
	vsip_qrsol_f	73
	vsip_cqrsol_f	73
8.6	Singular Value Decomposition	73
	vsip_Dsvd_P	73
	vsip_Dsvd_create_P	73
	vsip_Dsvd_destroy_P	73
	vsip_Dsvd_getattr_P	74
	vsip_Dsvdprodu_P	74
	vsip_Dsvdprodv_P	74
	vsip_Dsvdmatu_P	74
	vsip_Dsvdmatv_P	74

8.7 Extensions	74
vsip_Dminvlu_P	74
9 Glossary	75

Chapter 1. VSIPL Introduction

1.1 Introduction to VSIPL

The purpose of the Vector, Signal, and Image Processing Library (VSIPL) is to support portable, high performance application programs. The VSIPL specification is based upon existing libraries that have evolved and matured over decades of scientific and engineering computing. A layer of abstraction is added to support portability across diverse memory and processor architectures. The primary design focus of the specification has been embedded signal processing platforms. Enhanced portability of workstation applications is a side benefit.

1.1.1 Platform Requirements

VSIPL was designed so that it could be implemented on a wide variety of hardware. In order to use VSIPL functions on a given platform, a VSIPL compliant library must be available for the particular hardware and a tool-set (ANSI C compiler and linker) available for the operating system.

1.1.2 VSIPL Functionality

The VSIPL specification provides a number of functions to the programmer that support high performance numerical computation on dense rectangular arrays. These are organized in the VSIPL documentation according to category. The available categories include:

- Support
 - Library initialization and finalization
 - Object creation and interaction
 - Memory management
- Basic Scalar Operations
- Basic Vector Operations
- Random Number Generation
- Signal Processing
 - FFT operations
 - Filtering
 - Correlation and convolution
- Linear Algebra

- Basic matrix operations
- Linear system solution
- Least-squares problem solution

Although there are many functions in the VSIPL specification, not all functions are available in all libraries. The contents of a specific VSIPL library subset are defined in a profile. As of the completion of VSIPL 1.0 two profiles have been approved by the VSIPL Forum, referred to as the ‘Core’ and ‘Core Lite’ profiles. The ‘Core’ profile includes most of the signal processing and matrix algebra functionality of the library. The ‘Core Lite’ profile includes a smaller subset, suitable for vector-based signal processing applications. The VSIPL specification defines more functions than are present in either of these profiles.

This library implements the ‘Core’ profile with some extensions.

1.1.3 VSIPL Objects

The main difference between the VSIPL standard and existing libraries is a cleaner encapsulation of memory management through an ‘object-based’ design. In VSIPL, a block can be thought of as a contiguous area of memory for storage of data. A block consists of a data array, which is the memory used for data storage, and a block object, which is an abstract data type which stores information necessary for VSIPL to access the data array. VSIPL allows the user to construct a view of the data in a block as a vector, matrix, or higher dimensional object. A view consists of a block, which contains the data of interest, and a view object, which is an abstract data type that stores information necessary for VSIPL to access the data of interest.

Blocks and views are opaque: they can only be created, accessed and destroyed via library functions. Object data members are private to hide the details of non-portable memory hierarchy management. VSIPL library developers may hide information peculiar to their implementations in the objects in order to prevent the application programmer from accidentally writing code that is neither portable nor compatible.

Data arrays in VSIPL exist in one of two logical data spaces. These are the user data space, and VSIPL data space. VSIPL functions may only operate on data in VSIPL space. User supplied functions may only operate on data in user space. Data may be moved between these logical spaces. Depending on the specific implementation, this move may incur actual data movement penalties or may simply be a bookkeeping procedure. The user should consider the data in VSIPL space to be inaccessible except through VSIPL functions.

1.1.4 Other Features

Two versions of the library are described, referred to as development and performance libraries. These libraries operate to produce identical results with the exception of error reporting and timing. The performance version of the VSIPL library does not provide any error detection or handling except in the case of memory allocation. Other programming errors under a VSIPL performance library may have unpredictable results, up to and including complete system crashes. The development library runs slower than the performance library but includes more error detection capabilities.

1.2 Basic VSIPL Concepts

1.2.1 General Library Design Principles

VSIPL supports high performance numerical computation on dense rectangular arrays, and incorporates the following well-established characteristics of existing scientific and engineering libraries:

1. Elements are stored in one-dimensional data arrays, which appear to the application programmer as a single contiguous block of memory.
2. Data arrays can be viewed as either real or complex, vectors or matrices.
3. All operations on data arrays are performed indirectly through view objects, each of which specifies a particular view of a data array with particular offset, length(s) and stride(s).
4. In general, the application programmer cannot combine operators in a single statement to evaluate expressions. Operators which return a scalar may be combined, but most operators will return a view type or are void and may not be combined.

Operators are restricted to views of a data array that can be specified by an offset, lengths and strides. Views that are more arbitrary are converted into these simple views by functions like `gather` and `back` again by functions like `scatter`. VSIPL does not support triangular or sparse matrices very well, though future extensions might address these. The main difference between the VSIPL and existing libraries is a cleaner encapsulation of the above principles through an ‘object-based’ design. All of the view attributes are encapsulated in opaque objects: such an object can only be created, accessed and destroyed via library functions, which references it via a pointer.

1.2.2 Memory Management

The management of memory is important to efficient algorithm development. This is especially true in embedded systems, many of which are memory limited.

In VSIPL memory management is handled by the implementation. This section describes VSIPL memory management and how the user interacts with VSIPL objects.

Terminology

The terms *user data*, *VSIPL data*, *admitted*, and *released* are used throughout this document when describing memory allocation. It is important that the reader understands the terms that are defined in this section and in the Glossary.

Object Memory Allocation

All objects in VSIPL consist of abstract data types (ADT) that contain attributes defining the underlying data accessed by the object. Certain of the attributes are accessible to the application programmer via access functions; however, there may be any number of attributes assigned by the VSIPL library developer for internal use. Each time an object is defined, memory must be allocated for the ADT. All VSIPL objects are allocated by VSIPL library functions. There is no method by which the application programmer may allocate space for these objects outside of VSIPL. Most VSIPL objects are relatively small and of fixed size; however, some of the objects created for signal processing or linear algebra may allocate large workspaces.

Data Memory Allocation

A data array is an area of memory where data is stored. Data arrays in VSIPL exist in one of two logical data spaces. These are the user data space, and VSIPL data space. VSIPL functions may only operate on data in VSIPL space. User supplied functions may only operate on data in user space. Data may be moved between these logical spaces. Depending on the specific implementation, this move may incur actual data movement penalties or may simply be a bookkeeping procedure. The user should consider the data in VSIPL space to be inaccessible except through VSIPL functions.

A data array allocated by the application, using any method not part of the VSIPL standard, is considered to be a user data array. The application has a pointer to the user data array and knowledge of its type and size. Therefore the application can access a user data array directly using pointers, although it is not always correct to do so.

A data array allocated by a VSIPL function call is referred to as a VSIPL data array. The user has no proper method to retrieve a pointer to such a data array; it may only be accessed via VSIPL function calls.

Users may access data arrays in VSIPL space using an entity referred to as a block. The data array associated with a block is a contiguous series of elements of a given type. There is one block type for each type of data processed by VSIPL.

There are two categories of block: user blocks and VSIPL blocks. A user block is one that has been associated with a user data array. A VSIPL block is one that has been associated with a VSIPL data array. The data array referenced by the block is referred to as being ‘bound’ to the block. The user must provide a pointer to the associated data for a user block. The VSIPL library will allocate space for the data associated with a VSIPL block. Blocks can also be created without any data and then associated with data in user space. The process of associating user space data with a block is called *binding*. A block which does not have data bound to it may not be used, as there is no data to operate on.

A block that has been associated with data may exist in one of two states, admitted and released. The data in an *admitted* block is in the logical VSIPL data space, and the data in a *released* block is in the logical user data space. The process of moving data from the logical VSIPL data space to the logical user data space is called admission; the reverse process is called release.

Data in an admitted block is owned by the VSIPL library, and VSIPL functions operate on this data under the assumption that the data will only be modified using VSIPL functions. VSIPL blocks are always in the admitted state. User blocks may be in an admitted state. User data in an admitted block shall not be operated on except by VSIPL functions. Direct manipulation of user data bound to an admitted block via pointers to the allocated memory is incorrect and may cause erroneous behaviour.

Data in a released block may be accessed by the user, but VSIPL functions should not perform computation on it. User blocks are created in the released state. The block must be admitted to VSIPL before VSIPL functions can operate on the data bound to the block. A user block may be admitted for use by VSIPL and released when direct access to the data is needed by the application program. A VSIPL block may not be released.

Blocks represent logically contiguous data areas in memory (physical layout is undefined for VSIPL space), but users often wish to operate on non-contiguous subsets of these data areas. To provide support for such operations, VSIPL requires that users operate on the data in a block through another object called a view. Views allow the user to specify non-contiguous subsets of a data array and inform VSIPL how the data will be accessed (for example, as a vector or matrix). When creating a vector view, the user specifies an offset into the block, a view length, and a stride value which specifies the number of elements (defined in the type of the block) to advance between each access. Thus, for a block whose corresponding data array contains four elements, a view with an offset value of zero, a stride of two, and a length of two represents a logical data set consisting of members zero and two of the original block. For a matrix view, stride and length parameters are specified in each dimension, and a single offset is specified. By varying the stride, row-major or column-major matrices can be created.

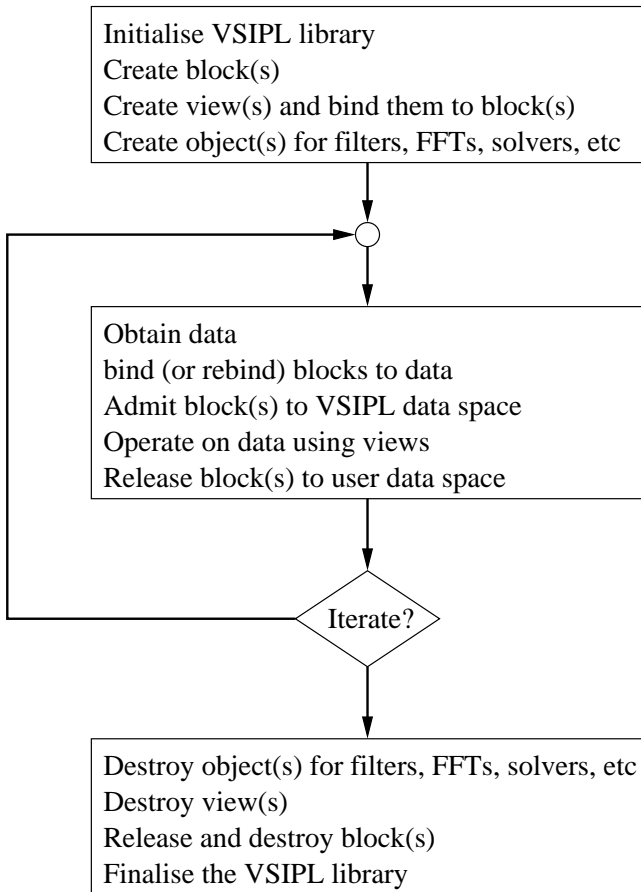
A block may have any number of views created on it; this allows the user to use

vector views to access particular rows or columns of a matrix view, for example. Since the blocks are typed, views are also typed. However, because views also include usage information (e.g. vector or matrix), there are multiple view types for each block type corresponding to how the data will be accessed. These types are immutable; thus for example, a block cannot have both integer and float views associated with it. This would not be useful in any event because the data layout inside VSIPL space is vendor specific.

New views of a block may be created directly using a block object, or indirectly using a previously created view of the block. Except for finding the real or imaginary view of a complex view, all views may be created directly using the block object.

1.2.3 Structure of a VSIPL application

Although there are a number of ways to program an application, the basic VSIPL program consists of the following sequence:



A VSIPL program must initialize the VSIPL library with a call to `vsip_init` before calling any other VSIPL function. Any program that uses VSIPL and that terminates must call `vsip_finalize` before terminating. See the Support chapter for additional conditions and restrictions on these functions.

1.2.4 VSIPL Naming Conventions

While there is nothing to prevent a programmer from writing VSIPL-compatible functions, only those functions that are approved and included in formal VSIPL documentation are a part of VSIPL. Functions outside the standard should not use the VSIPL naming conventions in order to avoid confusion and application porting problems. In particular, names outside of VSIPL should not start with

`vsip` or `vsipl`, in either upper or lower case.

Names for VSIPL types, objects and functions have the following format:

$$\text{vsip_}\langle\text{depth}\rangle\langle\text{shape}\rangle\text{basename_}\langle\text{precision-type}\rangle$$

The shape qualifier specifies scalar, vector or matrix; the depth qualifier specifies real or complex; the precision-type qualifier specifies the data type (boolean, integer, or float) and its precision.

The depth qualifier is `r` or `c` for real or complex, respectively. The real qualifier is understood (not included as part of the name) if there can be no confusion. For a generic name `D` is used to indicate either real or complex.

The shape qualifier is `s`, `v` or `m` for scalar, vector or matrix, respectively. The scalar qualifier is understood (not included as part of the name) if there can be no confusion. For a generic name `S` is used to indicate any shape.

The precision-type qualifier is one of the following:

<code>i</code>	signed integer	<code>bl</code>	boolean
<code>si</code>	short signed integer	<code>vi</code>	vector index
<code>u</code>	unsigned integer	<code>mi</code>	matrix index
<code>f</code>	float		

This qualifier has no default value; it is only omitted on void functions. For a generic name `P` is used to indicate any precision-type.

For example, the generic function `vsip_DSmag_P` takes the magnitude (absolute value) of its argument. Specific instances of this function could include:

<code>vsip_mag_i</code>	(real) (scalar) integer
<code>vsip_cmag_f</code>	complex (scalar) float
<code>vsip_vmag_f</code>	vector of (real) floats
<code>vsip_cvmag_f</code>	vector of complex floats
<code>vsip_mmag_f</code>	matrix of (real) floats
<code>vsip_cmmag_f</code>	matrix of complex floats

For functions with arguments of different depths, shapes or types, the qualifiers may be repeated. For example the copy functions have two precision-type qualifiers corresponding to the data types of the source and destination arrays.

1.2.5 Non-standard Scalar Data Types

In general, VSIPL scalar data types correspond to particular C data types depending on the underlying implementation. However, ANSI C does not define boolean or complex scalar types, both of which are defined in VSIPL. This section summarizes requirements for these data types.

Boolean Data Types

The VSIPL boolean data type (`b1`) is either true or false when used by a VSIPL function which sets or uses the boolean type. If a numeric vector or matrix is copied to a boolean vector or matrix then the value zero is copied to the boolean as false. Any other value is copied as true. If a boolean vector or matrix is copied to a numeric vector or matrix then the value false is copied as a zero, and the value true is copied as positive one. If a VSIPL function returns a boolean scalar then a false is returned as zero and a true is non-zero. If a scalar is tested as boolean using a VSIPL function then a zero is tested as false and a non-zero is tested as true.

Complex Data Types

The definition of the complex scalar is available in public header files, and has the usual structure for complex data as normally defined in ANSI C programs. In general, users are encouraged to not use the structure directly, but to instead use VSIPL scalar functions for manipulating complex scalars. This should enhance portability of user code.

1.2.6 Data Array Layout

A user data array that is bound to a block has a particular required layout, depending on the type of the block. This section describes the required layout of the user data array for various block types. The application programmer must use the data array formats for user data. These formats allow portable input of user data into VSIPL and portable output of VSIPL results to the application.

For basic VSIPL types, the user data array is simply contiguous memory of the corresponding VSIPL type. This applies to floating-point, integer, boolean and vector index types.

For matrix index data, the user data array is contiguous memory of type `vsip_scalar_vi`; each matrix index element is two consecutive elements of type `vsip_scalar_vi`; the first element is the row, the second is the column. Note that the matrix index element in a user data array is not the same as `vsip_scalar_mi`.

For complex float or complex integer data, the user data array is either interleaved or split as described below. Both the interleaved and split formats are supported for user data. Note that the data format for complex user data arrays is not of type `vsip_cscalar_P`.

Interleaved The user data array is contiguous memory of type `vsip_scalar_P`.

The complex element is two consecutive elements of type `vsip_scalar_P`.

The first element is the real component and the second is the imaginary component.

Split The user data array consists of two contiguous memory regions of equal

length, each of type `vsip_scalar_P`. The real and the imaginary regions are determined when the memory is bound to the block. A complex element consists of corresponding elements from the real and imaginary regions.

1.2.7 Errors and Restrictions

Many functions require that their arguments be **conformant**. This means that the objects passed have compatible attributes: for example, size and shape of matrices, lengths of vectors or filter kernels.

If an argument is required to be **valid**, it means:

- a pointer is not `NULL`
- a flag is a member of the required enumerated type
- an object has been initialised and not destroyed.

Errors can occur for the following reasons:

1. an argument is outside the domain for calculation
2. over/underflow during calculation
3. failure to allocate memory
4. algorithm failure because of inappropriate data (as when a matrix does not have full rank)
5. arguments are invalid, out of range, or non-conformant.

Only errors of type 5 are regarded as fatal: in this case, the development version of the library will write a message to `stderr` and call `exit`.

Errors of types 3 and 4 are signalled through the return value of the function. A create function will return `NULL` if the allocation fails; functions with integer return codes use zero to indicate success.

The calling program is not alerted to errors of types 1 and 2.

1.3 Implementation-specific Details

1.3.1 Types

The following VSIPL base types are available:

```

vsip_scalar_i
vsip_scalar_si
vsip_scalar_u
vsip_scalar_bl    vsip_bool
vsip_scalar_vi    vsip_index
vsip_scalar_mi
vsip_scalar_f
vsip_cscalar_i
vsip_cscalar_si
vsip_cscalar_f
vsip_offset
vsip_stride
vsip_length

```

VSIPL also passes information around in abstract data types. These objects are opaque structures (implemented as incomplete typedefs) and they can only be created, accessed, and destroyed with library functions that reference them via a pointer. Some are used to describe the data layout in memory; others store information on filters, matrix decompositions, and so on. Some objects have a ‘get attribute’ function that allows the user access to the internal values.

The following structures for passing data are available:

```

vsip_block_f      vsip_vview_f      vsip_mview_f
vsip_block_i      vsip_vview_i      vsip_mview_i
vsip_block_si     vsip_vview_si     vsip_mview_si
vsip_block_bl     vsip_vview_bl
vsip_block_vi     vsip_vview_vi
vsip_block_mi     vsip_vview_mi
vsip_cblock_f     vsip_cvview_f      vsip_cmview_f
vsip_cblock_i     vsip_cvview_i      vsip_cmview_i
vsip_cblock_si    vsip_cvview_si     vsip_cmview_si

```

1.3.2 Symbols and Flags

The following symbolic constants are defined.

```

VSIP_MIN_SCALAR_F    VSIP_MAX_SCALAR_F
VSIP_MIN_SCALAR_I    VSIP_MAX_SCALAR_I
VSIP_MIN_SCALAR_SI   VSIP_MAX_SCALAR_SI
VSIP_MIN_SCALAR_BL   VSIP_MAX_SCALAR_BL
VSIP_MIN_SCALAR_VI   VSIP_MAX_SCALAR_VI
VSIP_TRUE
VSIP_FALSE
VSIP_PI

```

Other symbols are defined in enumerated types. The valid choices are listed with each function description.

1.3.3 Complex Variables

The preferred storage arrangement for complex data is split.

1.3.4 Hints

VSIPL provides the following mechanisms for the programmer to indicate preferences for optimisation: **they are all ignored** in the current implementation.

- Flags of the enumerated type `vsip_memory_hint` specified when allocating or creating some objects.
- Flags of the enumerated type `vsip_alg_hint` used to indicate whether algorithmic optimisation should minimise execution time, memory use, or maximise numerical accuracy.
- An indication of how many times an object will be used (filters and FFT's have such a parameter).

1.3.5 Notation

The following standard mathematical notation is used in the function descriptions.

$:=$	assignment operator
i	square root of -1
$ x $	absolute value of the real number x
$ z $	modulus of the complex number z
$\lfloor x \rfloor$	floor of the real number x (largest integer less than or equal to x)
$\lceil x \rceil$	ceiling of the real number x (smallest integer greater than or equal to x)
z^*	conjugate of the complex number z
M^T	transpose of the matrix M
M^H	Hermitian (conjugate transpose) of the complex matrix M

Note that in expressions i is always the square root of -1 ; vectors and matrices are indexed with j and k .

An elementwise operation on vectors will be written $C[j] := A[j] + B[j]$. Sometimes, the range of the index variable is not given explicitly; in such cases it is clear from the context that it runs over all the elements in the vectors and that the lengths of the vectors must be equal.

An M by N matrix has M rows and N columns.

Chapter 2. Getting the Best Performance

This section is a short guide for programmers using the NAS VSIPL Library. It contains explanations of library behavior, and tips on selecting the right storage options for your data to increase performance.

2.1 Version Information

Information about the version of the NAS VSIPL library you are using can be found in the comments at the top of the include file `vsip.h`. There is no way that a program can determine the library version at run-time.

2.2 Memory Alignment

The efficiency of many operations is improved if data within memory is correctly aligned on certain word boundaries. This is only of concern for memory allocated outside VSIPL and then bound to a VSIPL block — any storage allocated by the library via a create function is optimally aligned automatically.

Vectors and matrices can be loaded and stored faster if they are vector aligned.

The following table gives the vector alignment and minimum vector length for float data:

Technology	Vector Alignment
SSE	16
AVX	32
AltiVec	16

Alignment can be controlled using a function such as `memalign`. This is a C function that is not in the ANSI standard but is available on many systems. It is defined in `malloc.h` on Linux systems.

The following macro redefines `malloc` so that all memory allocation is optimally aligned:

```
#include <malloc.h>
#define malloc(SIZE) memalign(16, SIZE)
```

Some operating systems (*e.g.* Apple's OSX) automatically align all memory to a 16-byte boundary so `memalign` is not needed.

2.3 Vector/Matrix Format

When available, vector and matrix calculations are done using single instruction, multiple data (SIMD) instructions to process several elements simultaneously. This imposes a minimum vector length given in the table below:

Technology Vector Length (floats)	minimum
SSE	4
AVX	8
Altivec	4

For short ints (16 bits) the vector length should be twice that of the float vector length. If the vector unit supports doubles (64 bits), then the vector length should be half that of floats.

For best performance all input and output vectors should:

- have a stride of 1

Vectors and matrices can be loaded and stored much quicker when they are contiguous in memory. The library includes special optimisations for a stride length of 2 (which was added for interleaved complex numbers), but all other non-unit strides will be significantly slower than a stride of 1 and, in many cases, almost as slow as unvectorised scalar code. Note that, a stride of -1 will also be significantly slower than a stride of $+1$.

- be vector aligned and have a vector/matrix view offset of zero or a multiple of the vector length.

VSIPL vector and matrix views have an option to offset the start of the vector/matrix from the start of the block; for optimal performance this should be zero or a multiple of the vector length otherwise the start of the vector/matrix will not be vector aligned.

- have length greater than or equal to the vector length

The vector unit works on arrays of the vector length so no speed up is gained by using the library on vectors of length less than this.

- have row (row major matrices) or column (column major matrices) length divisible by the vector length

For a row major matrix: if the row length of a matrix is not divisible by the vector length then the alignment of the first element of each row will change for each row/column. For optimal performance the first element of each row should be vector aligned.

If the row length cannot be set to a number divisible by the vector length, a matrix view can be created with a column stride divisible by vector length and greater than the row length. This technique can be used to vector align the first element of every row.

The same rule applies to columns in column major matrices.

- have a length divisible by the vector length

Any elements at the end of the vector which cannot be dealt with by the vector unit must be dealt with in normal scalar code, which will decrease the performance. The decrease in performance becomes less important for longer vectors.

2.4 Complex Number Format

VSIPL supports two storage formats for complex numbers: split and interleaved. Which format you use depends on personal choice or the task being performed.

Split This is the default format in this implementation of VSIPL. It is what is returned when you call a create function. It is also the optimal format to use when calling most NAS VSIPL functions (see Linear Algebra section for some exceptions).

Interleaved To create data blocks in this format, you must allocate the memory yourself and then bind it to a VSIPL complex block. It is the optimal format to use when calling some linear algebra functions.

The internal storage format does not change when you admit or release real or complex data.

2.5 Error Checking and Debugging

Two versions of the NAS VSIPL library are provided: a performance version and a development version. The development version of the library (signified by a ‘D’ in the library’s name) contains full error checking (as specified by the VSIPL standard) and should always be used when developing and debugging applications.

A few library functions return status information: always check the return code of those that do.

The performance version of the library contains no error checking, and consequently runs faster than the development library. The performance library should only be used with applications that have been run successfully with the development version of the library.

When timing code, the performance version of the library should be used.

Note: the performance version of the library reads in data before it knows how much will be used and as a result often reads more data than is needed. This is not a problem, except when using memory checkers such as Electric Fence which object to this behavior. The development library only reads in the data it intends to use and so is safe to use with memory checkers.

2.6 Support Functions

Always call `vsip_init` and `vsip_finalize` at the beginning and end of a program.

Note: For the AltiVec optimized library, calling `vsip_init` will put the AltiVec unit into non-Java mode if it is not already. This speeds up most AltiVec instructions.

See Memory Alignment and Vector/Matrix sections for full information on the optimal creation of blocks and views.

Many of the VSIPL create and bind functions have a memory hint parameter. This parameter is ignored in the current version of the library.

2.7 Scalar Functions

As the vector unit works on arrays of the vector length, scalar functions in the library are not vectorized.

2.8 Random Number Generation

The random number generation functions have not been vectorized in the current version of the library.

2.9 Vector and Elementwise Operations

All vector and elementwise operations work optimally on vectors which match the conditions given in Section 2.3. Complex vectors should be stored split (the default when using VSIPL create functions).

2.10 Signal Processing Functions

All signal processing operations work optimally on vectors which match the conditions given in Section 2.3. Complex vectors should be stored split (the default when using VSIPL create functions).

Most of the signal processing routines are split into three stages:

- a create stage
- a compute stage
- a destroy stage.

The library has been optimized to minimize the time taken to do the compute stage, which means as much precomputation as possible is done in the create stage. If you are using the same signal processing routine on many vectors of the same length, it is far quicker to just create the required signal processing object once and reuse it for each computation stage rather than recreating the object each time it is needed.

2.11 FFT Functions

To get the best performance from an FFT, a vector must have length a multiple of the numbers 2, 4, 8, and 3 only. If a vector length is not a multiple of these numbers, a DFT may be done, which is considerably slower than an FFT. Factors of 3 should be avoided if possible. An FFT will only be done for factors of 3 if the length also has a factor of 16, otherwise a DFT is done.

When doing large FFT's, optimal routines have been developed for the lengths: 4096, 8192, 16384, 32768, and 65536. These lengths should be much quicker than lengths of similar magnitude. In-place FFT's are normally faster than out-of-place FFT's. FFT's are fastest with a scale factor of 1. However, if you need to use a different scale factor, it is better to let the FFT routine do the scaling rather than to do it yourself.

The internal FFT routines only work on vector aligned data with a stride of 1. If vectors are used which do not match these restrictions an internal copy of the vector will be made. This is an important consideration when using large vectors. Also, if complex vectors are not stored split an internal copy will be made.

The current version of the NAS VSIPL library does not have any special FFT routines for doing multiple FFT's, so the time to do n single FFT's will be approximately the same as using the multiple FFT routines on a matrix of n rows.

The `ntimes` parameter to the FFT functions is ignored. The algorithmic hint is only used in the FFT create function: if the `VSIP_ALG_NOISE` hint is used, the FFT create function will take significantly longer. By default, the algorithms are optimized to minimize execution time.

2.12 FIR Filter, Convolution and Correlation Functions

These functions call the FFT functions internally and are therefore subject to the same restrictions.

Hints are ignored with the exception of the internal calling of the FFT create function described in the FFT functions section.

2.13 Linear Algebra Functions

For optimal performance the vectors and matrices used with the linear algebra functions should match the conditions given in Section 2.3. (See also the sections below when using complex LU, complex Cholesky, or complex QRD functions).

2.14 Matrix and Vector Operations

Matrix and vector operations should work optimally on row or column major matrices (row major is the default), however, the restriction exists that all matrices

passed to a function should be of the same order. For example, using two row major matrices as input to a function and a column major as output will be slower than using all row major or all column major. When matrices are passed to NAS VSIPL functions that are not all of the same order, the library will assume they are all row major and treat the column major matrices as strided matrices. (See also the sections below when using LU, Cholesky, or QRD functions).

2.15 LU Decomposition, Cholesky and QRD Functions

These functions have three separate stages:

- a create stage
- a compute stage
- a destroy stage.

The library has been optimized to minimize the time taken to do the compute stage, which means as much precomputation as possible is done in the create stage. If you are using the linear algebra routine on many matrices of the same size, it is far quicker to just create the required linear algebra object once and reuse it for each computation stage rather than recreating the object each time it is needed.

If matrices of different orders or strided matrices are passed to these functions, an internal copy will be made of the entire matrix before the computation is done. This is an important consideration when using large matrices. (Note: unaligned matrices do NOT require internal copying provided they have a stride of one and all matrices used with the functions are of the same order).

COMPLEX: unlike the rest of the NAS VSIPL library, the complex versions of these functions work on INTERLEAVED data. If non-interleaved complex matrices or vectors are passed to these functions, an interleaved internal copy will be created. This is an important consideration when using large matrices.

When using the QRD functions, it is only necessary to save the Q matrix if using the `qrdprodq` function; the `qrsol` and `qrdsolr` do not need the Q matrix.

2.16 Special Linear System Solvers

The `covsol` and `llsqsol` functions internally use the QRD functions and so have the same requirements for optimal performance, including requiring interleaved complex data.

The `toepsol` functions are based on vector operations and so have the same requirements for optimal performance.

2.17 Controlling the Number of Threads

The NAS VSIPL library is multithreaded and will take advantage of multiple cores on the processor invoking it. Utilizing multiple threads is automatic:

- The maximum number of threads used is set when `vsip_init` is called.
- The maximum number running at any one time is also set at that point. If a threaded routine is called with (say) 4 threads and we have hit this maximum number running then four of them are shut down before the new function is executed.
- The number of threads invoked when a routine is called, is decided by that routine by reference to the data (vector or matrix) size specified in the call, to provide the best performance for that call.

It is possible to change the maximum number of threads used.

1. A threaded, and a non-threaded (“serial”) version of the library are provided. If you wish to only ever use one thread in a library call, use the serial version of the library.
2. The maximum number of threads used for a specific function call, and the maximum number kept running at any one time, can be changed by a call to the routine `Thread_SetParams` with arguments `num_threads` and `max_num_running`. This call, if used, *must* be made before the library initialization routine `vsip_init` is called.
3. When calling the routine `Thread_SetParams` the value of `max_num_running` must be greater or equal to $3 * \text{num_threads}$. If the user enters a smaller value than this in their `Thread_SetParams` function call then the function will set the value of `max_num_running` to $3 * \text{num_threads}$.

If no call to `Thread_SetParams` is made, the library default values will be utilized.

Chapter 3. Support Functions

3.1 Initialization and Finalization

Prototype	Description
<pre>int vsip_init(void * ptr);</pre>	Provides initialization, allowing the library to allocate and set a global state, and prepare to support the use of VSIPL functionality by the user.
<pre>int vsip_finalize(void * ptr);</pre>	Provides cleanup and releases resources used by VSIPL (if the last of a nested series of calls), allowing the library to guarantee that any resources allocated by <code>vsip_init</code> are no longer in use after the call is complete.
<pre>void Thread_SetParams(vsip_length num_threads, vsip_length max_num_running);</pre>	Allows the library to allocate a number of threads.

3.2 Array and Block Functions

Prototype	Description
<pre>int vsip_Dblockadmit_P(vsip_Dblock_P * block, vsip_scalar_bl update);</pre>	Admit a data block for VSIPL operations. The following instances are supported: <code>vsip_blockadmit_f</code> <code>vsip_blockadmit_i</code> <code>vsip_blockadmit_si</code> <code>vsip_cblockadmit_f</code> <code>vsip_cblockadmit_i</code> <code>vsip_cblockadmit_si</code> <code>vsip_blockadmit_bl</code> <code>vsip_blockadmit_vi</code> <code>vsip_blockadmit_mi</code>
<pre>vsip_block_P * vsip_blockbind_P(vsip_scalar_P * user_data, vsip_length num_items, vsip_memory_hint hint);</pre>	Create and bind a VSIPL block to a user-allocated data array. The following instances are supported: <code>vsip_blockbind_f</code> <code>vsip_blockbind_i</code> <code>vsip_blockbind_si</code> <code>vsip_blockbind_bl</code> <code>vsip_blockbind_vi</code> <code>vsip_blockbind_mi</code>
<pre>vsip_cblock_P * vsip_cblockbind_P(vsip_scalar_P * user_data1, vsip_scalar_P * user_data2, vsip_length num_items, vsip_memory_hint hint);</pre>	Create and bind a VSIPL complex block to a user-allocated data array. The following instances are supported: <code>vsip_cblockbind_f</code> <code>vsip_cblockbind_i</code> <code>vsip_cblockbind_si</code>

Prototype	Description
<pre>vsip_Dblock_P * vsip_Dblockcreate_P(vsip_length num_items, vsip_memory_hint hint);</pre>	<p>Creates a VSIPL block and binds a (VSIPL-allocated) data array to it. The following instances are supported:</p> <pre>vsip_blockcreate_f vsip_blockcreate_i vsip_blockcreate_si vsip_cblockcreate_f vsip_cblockcreate_i vsip_cblockcreate_si vsip_blockcreate_bl vsip_blockcreate_vi vsip_blockcreate_mi</pre>
<pre>void vsip_Dblockdestroy_P(vsip_Dblock_P * block);</pre>	<p>Destroy a VSIPL block object and any memory allocated for it by VSIPL. The following instances are supported:</p> <pre>vsip_blockdestroy_f vsip_blockdestroy_i vsip_blockdestroy_si vsip_cblockdestroy_f vsip_cblockdestroy_i vsip_cblockdestroy_si vsip_blockdestroy_bl vsip_blockdestroy_vi vsip_blockdestroy_mi</pre>
<pre>vsip_scalar_P * vsip_blockfind_P(const vsip_block_P * block);</pre>	<p>Find the pointer to the data bound to a VSIPL released block object. The following instances are supported:</p> <pre>vsip_blockfind_f vsip_blockfind_i vsip_blockfind_si vsip_blockfind_bl vsip_blockfind_vi vsip_blockfind_mi</pre>
<pre>void vsip_cblockfind_P(const vsip_cblock_P * block, vsip_scalar_P ** user_data1, vsip_scalar_P ** user_data2);</pre>	<p>Find the pointer(s) to the data bound to a VSIPL released complex block object. The following instances are supported:</p> <pre>vsip_cblockfind_f vsip_cblockfind_i vsip_cblockfind_si</pre>
<pre>vsip_scalar_P * vsip_blockrebind_P(vsip_block_P * block, vsip_scalar_P * new_data);</pre>	<p>Rebind a VSIPL block to user-specified data. The following instances are supported:</p> <pre>vsip_blockrebind_f vsip_blockrebind_i vsip_blockrebind_si vsip_blockrebind_bl vsip_blockrebind_vi vsip_blockrebind_mi</pre>

Prototype	Description
<pre>void vsip_cblockrebind_P(vsip_cblock_P * block, vsip_scalar_P * new_data1, vsip_scalar_P * new_data2, vsip_scalar_P ** old_data1, vsip_scalar_P ** old_data2);</pre>	<p>Rebind a VSIPL complex block to user-specified data. The following instances are supported:</p> <p><code>vsip_cblockrebind_f</code> <code>vsip_cblockrebind_i</code> <code>vsip_cblockrebind_si</code></p>
<pre>vsip_scalar_P * vsip_blockrelease_P(vsip_block_P * block, vsip_scalar_bl update);</pre>	<p>Release a VSIPL block for direct user access. The following instances are supported:</p> <p><code>vsip_blockrelease_f</code> <code>vsip_blockrelease_i</code> <code>vsip_blockrelease_si</code> <code>vsip_blockrelease_bl</code> <code>vsip_blockrelease_vi</code> <code>vsip_blockrelease_mi</code></p>
<pre>void vsip_cblockrelease_P(vsip_cblock_P * block, vsip_scalar_bl update, vsip_scalar_P ** user_data1, vsip_scalar_P ** user_data2);</pre>	<p>Release a complex block from VSIPL for direct user access. The following instances are supported:</p> <p><code>vsip_cblockrelease_f</code> <code>vsip_cblockrelease_i</code> <code>vsip_cblockrelease_si</code></p>
<pre>void vsip_complete(void);</pre>	<p>Force all deferred VSIPL execution to complete.</p>
<pre>vsip_cmplx_mem vsip_cstorage(void);</pre>	<p>Returns the preferred complex storage format for the system.</p>

3.3 Vector View Functions

Prototype	Description
<pre>void vsip_Dvalldestroy_P(vsip_Dvview_P * vector);</pre>	<p>Destroy a vector, its associated block, and any VSIPL data array bound to the block. The following instances are supported:</p> <p><code>vsip_valldestroy_f</code> <code>vsip_valldestroy_i</code> <code>vsip_valldestroy_si</code> <code>vsip_cvalldestroy_f</code> <code>vsip_cvalldestroy_i</code> <code>vsip_cvalldestroy_si</code> <code>vsip_valldestroy_bl</code> <code>vsip_valldestroy_vi</code> <code>vsip_valldestroy_mi</code></p>

Prototype	Description
<pre>vsip_Dvview_P * vsip_Dvbind_P(vsip_Dblock_P * block, vsip_offset offset, vsip_stride stride, vsip_length length);</pre>	<p>Create a vector view object and bind it to a block object. The following instances are supported:</p> <pre>vsip_vbind_f vsip_vbind_i vsip_vbind_si vsip_cvbind_f vsip_cvbind_i vsip_cvbind_si vsip_vbind_bl vsip_vbind_vi vsip_vbind_mi</pre>
<pre>vsip_Dvview_P * vsip_Dvcloneview_P(const vsip_Dvview_P * vector);</pre>	<p>Create a clone of a vector view. The following instances are supported:</p> <pre>vsip_vcloneview_f vsip_vcloneview_i vsip_vcloneview_si vsip_cvcloneview_f vsip_cvcloneview_i vsip_cvcloneview_si vsip_vcloneview_bl vsip_vcloneview_vi vsip_vcloneview_mi</pre>
<pre>vsip_Dvview_P * vsip_Dvcreate_P(vsip_length length, vsip_memory_hint hint);</pre>	<p>Creates a block object and a vector view object of the block. The following instances are supported:</p> <pre>vsip_vcreate_f vsip_vcreate_i vsip_vcreate_si vsip_cvcreate_f vsip_cvcreate_i vsip_cvcreate_si vsip_vcreate_bl vsip_vcreate_vi vsip_vcreate_mi</pre>
<pre>vsip_Dblock_P * vsip_Dvdestroy_P(vsip_Dvview_P * vector);</pre>	<p>Destroy a vector view object and return a pointer to the associated block object. The following instances are supported:</p> <pre>vsip_vdestroy_f vsip_vdestroy_i vsip_vdestroy_si vsip_cvdestroy_f vsip_cvdestroy_i vsip_cvdestroy_si vsip_vdestroy_bl vsip_vdestroy_vi vsip_vdestroy_mi</pre>

Prototype	Description
<pre>vsip_Dscalar_P vsip_Dvget_P(const vsip_Dvview_P * vector, vsip_index j);</pre>	<p>Get the value of a specified element of a vector view object. The following instances are supported:</p> <pre>vsip_vget_f vsip_vget_i vsip_vget_si vsip_cvget_f vsip_cvget_i vsip_cvget_si vsip_vget_bl vsip_vget_vi vsip_vget_mi</pre>
<pre>void vsip_Dvgetattrib_P(const vsip_Dvview_P * vector, vsip_Dvattr_P * attr);</pre>	<p>Return the attributes of a vector view object. The following instances are supported:</p> <pre>vsip_vgetattrib_f vsip_vgetattrib_i vsip_vgetattrib_si vsip_cvgetattrib_f vsip_cvgetattrib_i vsip_cvgetattrib_si vsip_vgetattrib_bl vsip_vgetattrib_vi vsip_vgetattrib_mi</pre>
<pre>vsip_Dblock_P * vsip_Dvgetblock_P(const vsip_Dvview_P * vector);</pre>	<p>Get the block attribute of a vector view object. The following instances are supported:</p> <pre>vsip_vgetblock_f vsip_vgetblock_i vsip_vgetblock_si vsip_cvgetblock_f vsip_cvgetblock_i vsip_cvgetblock_si vsip_vgetblock_bl vsip_vgetblock_vi vsip_vgetblock_mi</pre>
<pre>vsip_length vsip_Dvgetlength_P(const vsip_Dvview_P * vector);</pre>	<p>Get the length attribute of a vector view object. The following instances are supported:</p> <pre>vsip_vgetlength_f vsip_vgetlength_i vsip_vgetlength_si vsip_cvgetlength_f vsip_cvgetlength_i vsip_cvgetlength_si vsip_vgetlength_bl vsip_vgetlength_vi vsip_vgetlength_mi</pre>

Prototype	Description
<pre><code>vsip_offset vsip_Dvgetoffset_P(const vsip_Dvview_P * vector);</code></pre>	<p>Get the offset attribute of a vector view object. The following instances are supported:</p> <pre><code>vsip_vgetoffset_f vsip_vgetoffset_i vsip_vgetoffset_si vsip_cvgetoffset_f vsip_cvgetoffset_i vsip_cvgetoffset_si vsip_vgetoffset_bl vsip_vgetoffset_vi vsip_vgetoffset_mi</code></pre>
<pre><code>vsip_stride vsip_Dvgetstride_P(const vsip_Dvview_P * vector);</code></pre>	<p>Get the stride attribute of a vector view object. The following instances are supported:</p> <pre><code>vsip_vgetstride_f vsip_vgetstride_i vsip_vgetstride_si vsip_cvgetstride_f vsip_cvgetstride_i vsip_cvgetstride_si vsip_vgetstride_bl vsip_vgetstride_vi vsip_vgetstride_mi</code></pre>
<pre><code>vsip_vview_P * vsip_vimagview_P(const vsip_cvview_P * complex_vector);</code></pre>	<p>Create a vector view object of the imaginary part of a complex vector from a complex vector view object. The following instances are supported:</p> <pre><code>vsip_vimagview_f vsip_vimagview_i vsip_vimagview_si</code></pre>
<pre><code>void vsip_Dvput_P(vsip_Dvview_P * vector, vsip_index j, vsip_Dscalar_P value);</code></pre>	<p>Set the value of a specified element of a vector view object. The following instances are supported:</p> <pre><code>vsip_vput_f vsip_vput_i vsip_vput_si vsip_cvput_f vsip_cvput_i vsip_cvput_si vsip_vput_bl vsip_vput_vi vsip_vput_mi</code></pre>
<pre><code>vsip_Dvview_P * vsip_Dvputattrib_P(vsip_Dvview_P * ve3tor, const vsip_Dvattr_P * attr);</code></pre>	<p>Set the attributes of a vector view object. The following instances are supported:</p> <pre><code>vsip_vputattrib_f vsip_vputattrib_i vsip_vputattrib_si vsip_cvputattrib_f vsip_cvputattrib_i vsip_cvputattrib_si vsip_vputattrib_bl vsip_vputattrib_vi vsip_vputattrib_mi</code></pre>

Prototype	Description
<pre>vsip_Dvview_P * vsip_Dvputlength_P(vsip_Dvview_P * vector, vsip_length length);</pre>	<p>Set the length attribute of a vector view object. The following instances are supported:</p> <pre>vsip_vputlength_f vsip_vputlength_i vsip_vputlength_si vsip_cvputlength_f vsip_cvputlength_i vsip_cvputlength_si vsip_vputlength_bl vsip_vputlength_vi vsip_vputlength_mi</pre>
<pre>vsip_Dvview_P * vsip_Dvputoffset_P(vsip_Dvview_P * vector, vsip_offset offset);</pre>	<p>Set the offset attribute of a vector view object. The following instances are supported:</p> <pre>vsip_vputoffset_f vsip_vputoffset_i vsip_vputoffset_si vsip_cvputoffset_f vsip_cvputoffset_i vsip_cvputoffset_si vsip_vputoffset_bl vsip_vputoffset_vi vsip_vputoffset_mi</pre>
<pre>vsip_Dvview_P * vsip_Dvputstride_P(vsip_Dvview_P * vector, vsip_stride stride);</pre>	<p>Set the stride attribute of a vector view object. The following instances are supported:</p> <pre>vsip_vputstride_f vsip_vputstride_i vsip_vputstride_si vsip_cvputstride_f vsip_cvputstride_i vsip_cvputstride_si vsip_vputstride_bl vsip_vputstride_vi vsip_vputstride_mi</pre>
<pre>vsip_vview_P * vsip_vrealview_P(const vsip_cvview_P * complex_vector);</pre>	<p>Create a vector view object of the real part of a complex vector from a complex vector view object. The following instances are supported:</p> <pre>vsip_vrealview_f vsip_vrealview_i vsip_vrealview_si</pre>
<pre>vsip_Dvview_P * vsip_Dvsubview_P(const vsip_Dvview_P * vector, vsip_index j, vsip_length n);</pre>	<p>Create a vector view object that is a subview of a vector view object. The following instances are supported:</p> <pre>vsip_vsubview_f vsip_vsubview_i vsip_vsubview_si vsip_cvsubview_f vsip_cvsubview_i vsip_cvsubview_si vsip_vsubview_bl vsip_vsubview_vi vsip_vsubview_mi</pre>

3.4 Matrix View Functions

Prototype	Description
<pre>void vsip_Dmalldestroy_P(vsip_Dmview_P * matrix);</pre>	<p>Destroy a matrix, its associated block, and any VSIPL data array bound to the block.</p> <p>The following instances are supported:</p> <pre>vsip_malldestroy_f vsip_malldestroy_i vsip_malldestroy_si vsip_cmalldestroy_f vsip_cmalldestroy_i vsip_cmalldestroy_si vsip_malldestroy_bl</pre>
<pre>vsip_Dmview_P * vsip_Dmbind_P(vsip_Dblock_P * block, vsip_offset offset, vsip_stride col_stride, vsip_length col_length, vsip_stride row_stride, vsip_length row_length);</pre>	<p>Create a matrix view object and bind it to a block object.</p> <p>The following instances are supported:</p> <pre>vsip_mbind_f vsip_mbind_i vsip_mbind_si vsip_cmbind_f vsip_cmbind_i vsip_cmbind_si vsip_mbind_bl</pre>
<pre>vsip_Dmview_P * vsip_Dmcloneview_P(const vsip_Dmview_P * matrix);</pre>	<p>Create a clone of a matrix view.</p> <p>The following instances are supported:</p> <pre>vsip_mcloneview_f vsip_mcloneview_i vsip_mcloneview_si vsip_cmcloneview_f vsip_cmcloneview_i vsip_cmcloneview_si vsip_mcloneview_bl</pre>
<pre>vsip_Dvview_P * vsip_Dmcolview_P(const vsip_Dmview_P * matrix, vsip_index j);</pre>	<p>Create a vector view object of a specified column of the source matrix view object.</p> <p>The following instances are supported:</p> <pre>vsip_mcolview_f vsip_mcolview_i vsip_mcolview_si vsip_cmcolview_f vsip_cmcolview_i vsip_cmcolview_si vsip_mcolview_bl</pre>
<pre>vsip_Dmview_P * vsip_Dmcreate_P(vsip_length col_length, vsip_length row_length, vsip_major rc, vsip_memory_hint hint);</pre>	<p>Creates a block object and matrix view object of the block.</p> <p>The following instances are supported:</p> <pre>vsip_mcreate_f vsip_mcreate_i vsip_mcreate_si vsip_cmcreate_f vsip_cmcreate_i vsip_cmcreate_si vsip_mcreate_bl</pre>

Prototype	Description
<pre>vsip_Dblock_P * vsip_Dmdestroy_P(vsip_Dmview_P * matrix);</pre>	<p>Destroy a matrix view object and returns a pointer to the associated block object.</p> <p>The following instances are supported:</p> <pre>vsip_mdestroy_f vsip_mdestroy_i vsip_mdestroy_si vsip_cmdestroy_f vsip_cmdestroy_i vsip_cmdestroy_si vsip_mdestroy_bl</pre>
<pre>vsip_Dvview_P * vsip_Dmdiagview_P(const vsip_Dmview_P * matrix, vsip_stride index);</pre>	<p>Create a vector view object of a matrix diagonal of a matrix view object.</p> <p>The following instances are supported:</p> <pre>vsip_mdiagview_f vsip_mdiagview_i vsip_mdiagview_si vsip_cmdiagview_f vsip_cmdiagview_i vsip_cmdiagview_si vsip_mdiagview_bl</pre>
<pre>vsip_Dscalar_P vsip_Dmget_P(const vsip_Dmview_P * matrix, vsip_index i, vsip_index j);</pre>	<p>Get the value of a specified element of a matrix view object.</p> <p>The following instances are supported:</p> <pre>vsip_mget_f vsip_mget_i vsip_mget_si vsip_cmget_f vsip_cmget_i vsip_cmget_si vsip_mget_bl</pre>
<pre>void vsip_Dmgetattrib_P(const vsip_Dmview_P * matrix, vsip_Dmattr_P * attr);</pre>	<p>Get the attributes of a matrix view object.</p> <p>The following instances are supported:</p> <pre>vsip_mgetattrib_f vsip_mgetattrib_i vsip_mgetattrib_si vsip_cmgetattrib_f vsip_cmgetattrib_i vsip_cmgetattrib_si vsip_mgetattrib_bl</pre>
<pre>vsip_Dblock_P * vsip_Dmgetblock_P(const vsip_Dmview_P * matrix);</pre>	<p>Get the block attribute of a matrix view object.</p> <p>The following instances are supported:</p> <pre>vsip_mgetblock_f vsip_mgetblock_i vsip_mgetblock_si vsip_cmgetblock_f vsip_cmgetblock_i vsip_cmgetblock_si vsip_mgetblock_bl</pre>

Prototype	Description
<pre><i>vsip_length</i> vsip_Dmgetcollength_P(const vsip_Dmview_P * matrix);</pre>	<p>Get the column length attribute of a matrix view object. The following instances are supported:</p> <pre>vsip_mgetcollength_f vsip_mgetcollength_i vsip_mgetcollength_si vsip_cmgetcollength_f vsip_cmgetcollength_i vsip_cmgetcollength_si vsip_mgetcollength_bl</pre>
<pre><i>vsip_stride</i> vsip_Dmgetcolstride_P(const vsip_Dmview_P * matrix);</pre>	<p>Get the column stride attribute of a matrix view object. The following instances are supported:</p> <pre>vsip_mgetcolstride_f vsip_mgetcolstride_i vsip_mgetcolstride_si vsip_cmgetcolstride_f vsip_cmgetcolstride_i vsip_cmgetcolstride_si vsip_mgetcolstride_bl</pre>
<pre><i>vsip_offset</i> vsip_Dmgetoffset_P(const vsip_Dmview_P * matrix);</pre>	<p>Get the offset attribute of a matrix view object. The following instances are supported:</p> <pre>vsip_mgetoffset_f vsip_mgetoffset_i vsip_mgetoffset_si vsip_cmgetoffset_f vsip_cmgetoffset_i vsip_cmgetoffset_si vsip_mgetoffset_bl</pre>
<pre><i>vsip_length</i> vsip_Dmgetrowlength_P(const vsip_Dmview_P * matrix);</pre>	<p>Get the row length attribute of a matrix view object. The following instances are supported:</p> <pre>vsip_mgetrowlength_f vsip_mgetrowlength_i vsip_mgetrowlength_si vsip_cmgetrowlength_f vsip_cmgetrowlength_i vsip_cmgetrowlength_si vsip_mgetrowlength_bl</pre>
<pre><i>vsip_stride</i> vsip_Dmgetrowstride_P(const vsip_Dmview_P * matrix);</pre>	<p>Get the row stride attribute of a matrix view object. The following instances are supported:</p> <pre>vsip_mgetrowstride_f vsip_mgetrowstride_i vsip_mgetrowstride_si vsip_cmgetrowstride_f vsip_cmgetrowstride_i vsip_cmgetrowstride_si vsip_mgetrowstride_bl</pre>

Prototype	Description
<pre>vsip_mview_P * vsip_mimagview_P(const vsip_cmview_P * complex_matrix);</pre>	<p>Create a matrix view object of the imaginary part of complex matrix from a complex matrix view object. The following instances are supported:</p> <pre>vsip_mimagview_f vsip_mimagview_i vsip_mimagview_si</pre>
<pre>void vsip_Dmput_P(const vsip_Dmview_P * matrix, vsip_index i, vsip_index j, vsip_Dscalar_P value);</pre>	<p>Set the value of a specified element of a matrix view object. The following instances are supported:</p> <pre>vsip_mput_f vsip_mput_i vsip_mput_si vsip_cmput_f vsip_cmput_i vsip_cmput_si vsip_mput_bl</pre>
<pre>vsip_Dmview_P * vsip_Dmputattrib_P(vsip_Dmview_P * matrix, const vsip_Dmattr_P * attr);</pre>	<p>Set the attributes of a matrix view object. The following instances are supported:</p> <pre>vsip_mputattrib_f vsip_mputattrib_i vsip_mputattrib_si vsip_cmputattrib_f vsip_cmputattrib_i vsip_cmputattrib_si vsip_mputattrib_bl</pre>
<pre>vsip_Dmview_P * vsip_Dmputcollength_P(vsip_Dmview_P * matrix, vsip_length n2);</pre>	<p>Set the column length attribute of a matrix view object. The following instances are supported:</p> <pre>vsip_mputcollength_f vsip_mputcollength_i vsip_mputcollength_si vsip_cmputcollength_f vsip_cmputcollength_i vsip_cmputcollength_si vsip_mputcollength_bl</pre>
<pre>vsip_Dmview_P * vsip_Dmputcolstride_P(vsip_Dmview_P * matrix, vsip_stride s2);</pre>	<p>Set the column stride attribute of a matrix view object. The following instances are supported:</p> <pre>vsip_mputcolstride_f vsip_mputcolstride_i vsip_mputcolstride_si vsip_cmputcolstride_f vsip_cmputcolstride_i vsip_cmputcolstride_si vsip_mputcolstride_bl</pre>

Prototype	Description
<pre>vsip_Dmview_P * vsip_Dmputoffset_P(vsip_Dmview_P * matrix, vsip_offset offset);</pre>	<p>Set the offset attribute of a matrix view object. The following instances are supported:</p> <pre>vsip_mputoffset_f vsip_mputoffset_i vsip_mputoffset_si vsip_cputoffset_f vsip_cputoffset_i vsip_cputoffset_si vsip_mputoffset_bl</pre>
<pre>vsip_Dmview_P * vsip_Dmputrowlength_P(vsip_Dmview_P * matrix, vsip_length n1);</pre>	<p>Set the row length attribute of a matrix view object. The following instances are supported:</p> <pre>vsip_mputrowlength_f vsip_mputrowlength_i vsip_mputrowlength_si vsip_cputrowlength_f vsip_cputrowlength_i vsip_cputrowlength_si vsip_mputrowlength_bl</pre>
<pre>vsip_Dmview_P * vsip_Dmputrowstride_P(vsip_Dmview_P * matrix, vsip_stride s1);</pre>	<p>Set the row stride attribute of a matrix view object. The following instances are supported:</p> <pre>vsip_mputrowstride_f vsip_mputrowstride_i vsip_mputrowstride_si vsip_cputrowstride_f vsip_cputrowstride_i vsip_cputrowstride_si vsip_mputrowstride_bl</pre>
<pre>vsip_mview_P * vsip_mrealview_P(const vsip_cmview_P * complex_matrix);</pre>	<p>Create a matrix view object of the real part of complex matrix from a complex matrix view object. The following instances are supported:</p> <pre>vsip_mrealview_f vsip_mrealview_i vsip_mrealview_si</pre>
<pre>vsip_Dvview_P * vsip_Dmrowview_P(const vsip_Dmview_P * matrix, vsip_index i);</pre>	<p>Create a vector view object of a specified row of the source matrix view object. The following instances are supported:</p> <pre>vsip_mrowview_f vsip_mrowview_i vsip_mrowview_si vsip_cmrowview_f vsip_cmrowview_i vsip_cmrowview_si vsip_mrowview_bl</pre>

Prototype	Description
<pre>vsip_Dmview_P * vsip_Dmsubview_P(const vsip_Dmview_P * matrix, vsip_index i, vsip_index j, vsip_length m, vsip_length n);</pre>	<p>Create a matrix view object that is a subview of matrix view object. The following instances are supported:</p> <pre>vsip_msubview_f vsip_msubview_i vsip_msubview_si vsip_cmsubview_f vsip_cmsubview_i vsip_cmsubview_si vsip_msubview_bl</pre>
<pre>vsip_Dmview_P * vsip_Dmtransview_P(const vsip_Dmview_P * matrix);</pre>	<p>Create a matrix view object that is the transpose of a matrix view object. The following instances are supported:</p> <pre>vsip_mtransview_f vsip_mtransview_i vsip_mtransview_si vsip_cmtransview_f vsip_cmtransview_i vsip_cmtransview_si vsip_mtransview_bl</pre>

Chapter 4. Scalar Functions

4.1 Real Scalar Functions

Prototype	Description
<code>vsip_scalar_f vsip_acos_f(const vsip_scalar_f A);</code>	Computes the principal radian value in $[0, \pi]$ of the inverse cosine of a scalar.
<code>vsip_scalar_f vsip_asin_f(const vsip_scalar_f A);</code>	Computes the principal radian value in $[0, \pi]$ of the inverse sine of a scalar.
<code>vsip_scalar_f vsip_atan_f(const vsip_scalar_f A);</code>	Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent of a scalar.
<code>vsip_scalar_f vsip_atan2_f(const vsip_scalar_f A, const vsip_scalar_f B);</code>	Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of two scalars.
<code>vsip_scalar_f vsip_ceil_f(const vsip_scalar_f A);</code>	Computes the ceiling of a scalar.
<code>vsip_scalar_f vsip_cos_f(const vsip_scalar_f A);</code>	Computes the cosine of a scalar angle in radians.
<code>vsip_scalar_f vsip_cosh_f(const vsip_scalar_f A);</code>	Computes the hyperbolic cosine of a scalar.
<code>vsip_scalar_f vsip_exp_f(const vsip_scalar_f A);</code>	Computes the exponential of a scalar.
<code>vsip_Dscalar_f vsip_exp10_f(const vsip_Dscalar_f A);</code>	Computes the base-10 exponential of a scalar.
<code>vsip_scalar_f vsip_floor_f(const vsip_scalar_f A);</code>	Computes the floor of a scalar.
<code>vsip_scalar_f vsip_fmod_f(const vsip_scalar_f A, const vsip_scalar_f B);</code>	Computes the remainder of the quotient (modulo) of two scalars.
<code>vsip_scalar_f vsip_hypot_f(const vsip_scalar_f A, const vsip_scalar_f B);</code>	Computes the square root of the sum of the squares (hypotenuse) of two scalars.
<code>vsip_scalar_f vsip_log_f(const vsip_scalar_f A);</code>	Computes the natural logarithm of a scalar.
<code>vsip_scalar_f vsip_log10_f(const vsip_scalar_f A);</code>	Computes the base 10 logarithm of a scalar.
<code>vsip_scalar_f vsip_mag_f(const vsip_scalar_f A);</code>	Computes the magnitude (absolute value) of a scalar.
<code>vsip_scalar_P vsip_max_P(const vsip_scalar_P A, const vsip_scalar_P B);</code>	Computes the maximum of two scalars. The following instances are supported: <code>vsip_max_f</code> <code>vsip_max_i</code>

Prototype	Description
<pre>vsip_scalar_P vsip_min_P(const vsip_scalar_P A, const vsip_scalar_P B);</pre>	<p>Computes the minimum of two scalars. The following instances are supported:</p> <pre>vsip_min_f vsip_min_i</pre>
<pre>vsip_scalar_f vsip_pow_f(const vsip_scalar_f A, const vsip_scalar_f B);</pre>	<p>Computes the power function of two scalars.</p>
<pre>vsip_scalar_f vsip_rsqr_f(const vsip_scalar_f A);</pre>	<p>Computes the reciprocal square root of a scalar.</p>
<pre>vsip_scalar_f vsip_sin_f(const vsip_scalar_f A);</pre>	<p>Computes the sine of a scalar angle in radians.</p>
<pre>vsip_scalar_f vsip_sinh_f(const vsip_scalar_f A);</pre>	<p>Computes the hyperbolic sine of a scalar.</p>
<pre>vsip_scalar_f vsip_sqrt_f(const vsip_scalar_f A);</pre>	<p>Computes the square root of a scalar.</p>
<pre>vsip_scalar_f vsip_tan_f(const vsip_scalar_f A);</pre>	<p>Computes the tangent of a scalar angle in radians.</p>
<pre>vsip_scalar_f vsip_tanh_f(const vsip_scalar_f A);</pre>	<p>Computes the hyperbolic tangent of a scalar.</p>

4.2 Complex Scalar Functions

Prototype	Description
<pre>vsip_scalar_f vsip_arg_f(vsip_cscalar_f x);</pre>	<p>Returns the argument in radians $[-\pi, \pi]$ of a complex scalar.</p>
<pre>void vsip_CADD_f(vsip_cscalar_f x, vsip_cscalar_f y, vsip_cscalar_f * z);</pre>	<p>Computes the complex sum of two scalars.</p>
<pre>vsip_cscalar_f vsip_cadd_f(vsip_cscalar_f x, vsip_cscalar_f y);</pre>	<p>Computes the complex sum of two scalars.</p>
<pre>void vsip_RCADD_f(vsip_scalar_f x, vsip_cscalar_f y, vsip_cscalar_f * z);</pre>	<p>Computes the complex sum of two scalars.</p>
<pre>vsip_cscalar_f vsip_rcadd_f(vsip_scalar_f x, vsip_cscalar_f y);</pre>	<p>Computes the complex sum of two scalars.</p>
<pre>void vsip_CDIV_f(vsip_cscalar_f x, vsip_cscalar_f y, vsip_cscalar_f * z);</pre>	<p>Computes the complex quotient of two scalars.</p>

Prototype	Description
<code>vsip_cscalar_f</code> <code>vsip_cdiv_f</code> <code>vsip_cscalar_f x</code> , <code>vsip_cscalar_f y</code>);	Computes the complex quotient of two scalars.
<code>void</code> <code>vsip_CRDIV_f</code> <code>vsip_cscalar_f x</code> , <code>vsip_scalar_f y</code> , <code>vsip_cscalar_f * z</code>);	Computes the complex quotient of two scalars.
<code>vsip_cscalar_f</code> <code>vsip_crdiv_f</code> <code>vsip_cscalar_f x</code> , <code>vsip_scalar_f y</code>);	Computes the complex quotient of two scalars.
<code>void</code> <code>vsip_CEXP_f</code> <code>vsip_cscalar_f x</code> , <code>vsip_cscalar_f * y</code>);	Computes the exponential of a scalar.
<code>vsip_cscalar_f</code> <code>vsip_cexp_f</code> <code>const vsip_cscalar_f A</code>);	Computes the exponential of a scalar.
<code>void</code> <code>vsip_CJMUL_f</code> <code>vsip_cscalar_f x</code> , <code>vsip_cscalar_f y</code> , <code>vsip_cscalar_f * z</code>);	Computes the product a complex scalar with the conjugate of a second complex scalar.
<code>vsip_cscalar_f</code> <code>vsip_cjmul_f</code> <code>vsip_cscalar_f x</code> , <code>vsip_cscalar_f y</code>);	Computes the product a complex scalar with the conjugate of a second complex scalar.
<code>vsip_cscalar_f</code> <code>vsip_clog_f</code> <code>vsip_cscalar_f x</code>);	Computes the complex logarithm of a scalar.
<code>void</code> <code>vsip_CLOG_f</code> <code>vsip_cscalar_f x</code> , <code>vsip_cscalar_f * y</code>);	Computes the complex logarithm of a scalar.
<code>vsip_cscalar_f</code> <code>vsip_cmag_f</code> <code>const vsip_cscalar_f A</code>);	Computes the magnitude (absolute value) of a scalar.
<code>vsip_scalar_f</code> <code>vsip_cmagsq_f</code> <code>vsip_cscalar_f x</code>);	Computes the magnitude squared of a complex scalar.
<code>void</code> <code>vsip_CMPLX_f</code> <code>vsip_scalar_f a</code> , <code>vsip_scalar_f b</code> , <code>vsip_cscalar_f * r</code>);	Form a complex scalar from two real scalars.
<code>vsip_cscalar_f</code> <code>vsip_cmplx_f</code> <code>vsip_scalar_f re</code> , <code>vsip_scalar_f im</code>);	Form a complex scalar from two real scalars.
<code>void</code> <code>vsip_CMUL_f</code> <code>vsip_cscalar_f x</code> , <code>vsip_cscalar_f y</code> , <code>vsip_cscalar_f * z</code>);	Computes the complex product of two scalars.
<code>vsip_cscalar_f</code> <code>vsip_cmul_f</code> <code>vsip_cscalar_f x</code> , <code>vsip_cscalar_f y</code>);	Computes the complex product of two scalars.

Prototype	Description
<code>void vsip_RCMUL_f(vsip_scalar_f x, vsip_cscalar_f y, vsip_cscalar_f * z);</code>	Computes the complex product of two scalars.
<code>vsip_cscalar_f vsip_rcmul_f(vsip_scalar_f x, vsip_cscalar_f y);</code>	Computes the complex product of two scalars.
<code>void vsip_CNEG_f(vsip_cscalar_f x, vsip_cscalar_f * y);</code>	Computes the negation of a complex scalar.
<code>vsip_cscalar_f vsip_cneg_f(vsip_cscalar_f x);</code>	Computes the negation of a complex scalar.
<code>void vsip_CONJ_f(vsip_cscalar_f x, vsip_cscalar_f * y);</code>	Computes the complex conjugate of a scalar.
<code>vsip_cscalar_f vsip_conj_f(vsip_cscalar_f x);</code>	Computes the complex conjugate of a scalar.
<code>void vsip_CRECIP_f(vsip_cscalar_f x, vsip_cscalar_f * y);</code>	Computes the reciprocal of a complex scalar.
<code>vsip_cscalar_f vsip_crecip_f(vsip_cscalar_f x);</code>	Computes the reciprocal of a complex scalar.
<code>void vsip_CSQRT_f(vsip_cscalar_f x, vsip_cscalar_f * y);</code>	Computes the square root a complex scalar.
<code>vsip_cscalar_f vsip_csqrt_f(vsip_cscalar_f x);</code>	Computes the square root a complex scalar.
<code>void vsip_CSUB_f(vsip_cscalar_f x, vsip_cscalar_f y, vsip_cscalar_f * z);</code>	Computes the complex difference of two scalars.
<code>vsip_cscalar_f vsip_csub_f(vsip_cscalar_f x, vsip_cscalar_f y);</code>	Computes the complex difference of two scalars.
<code>void vsip_RCSUB_f(vsip_scalar_f x, vsip_cscalar_f y, vsip_cscalar_f * z);</code>	Computes the complex difference of two scalars.
<code>vsip_cscalar_f vsip_rcsub_f(vsip_scalar_f x, vsip_cscalar_f y);</code>	Computes the complex difference of two scalars.
<code>void vsip_CRSUB_f(vsip_cscalar_f x, vsip_scalar_f y, vsip_cscalar_f * z);</code>	Computes the complex difference of two scalars.

Prototype	Description
<pre>vsip_cscalar_f vsip_crsub_f(vsip_cscalar_f x, vsip_scalar_f y);</pre>	Computes the complex difference of two scalars.
<pre>vsip_scalar_f vsip_imag_f(vsip_cscalar_f x);</pre>	Extract the imaginary part of a complex scalar.
<pre>void vsip_polar_f(vsip_cscalar_f a, vsip_scalar_f * r, vsip_scalar_f * t);</pre>	Convert a complex scalar from rectangular to polar form. The polar data consists of a real scalar containing the radius and a corresponding real scalar containing the argument (angle) of the complex scalar.
<pre>vsip_scalar_f vsip_real_f(vsip_cscalar_f x);</pre>	Extract the real part of a complex scalar.
<pre>void vsip_RECT_f(vsip_scalar_f radius, vsip_scalar_f theta, vsip_cscalar_f * r);</pre>	Convert a pair of real scalars from complex polar to complex rectangular form.
<pre>vsip_cscalar_f vsip_rect_f(vsip_scalar_f r, vsip_scalar_f t);</pre>	Convert a pair of real scalars from complex polar to complex rectangular form.

4.3 Index Scalar Functions

Prototype	Description
<pre>void vsip_MATINDEX(vsip_index r, vsip_index c, vsip_scalar_mi * mi);</pre>	Form a matrix index from two vector indices.
<pre>vsip_scalar_mi vsip_matindex(vsip_index r, vsip_index c);</pre>	Form a matrix index from two vector indices.
<pre>vsip_index vsip_mcolindex(vsip_scalar_mi mi);</pre>	Returns the column vector index from a matrix index.
<pre>vsip_index vsip_mrowindex(vsip_scalar_mi mi);</pre>	Returns the row vector index from a matrix index.

Chapter 5. Random Number Generation

5.1 Random Number Functions

Prototype	Description
<pre>vsip_randstate * vsip_randcreate(const vsip_index seed, const vsip_index numprocs, const vsip_index id, const vsip_rng portable);</pre>	Create a random number generator state object.
<pre>int vsip_randdestroy(vsip_randstate * rand);</pre>	Destroys (frees the memory used by) a random number generator state object. Returns zero on success, non-zero on failure.
<pre>vsip_scalar_f vsip_randu_f(vsip_randstate * state);</pre>	Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.
<pre>vsip_cscalar_f vsip_crandu_f(vsip_randstate * state);</pre>	Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.
<pre>void vsip_vrandu_f(vsip_randstate * state, const vsip_vview_f * R);</pre>	Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.
<pre>void vsip_cvrandu_f(vsip_randstate * state, const vsip_cvview_f * R);</pre>	Generate a uniformly distributed (pseudo-)random number. Floating point values are uniformly distributed over the open interval (0,1). Integer deviates are uniformly distributed over the open interval $(0, 2^{31} - 1)$.
<pre>vsip_scalar_f vsip_randn_f(vsip_randstate * state);</pre>	Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.
<pre>vsip_cscalar_f vsip_crandn_f(vsip_randstate * state);</pre>	Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.

Prototype	Description
<pre>void vsip_vrandn_f(vsip_randstate * state, const vsip_vview_f * R);</pre>	<p>Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.</p>
<pre>void vsip_cvrandn_f(vsip_randstate * state, const vsip_cvview_f * R);</pre>	<p>Generate an approximately normally distributed (pseudo-)random deviate having mean zero and unit variance: $N(0, 1)$. The random numbers are generated by summing values returned by the uniform random number generator.</p>

Chapter 6. Elementwise Functions

6.1 Elementary Mathematical Functions

Prototype	Description
<pre>void vsip_vacos_f(const vsip_vview_f * A, const vsip_vview_f * R);</pre>	Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a vector.
<pre>void vsip_macos_f(const vsip_mview_f * A, const vsip_mview_f * R);</pre>	Computes the principal radian value in $[0, \pi]$ of the inverse cosine for each element of a matrix.
<pre>void vsip_vasin_f(const vsip_vview_f * A, const vsip_vview_f * R);</pre>	Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a vector.
<pre>void vsip_masin_f(const vsip_mview_f * A, const vsip_mview_f * R);</pre>	Computes the principal radian value in $[0, \pi]$ of the inverse sine for each element of a matrix.
<pre>void vsip_vatan_f(const vsip_vview_f * A, const vsip_vview_f * R);</pre>	Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a vector.
<pre>void vsip_matan_f(const vsip_mview_f * A, const vsip_mview_f * R);</pre>	Computes the principal radian value in $[-\pi/2, \pi/2]$ of the inverse tangent for each element of a matrix.
<pre>void vsip_vatan2_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);</pre>	Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input vectors.
<pre>void vsip_matan2_f(const vsip_mview_f * A, const vsip_mview_f * B, const vsip_mview_f * R);</pre>	Computes the four-quadrant radian value in $[-\pi, \pi]$ of the inverse tangent of the ratio of the elements of two input matrices.
<pre>void vsip_vcos_f(const vsip_vview_f * A, const vsip_vview_f * R);</pre>	Computes the cosine for each element of a vector. Element angle values are in radians.
<pre>void vsip_mcos_f(const vsip_mview_f * A, const vsip_mview_f * R);</pre>	Computes the cosine for each element of a matrix. Element angle values are in radians.
<pre>void vsip_vcosh_f(const vsip_vview_f * A, const vsip_vview_f * R);</pre>	Computes the hyperbolic cosine for each element of a vector.
<pre>void vsip_mcosh_f(const vsip_mview_f * A, const vsip_mview_f * R);</pre>	Computes the hyperbolic cosine for each element of a matrix.
<pre>void vsip_vexp_f(const vsip_vview_f * A, const vsip_vview_f * R);</pre>	Computes the exponential function value for each element of a vector.

Prototype	Description
<code>void vsip_cvexp_f(const vsip_cvview_f * A, const vsip_cvview_f * R);</code>	Computes the exponential function value for each element of a vector.
<code>void vsip_mexp_f(const vsip_mview_f * A, const vsip_mview_f * R);</code>	Computes the exponential function value for each element of a matrix.
<code>void vsip_cmexp_f(const vsip_cmview_f * A, const vsip_cmview_f * R);</code>	Computes the exponential function value for each element of a matrix.
<code>void vsip_vexp10_f(const vsip_vview_f * A, const vsip_vview_f * R);</code>	Computes the base 10 exponential for each element of a vector.
<code>void vsip_mexp10_f(const vsip_mview_f * A, const vsip_mview_f * R);</code>	Computes the base 10 exponential for each element of a matrix.
<code>void vsip_vlog_f(const vsip_vview_f * A, const vsip_vview_f * R);</code>	Computes the natural logarithm for each element of a vector.
<code>void vsip_cvlog_f(const vsip_cvview_f * A, const vsip_cvview_f * R);</code>	Computes the natural logarithm for each element of a vector.
<code>void vsip_mlog_f(const vsip_mview_f * A, const vsip_mview_f * R);</code>	Computes the natural logarithm for each element of a matrix.
<code>void vsip_cmlog_f(const vsip_cmview_f * A, const vsip_cmview_f * R);</code>	Computes the natural logarithm for each element of a matrix.
<code>void vsip_vlog10_f(const vsip_vview_f * A, const vsip_vview_f * R);</code>	Compute the base ten logarithm for each element of a vector.
<code>void vsip_mlog10_f(const vsip_mview_f * A, const vsip_mview_f * R);</code>	Compute the base ten logarithm for each element of a matrix.
<code>void vsip_vsin_f(const vsip_vview_f * A, const vsip_vview_f * R);</code>	Compute the sine for each element of a vector. Element angle values are in radians.
<code>void vsip_msin_f(const vsip_mview_f * A, const vsip_mview_f * R);</code>	Compute the sine for each element of a matrix. Element angle values are in radians.
<code>void vsip_vsinh_f(const vsip_vview_f * A, const vsip_vview_f * R);</code>	Computes the hyperbolic sine for each element of a vector.
<code>void vsip_msinh_f(const vsip_mview_f * A, const vsip_mview_f * R);</code>	Computes the hyperbolic sine for each element of a matrix.

Prototype	Description
<pre>void vsip_Dvsqrt_P(const vsip_Dvview_P * A, const vsip_Dvview_P * R);</pre>	<p>Compute the square root for each element of a vector. The following instances are supported:</p> <p><code>vsip_vsqrt_f</code> <code>vsip_cvsqrt_f</code></p>
<pre>void vsip_Dmsqrt_P(const vsip_Dmview_P * A, const vsip_Dmview_P * R);</pre>	<p>Compute the square root for each element of a matrix. The following instances are supported:</p> <p><code>vsip_msqrt_f</code> <code>vsip_cmsqrt_f</code></p>
<pre>void vsip_vtan_f(const vsip_vvview_f * A, const vsip_vvview_f * R);</pre>	<p>Compute the tangent for each element of a vector. Element angle values are in radians.</p>
<pre>void vsip_mtan_f(const vsip_mvview_f * A, const vsip_mvview_f * R);</pre>	<p>Compute the tangent for each element of a matrix. Element angle values are in radians.</p>
<pre>void vsip_vtanh_f(const vsip_vvview_f * A, const vsip_vvview_f * R);</pre>	<p>Computes the hyperbolic tangent for each element of a vector.</p>
<pre>void vsip_mtanh_f(const vsip_mvview_f * A, const vsip_mvview_f * R);</pre>	<p>Computes the hyperbolic tangent for each element of a matrix.</p>

6.2 Unary Operations

Prototype	Description
<pre>void vsip_varg_f(const vsip_cvview_f * A, const vsip_vvview_f * R);</pre>	<p>Computes the argument in radians $[-\pi, \pi]$ for each element of a complex vector.</p>
<pre>void vsip_marg_f(const vsip_cmview_f * A, const vsip_mvview_f * R);</pre>	<p>Computes the argument in radians $[-\pi, \pi]$ for each element of a complex matrix.</p>
<pre>void vsip_vceil_f(const vsip_vvview_f * A, const vsip_vvview_f * R);</pre>	<p>Computes the ceiling for each element of a vector.</p>
<pre>void vsip_cvconj_f(const vsip_cvview_f * A, const vsip_cvview_f * R);</pre>	<p>Compute the conjugate for each element of a complex vector.</p>
<pre>void vsip_cmconj_f(const vsip_cmview_f * A, const vsip_cmview_f * R);</pre>	<p>Compute the conjugate for each element of a complex matrix.</p>

Prototype	Description
<pre>void vsip_Dvcumsum_P(const vsip_Dvview_P * A, const vsip_Dvview_P * R);</pre>	<p>Compute the cumulative sum of the elements of a vector. The following instances are supported:</p> <pre>vsip_vcumsum_f vsip_vcumsum_i vsip_cvcumsum_f vsip_cvcumsum_i</pre>
<pre>void vsip_Dmcumsum_P(vsip_major dir, const vsip_Dmview_P * R);</pre>	<p>Compute the cumulative sums of the elements in the rows or columns of a matrix. The following instances are supported:</p> <pre>vsip_mcumsum_f vsip_mcumsum_i vsip_cmcumsum_f vsip_cmcumsum_i</pre>
<pre>void vsip_veuler_f(const vsip_vview_f * A, const vsip_cvview_f * R);</pre>	<p>Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a vector.</p>
<pre>void vsip_meuler_f(const vsip_mview_f * A, const vsip_cmview_f * R);</pre>	<p>Computes the complex numbers corresponding to the angle of a unit vector in the complex plane for each element of a matrix.</p>
<pre>void vsip_vfloor_f(const vsip_vview_f * A, const vsip_vview_f * R);</pre>	<p>Computes the floor for each element of a vector.</p>
<pre>void vsip_Dvmag_P(const vsip_Dvview_P * A, const vsip_vview_P * R);</pre>	<p>Compute the magnitude for each element of a vector. The following instances are supported:</p> <pre>vsip_vmag_f vsip_vmag_i vsip_vmag_si vsip_cvmag_f</pre>
<pre>void vsip_Dmmag_P(const vsip_Dmview_P * A, const vsip_mview_P * R);</pre>	<p>Compute the magnitude for each element of a matrix. The following instances are supported:</p> <pre>vsip_mmag_f vsip_cmmag_f</pre>
<pre>void vsip_vcmagsq_f(const vsip_cvview_f * A, const vsip_vview_f * R);</pre>	<p>Computes the square of the magnitudes for each element of a vector.</p>
<pre>void vsip_mcmagsq_f(const vsip_cmview_f * A, const vsip_mview_f * R);</pre>	<p>Computes the square of the magnitudes for each element of a matrix.</p>
<pre>vsip_Dscalar_P vsip_Dvmeanval_P(const vsip_Dvview_P * A);</pre>	<p>Returns the mean value of the elements of a vector. The following instances are supported:</p> <pre>vsip_vmeanval_f vsip_cvmeanval_f</pre>

Prototype	Description
<pre>vsip_Dscalar_P vsip_Dmmeanval_P(const vsip_Dmview_P * A);</pre>	<p>Returns the mean value of the elements of a matrix.</p> <p>The following instances are supported:</p> <pre>vsip_mmeanval_f vsip_cmmeanval_f</pre>
<pre>vsip_scalar_P vsip_Dvmeansqval_P(const vsip_Dvview_P * A);</pre>	<p>Returns the mean magnitude squared value of the elements of a vector.</p> <p>The following instances are supported:</p> <pre>vsip_vmeansqval_f vsip_cvmeansqval_f</pre>
<pre>vsip_scalar_P vsip_Dmmeansqval_P(const vsip_Dmview_P * A);</pre>	<p>Returns the mean magnitude squared value of the elements of a matrix.</p> <p>The following instances are supported:</p> <pre>vsip_mmeansqval_f vsip_cmmeansqval_f</pre>
<pre>vsip_scalar_P vsip_Dvmodulate_P(const vsip_Dvview_P * A, const vsip_scalar_P nu, const vsip_scalar_P phi, const vsip_Dvview_P * R);</pre>	<p>Computes the modulation of a real vector by a specified complex frequency.</p> <p>The following instances are supported:</p> <pre>vsip_vmodulate_f vsip_cvmodulate_f</pre>
<pre>void vsip_Dvneg_P(const vsip_Dvview_P * A, const vsip_Dvview_P * R);</pre>	<p>Computes the negation for each element of a vector.</p> <p>The following instances are supported:</p> <pre>vsip_vneg_f vsip_vneg_i vsip_vneg_si vsip_cvneg_f</pre>
<pre>void vsip_Dmneg_P(const vsip_Dmview_P * A, const vsip_Dmview_P * R);</pre>	<p>Computes the negation for each element of a matrix.</p> <p>The following instances are supported:</p> <pre>vsip_mneg_f vsip_mneg_i vsip_cmneg_f</pre>
<pre>void vsip_Dvrecip_P(const vsip_Dvview_P * A, const vsip_Dvview_P * R);</pre>	<p>Computes the reciprocal for each element of a vector.</p> <p>The following instances are supported:</p> <pre>vsip_vrecip_f vsip_cvrecip_f</pre>
<pre>void vsip_Dmrecip_P(const vsip_Dmview_P * A, const vsip_Dmview_P * R);</pre>	<p>Computes the reciprocal for each element of a matrix.</p> <p>The following instances are supported:</p> <pre>vsip_mrecip_f vsip_cmrecip_f</pre>
<pre>void vsip_vrsqrt_f(const vsip_vview_f * A, const vsip_vview_f * R);</pre>	<p>Computes the reciprocal of the square root for each element of a vector.</p>

Prototype	Description
<pre>void vsip_mrsqrt_f(const vsip_mview_f * A, const vsip_mview_f * R);</pre>	Computes the reciprocal of the square root for each element of a matrix.
<pre>void vsip_vsqr_f(const vsip_vview_f * A, const vsip_vview_f * R);</pre>	Computes the square for each element of a vector.
<pre>void vsip_msqr_f(const vsip_mview_f * A, const vsip_mview_f * R);</pre>	Computes the square for each element of a matrix.
<pre>vsip_scalar_P vsip_Dvsumval_P(const vsip_vview_P * A);</pre>	Returns the sum of the elements of a vector. The following instances are supported: <pre>vsip_vsumval_f vsip_vsumval_i vsip_vsumval_si vsip_vsumval_bl vsip_cvsumval_f vsip_cvsumval_i</pre>
<pre>vsip_scalar_P vsip_Dmsumval_P(const vsip_mview_P * A);</pre>	Returns the sum of the elements of a vector. The following instances are supported: <pre>vsip_msumval_f vsip_msumval_i vsip_msumval_bl vsip_cmsumval_f vsip_cmsumval_i</pre>
<pre>vsip_scalar_f vsip_vsumsqval_f(const vsip_vview_f * A);</pre>	Returns the sum of the squares of the elements of a vector.
<pre>vsip_scalar_f vsip_msumsqval_f(const vsip_mview_f * A);</pre>	Returns the sum of the squares of the elements of a matrix.

6.3 Binary Operations

Prototype	Description
<pre>void vsip_Dvadd_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);</pre>	Computes the sum, by element, of two vectors. The following instances are supported: <pre>vsip_vadd_f vsip_vadd_i vsip_vadd_si vsip_cvadd_f vsip_cvadd_i</pre>
<pre>void vsip_Dmadd_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);</pre>	Computes the sum, by element, of two matrices. The following instances are supported: <pre>vsip_madd_f vsip_madd_i vsip_cmadd_f</pre>

Prototype	Description
<pre>void vsip_rcvadd_f(const vsip_vview_f * A, const vsip_cvview_f * B, const vsip_cvview_f * R);</pre>	Computes the sum, by element, of two vectors.
<pre>void vsip_rcmadd_f(const vsip_mview_f * A, const vsip_cmview_f * B, const vsip_cmview_f * R);</pre>	Computes the sum, by element, of two matrices.
<pre>void vsip_Dsvadd_P(const vsip_Dscalar_P a, const vsip_Dvview_P * B, const vsip_Dvview_P * R);</pre>	Computes the sum, by element, of a scalar and a vector. The following instances are supported: <code>vsip_svadd_f</code> <code>vsip_svadd_i</code> <code>vsip_svadd_si</code> <code>vsip_csvadd_f</code>
<pre>void vsip_Dsmadd_P(const vsip_Dscalar_P a, const vsip_Dmview_P * B, const vsip_Dmview_P * R);</pre>	Computes the sum, by element, of a scalar and a matrix. The following instances are supported: <code>vsip_smadd_f</code> <code>vsip_smadd_i</code> <code>vsip_csmadd_f</code>
<pre>void vsip_rscvadd_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);</pre>	Computes the sum, by element, of a real scalar and a complex vector.
<pre>void vsip_rscmadd_f(const vsip_scalar_f a, const vsip_cmview_f * B, const vsip_cmview_f * R);</pre>	Computes the sum, by element, of a real scalar and a complex matrix.
<pre>void vsip_Dvdiv_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);</pre>	Computes the quotient, by element, of two vectors. The following instances are supported: <code>vsip_vdiv_f</code> <code>vsip_cvdiv_f</code>
<pre>void vsip_Dmdiv_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);</pre>	Computes the quotient, by element, of two matrices. The following instances are supported: <code>vsip_mdiv_f</code> <code>vsip_cmdiv_f</code>
<pre>void vsip_rcvdiv_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);</pre>	Computes the quotient, by element, of two vectors.
<pre>void vsip_rcmdiv_f(const vsip_scalar_f a, const vsip_cmview_f * B, const vsip_cmview_f * R);</pre>	Computes the quotient, by element, of two matrices.

Prototype	Description
<pre>void vsip_crdiv_f(const vsip_cvview_f * A, const vsip_vview_f * B, const vsip_cvview_f * R);</pre>	Computes the quotient, by element, of two vectors.
<pre>void vsip_crmdiv_f(const vsip_cmview_f * A, const vsip_mview_f * B, const vsip_cmview_f * R);</pre>	Computes the quotient, by element, of two matrices.
<pre>void vsip_Dsdiv_P(const vsip_Dscalar_P a, const vsip_Dvview_P * B, const vsip_Dvview_P * R);</pre>	Computes the quotient, by element, of a scalar and a vector. The following instances are supported: <code>vsip_sdiv_f</code> <code>vsip_csdiv_f</code>
<pre>void vsip_Dsmdiv_P(const vsip_Dscalar_P a, const vsip_Dmview_P * B, const vsip_Dmview_P * R);</pre>	Computes the quotient, by element, of a scalar and a matrix. The following instances are supported: <code>vsip_smdiv_f</code> <code>vsip_csmdiv_f</code>
<pre>void vsip_rscdiv_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);</pre>	Computes the quotient, by element, of a real scalar and a complex vector.
<pre>void vsip_rscvsub_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);</pre>	Computes the difference, by element, of a real scalar and a complex vector.
<pre>void vsip_rscmdiv_f(const vsip_scalar_f a, const vsip_cmview_f * B, const vsip_cmview_f * R);</pre>	Computes the quotient, by element, of a real scalar and a complex matrix.
<pre>void vsip_rscmsub_f(const vsip_scalar_f a, const vsip_cmview_f * B, const vsip_cmview_f * R);</pre>	Computes the difference, by element, of a real scalar and a complex matrix.
<pre>void vsip_Dvsdiv_P(const vsip_Dvview_P * A, const vsip_Dscalar_P b, const vsip_Dvview_P * R);</pre>	Computes the quotient, by element, of a vector and a scalar. The following instances are supported: <code>vsip_vsdiv_f</code> <code>vsip_cvrsdiv_f</code>
<pre>void vsip_Dmsdiv_P(const vsip_Dmview_P * A, const vsip_Dscalar_P b, const vsip_Dmview_P * R);</pre>	Computes the quotient, by element, of a matrix and a scalar. The following instances are supported: <code>vsip_msdiv_f</code> <code>vsip_cmrsdiv_f</code>
<pre>void vsip_Dvexpoavg_P(const vsip_scalar_P a, const vsip_Dvview_P * B, const vsip_Dvview_P * C);</pre>	Computes an exponential weighted average, by element, of two vectors. The following instances are supported: <code>vsip_vexpoavg_f</code> <code>vsip_cvexpoavg_f</code>

Prototype	Description
<pre>void vsip_Dmexpoavg_P(const vsip_scalar_P a, const vsip_Dmview_P * B, const vsip_Dmview_P * C);</pre>	<p>Computes an exponential weighted average, by element, of two matrices. The following instances are supported:</p> <p><code>vsip_mexpoavg_f</code> <code>vsip_cmexpoavg_f</code></p>
<pre>void vsip_vhypot_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);</pre>	<p>Computes the square root of the sum of squares, by element, of two input vectors.</p>
<pre>void vsip_mhypot_f(const vsip_mview_f * A, const vsip_mview_f * B, const vsip_mview_f * R);</pre>	<p>Computes the square root of the sum of squares, by element, of two input matrices.</p>
<pre>void vsip_cvjmul_f(const vsip_cvview_f * A, const vsip_cvview_f * B, const vsip_cvview_f * R);</pre>	<p>Computes the product of a complex vector with the conjugate of a second complex vector, by element.</p>
<pre>void vsip_cmjmul_f(const vsip_cmview_f * A, const vsip_cmview_f * B, const vsip_cmview_f * R);</pre>	<p>Computes the product of a complex matrix with the conjugate of a second complex matrix, by element.</p>
<pre>void vsip_Dvmul_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);</pre>	<p>Computes the product, by element, of two vectors. The following instances are supported:</p> <p><code>vsip_vmul_f</code> <code>vsip_vmul_i</code> <code>vsip_vmul_si</code> <code>vsip_cvmul_f</code></p>
<pre>void vsip_Dmmul_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);</pre>	<p>Computes the product, by element, of two matrices. The following instances are supported:</p> <p><code>vsip_mmul_f</code> <code>vsip_mmul_i</code> <code>vsip_cmmul_f</code></p>
<pre>void vsip_rcvmul_f(const vsip_vview_f * A, const vsip_cvview_f * B, const vsip_cvview_f * R);</pre>	<p>Computes the product, by element, of two vectors.</p>
<pre>void vsip_rcmmul_f(const vsip_mview_f * A, const vsip_cmview_f * B, const vsip_cmview_f * R);</pre>	<p>Computes the product, by element, of two matrices.</p>
<pre>void vsip_Dsvmul_P(const vsip_Dscalar_P a, const vsip_Dvview_P * B, const vsip_Dvview_P * R);</pre>	<p>Computes the product, by element, of a scalar and a vector. The following instances are supported:</p> <p><code>vsip_svmul_f</code> <code>vsip_svmul_i</code> <code>vsip_svmul_si</code> <code>vsip_csvmul_f</code></p>

Prototype	Description
<pre>void vsip_Dsmmul_P(const vsip_Dscalar_P a, const vsip_Dmview_P * B, const vsip_Dmview_P * R);</pre>	<p>Computes the product, by element, of a scalar and a matrix. The following instances are supported:</p> <p><code>vsip_smmul_f</code> <code>vsip_csmmul_f</code></p>
<pre>void vsip_rscvmul_f(const vsip_scalar_f a, const vsip_cvview_f * B, const vsip_cvview_f * R);</pre>	<p>Computes the product, by element, of a real scalar and a complex vector.</p>
<pre>void vsip_rscmmul_f(const vsip_scalar_f a, const vsip_cmview_f * B, const vsip_cmview_f * R);</pre>	<p>Computes the product, by element, of a real scalar and a complex matrix.</p>
<pre>void vsip_DvDmmul_P(const vsip_Dvview_P * A, const vsip_Dmview_P * B, const vsip_major major, const vsip_Dmview_P * R);</pre>	<p>Computes the product, by element, of a vector and the rows or columns of a matrix. The following instances are supported:</p> <p><code>vsip_vmmul_f</code> <code>vsip_cvmmul_f</code></p>
<pre>void vsip_rvcmmul_f(const vsip_vview_f * A, const vsip_cmview_f * B, const vsip_major major, const vsip_cmview_f * R);</pre>	<p>Computes the product, by element, of a vector and the rows or columns of a matrix.</p>
<pre>void vsip_Dvsub_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * R);</pre>	<p>Computes the difference, by element, of two vectors. The following instances are supported:</p> <p><code>vsip_vsub_f</code> <code>vsip_vsub_i</code> <code>vsip_vsub_si</code> <code>vsip_cvsub_f</code></p>
<pre>void vsip_Dmsub_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);</pre>	<p>Computes the difference, by element, of two matrices. The following instances are supported:</p> <p><code>vsip_msub_f</code> <code>vsip_msub_i</code> <code>vsip_cmsub_f</code></p>
<pre>void vsip_crvsb_f(const vsip_cvview_f * A, const vsip_vview_f * B, const vsip_cvview_f * R);</pre>	<p>Computes the difference, by element, of two vectors.</p>
<pre>void vsip_crmsub_f(const vsip_cmview_f * A, const vsip_mview_f * B, const vsip_cmview_f * R);</pre>	<p>Computes the difference, by element, of two matrices.</p>
<pre>void vsip_rcvsub_f(const vsip_vview_f * A, const vsip_cvview_f * B, const vsip_cvview_f * R);</pre>	<p>Computes the difference, by element, of two vectors.</p>

Prototype	Description
<pre>void vsip_rcmsub_f(const vsip_mview_f * A, const vsip_cmview_f * B, const vsip_cmview_f * R);</pre>	<p>Computes the difference, by element, of two matrices.</p>
<pre>void vsip_Dsvsub_P(const vsip_Dscalar_P a, const vsip_Dvview_P * B, const vsip_Dvview_P * R);</pre>	<p>Computes the difference, by element, of a scalar and a vector. The following instances are supported:</p> <pre>vsip_svsub_f vsip_svsub_i vsip_svsub_si vsip_csvsub_f</pre>
<pre>void vsip_Dmsub_P(const vsip_Dscalar_P a, const vsip_Dmview_P * B, const vsip_Dmview_P * R);</pre>	<p>Computes the difference, by element, of a scalar and a matrix. The following instances are supported:</p> <pre>vsip_smsub_f vsip_smsub_i vsip_csmsub_f</pre>

6.4 Ternary Operations

Prototype	Description
<pre>void vsip_Dvam_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * C, const vsip_Dvview_P * R);</pre>	<p>Computes the sum of two vectors and product of a third vector, by element. The following instances are supported:</p> <pre>vsip_vam_f vsip_cvam_f</pre>
<pre>void vsip_Dvma_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * C, const vsip_Dvview_P * R);</pre>	<p>Computes the product of two vectors and sum of a third vector, by element. The following instances are supported:</p> <pre>vsip_vma_f vsip_cvma_f</pre>
<pre>void vsip_Dvmsa_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dscalar_P c, const vsip_Dvview_P * R);</pre>	<p>Computes the product of two vectors and sum of a scalar, by element. The following instances are supported:</p> <pre>vsip_vmsa_f vsip_cvmsa_f</pre>
<pre>void vsip_Dvmsb_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * C, const vsip_Dvview_P * R);</pre>	<p>Computes the product of two vectors and difference of a third vector, by element. The following instances are supported:</p> <pre>vsip_vmsb_f vsip_cvmsb_f</pre>
<pre>void vsip_Dvsam_P(const vsip_Dvview_P * A, const vsip_Dscalar_P b, const vsip_Dvview_P * C, const vsip_Dvview_P * R);</pre>	<p>Computes the sum of a vector and a scalar, and product with a second vector, by element. The following instances are supported:</p> <pre>vsip_vsam_f vsip_cvsam_f</pre>

Prototype	Description
<pre>void vsip_Dvsbm_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_Dvview_P * C, const vsip_Dvview_P * R);</pre>	<p>Computes the difference of two vectors, and product with a third vector, by element.</p> <p>The following instances are supported:</p> <pre>vsip_vsbm_f vsip_cvsbm_f</pre>
<pre>void vsip_Dvsma_P(const vsip_Dvview_P * A, const vsip_Dscalar_P b, const vsip_Dvview_P * C, const vsip_Dvview_P * R);</pre>	<p>Computes the product of a vector and a scalar, and sum with a second vector, by element.</p> <p>The following instances are supported:</p> <pre>vsip_vsma_f vsip_cvsm_a_f</pre>
<pre>void vsip_Dvsmsa_P(const vsip_Dvview_P * A, const vsip_Dscalar_P b, const vsip_Dscalar_P c, const vsip_Dvview_P * R);</pre>	<p>Computes the product of a vector and a scalar, and sum with a second scalar, by element.</p> <p>The following instances are supported:</p> <pre>vsip_vsmsa_f vsip_cvmsa_f</pre>

6.5 Logical Operations

Prototype	Description
<pre>vsip_scalar_bl vsip_valltrue_bl(const vsip_vview_bl * A);</pre>	<p>Returns true if all the elements of a vector are true.</p>
<pre>vsip_scalar_bl vsip_malltrue_bl(const vsip_vview_bl * A);</pre>	<p>Returns true if all the elements of a vector are true.</p>
<pre>vsip_scalar_bl vsip_vanytrue_bl(const vsip_vview_bl * A);</pre>	<p>Returns true if one or more elements of a vector are true.</p>
<pre>vsip_scalar_bl vsip_manytrue_bl(const vsip_vview_bl * A);</pre>	<p>Returns true if one or more elements of a vector are true.</p>
<pre>void vsip_Dvleq_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of 'equal', by element, of two vectors.</p> <p>The following instances are supported:</p> <pre>vsip_vleq_f vsip_vleq_i vsip_cvleq_f vsip_cvleq_i</pre>
<pre>void vsip_Dmleq_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of 'equal', by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <pre>vsip_mleq_f vsip_mleq_i vsip_cmleq_f vsip_cmleq_i</pre>

Prototype	Description
<pre>void vsip_vlge_P(const vsip_vview_P * A, const vsip_vview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of ‘greater than or equal’, by element, of two vectors.</p> <p>The following instances are supported:</p> <p>vsip_vlge_f vsip_vlge_i</p>
<pre>void vsip_mlge_P(const vsip_mview_P * A, const vsip_mview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of ‘greater than or equal’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <p>vsip_mlge_f vsip_mlge_i</p>
<pre>void vsip_vlgt_P(const vsip_vview_P * A, const vsip_vview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of ‘greater than’, by element, of two vectors.</p> <p>The following instances are supported:</p> <p>vsip_vlgt_f vsip_vlgt_i</p>
<pre>void vsip_mlgt_P(const vsip_mview_P * A, const vsip_mview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of ‘greater than’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <p>vsip_mlgt_f vsip_mlgt_i</p>
<pre>void vsip_vlle_P(const vsip_vview_P * A, const vsip_vview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of ‘less than or equal’, by element, of two vectors.</p> <p>The following instances are supported:</p> <p>vsip_vlle_f vsip_vlle_i</p>
<pre>void vsip_mlle_P(const vsip_mview_P * A, const vsip_mview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of ‘less than or equal’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <p>vsip_mlle_f vsip_mlle_i</p>
<pre>void vsip_vllt_P(const vsip_vview_P * A, const vsip_vview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of ‘less than’, by element, of two vectors.</p> <p>The following instances are supported:</p> <p>vsip_vllt_f vsip_vllt_i</p>
<pre>void vsip_mllt_P(const vsip_mview_P * A, const vsip_mview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of ‘less than’, by element, of two vectors/matrices.</p> <p>The following instances are supported:</p> <p>vsip_mllt_f vsip_mllt_i</p>

Prototype	Description
<pre>void vsip_Dvln_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of ‘not equal’, by element, of two vectors. The following instances are supported:</p> <p><code>vsip_vln_f</code> <code>vsip_vln_i</code> <code>vsip_cvln_f</code> <code>vsip_cvln_i</code></p>
<pre>void vsip_Dmln_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_vview_bl * R);</pre>	<p>Computes the boolean comparison of ‘not equal’, by element, of two vectors/matrices. The following instances are supported:</p> <p><code>vsip_mln_f</code> <code>vsip_mln_i</code> <code>vsip_cmln_f</code> <code>vsip_cmln_i</code></p>

6.6 Selection Operations

Prototype	Description
<pre>void vsip_vclip_P(const vsip_vview_P * A, const vsip_scalar_P t1, const vsip_scalar_P t2, const vsip_scalar_P c1, const vsip_scalar_P c2, const vsip_vview_P * R);</pre>	<p>Computes the generalised double clip, by element, of two vectors. The following instances are supported:</p> <p><code>vsip_vclip_f</code> <code>vsip_vclip_i</code> <code>vsip_vclip_si</code></p>
<pre>void vsip_mclip_f(const vsip_mview_f * A, const vsip_scalar_f t1, const vsip_scalar_f t2, const vsip_scalar_f c1, const vsip_scalar_f c2, const vsip_mview_f * R);</pre>	<p>Computes the generalised double clip, by element, of two matrices.</p>
<pre>void vsip_vinvclip_P(const vsip_vview_P * A, const vsip_scalar_P t1, const vsip_scalar_P t2, const vsip_scalar_P t3, const vsip_scalar_P c1, const vsip_scalar_P c2, const vsip_vview_P * R);</pre>	<p>Computes the generalised inverted double clip, by element, of two vectors. The following instances are supported:</p> <p><code>vsip_vinvclip_f</code> <code>vsip_vinvclip_i</code> <code>vsip_vinvclip_si</code></p>
<pre>void vsip_minvclip_f(const vsip_mview_f * A, const vsip_scalar_f t1, const vsip_scalar_f t2, const vsip_scalar_f t3, const vsip_scalar_f c1, const vsip_scalar_f c2, const vsip_mview_f * R);</pre>	<p>Computes the generalised inverted double clip, by element, of two matrices.</p>
<pre>vsip_length vsip_vindexbool(const vsip_vview_bl * X, vsip_vview_vi * Y);</pre>	<p>Computes an index vector of the indices of the non-false elements of the boolean vector, and returns the number of non-false elements.</p>

Prototype	Description
<pre>void vsip_vmax_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);</pre>	Computes the maximum, by element, of two vectors.
<pre>void vsip_mmax_f(const vsip_mview_f * A, const vsip_mview_f * B, const vsip_mview_f * R);</pre>	Computes the maximum, by element, of two matrices.
<pre>void vsip_vmaxmg_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);</pre>	Computes the maximum magnitude (absolute value), by element, of two vectors.
<pre>void vsip_mmaxmg_f(const vsip_mview_f * A, const vsip_mview_f * B, const vsip_mview_f * R);</pre>	Computes the maximum magnitude (absolute value), by element, of two matrices.
<pre>void vsip_vcmaxmgsq_f(const vsip_cvview_f * A, const vsip_cvview_f * B, const vsip_vview_f * R);</pre>	Computes the maximum magnitude squared, by element, of two complex vectors.
<pre>void vsip_mcmaxmgsq_f(const vsip_cmview_f * A, const vsip_cmview_f * B, const vsip_mview_f * R);</pre>	Computes the maximum magnitude squared, by element, of two complex matrices.
<pre>vsip_scalar_f vsip_vcmaxmgsqval_f(const vsip_cvview_f * A, vsip_index * index);</pre>	Returns the index and value of the maximum magnitude squared of the elements of a complex vector. The index is returned by reference as one of the arguments.
<pre>vsip_scalar_f vsip_mcmaxmgsqval_f(const vsip_cmview_f * A, vsip_index * index);</pre>	Returns the index and value of the maximum magnitude squared of the elements of a complex matrix. The index is returned by reference as one of the arguments.
<pre>vsip_scalar_f vsip_vmaxmgval_f(const vsip_vview_f * A, vsip_index * index);</pre>	Returns the index and value of the maximum absolute value of the elements of a vector. The index is returned by reference as one of the arguments.
<pre>vsip_scalar_f vsip_mmaxmgval_f(const vsip_mview_f * A, vsip_index * index);</pre>	Returns the index and value of the maximum absolute value of the elements of a matrix. The index is returned by reference as one of the arguments.
<pre>vsip_scalar_f vsip_vmaxval_f(const vsip_vview_f * A, vsip_index * index);</pre>	Returns the index and value of the maximum value of the elements of a vector. The index is returned by reference as one of the arguments.
<pre>vsip_scalar_f vsip_mmaxval_f(const vsip_mview_f * A, vsip_index * index);</pre>	Returns the index and value of the maximum value of the elements of a matrix. The index is returned by reference as one of the arguments.

Prototype	Description
<pre>void vsip_vmin_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);</pre>	Computes the minimum, by element, of two vectors.
<pre>void vsip_mmin_f(const vsip_mview_f * A, const vsip_mview_f * B, const vsip_mview_f * R);</pre>	Computes the minimum, by element, of two matrices.
<pre>void vsip_vminmg_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_vview_f * R);</pre>	Computes the minimum magnitude (absolute value), by element, of two vectors.
<pre>void vsip_mminmg_f(const vsip_mview_f * A, const vsip_mview_f * B, const vsip_mview_f * R);</pre>	Computes the minimum magnitude (absolute value), by element, of two matrices.
<pre>void vsip_vcminmgsq_f(const vsip_cvview_f * A, const vsip_cvview_f * B, const vsip_vview_f * R);</pre>	Computes the minimum magnitude squared, by element, of two complex vectors.
<pre>void vsip_mcmminmgsq_f(const vsip_cmview_f * A, const vsip_cmview_f * B, const vsip_mview_f * R);</pre>	Computes the minimum magnitude squared, by element, of two complex matrices.
<pre>vsip_scalar_f vsip_vcminmgsqval_f(const vsip_cvview_f * A, vsip_index * index);</pre>	Returns the index and value of the minimum magnitude squared of the elements of a complex vector. The index is returned by reference as one of the arguments.
<pre>vsip_scalar_f vsip_mcmminmgsqval_f(const vsip_cmview_f * A, vsip_index * index);</pre>	Returns the index and value of the minimum magnitude squared of the elements of a complex matrix. The index is returned by reference as one of the arguments.
<pre>vsip_scalar_f vsip_vminmgval_f(const vsip_vview_f * A, vsip_index * index);</pre>	Returns the index and value of the minimum absolute value of the elements of a vector. The index is returned by reference as one of the arguments.
<pre>vsip_scalar_f vsip_mminmgval_f(const vsip_mview_f * A, vsip_index * index);</pre>	Returns the index and value of the minimum absolute value of the elements of a matrix. The index is returned by reference as one of the arguments.
<pre>vsip_scalar_f vsip_vminval_f(const vsip_vview_f * A, vsip_index * index);</pre>	Returns the index and value of the minimum value of the elements of a vector. The index is returned by reference as one of the arguments.
<pre>vsip_scalar_f vsip_mminval_f(const vsip_mview_f * A, vsip_index * index);</pre>	Returns the index and value of the minimum value of the elements of a matrix. The index is returned by reference as one of the arguments.

6.7 Bitwise and Boolean Logical Operators

Prototype	Description
<pre>void vsip_vand_P(const vsip_vview_P * A, const vsip_vview_P * B, const vsip_vview_P * R);</pre>	<p>Computes the bitwise and, by element, of two vectors. The following instances are supported:</p> <p><code>vsip_vand_i</code> <code>vsip_vand_si</code> <code>vsip_vand_bl</code></p>
<pre>void vsip_mand_P(const vsip_mview_P * A, const vsip_mview_P * B, const vsip_mview_P * R);</pre>	<p>Computes the bitwise and, by element, of two matrices. The following instances are supported:</p> <p><code>vsip_mand_i</code> <code>vsip_mand_si</code> <code>vsip_mand_bl</code></p>
<pre>void vsip_vnot_P(const vsip_vview_P * A, const vsip_vview_P * R);</pre>	<p>Computes the bitwise not (one's complement), by element, of two vectors. The following instances are supported:</p> <p><code>vsip_vnot_i</code> <code>vsip_vnot_si</code> <code>vsip_vnot_bl</code></p>
<pre>void vsip_mnot_P(const vsip_mview_P * A, const vsip_mview_P * R);</pre>	<p>Computes the bitwise not (one's complement), by element, of two matrices. The following instances are supported:</p> <p><code>vsip_mnot_i</code> <code>vsip_mnot_si</code> <code>vsip_mnot_bl</code></p>
<pre>void vsip_vor_P(const vsip_vview_P * A, const vsip_vview_P * B, const vsip_vview_P * R);</pre>	<p>Computes the bitwise inclusive or, by element, of two vectors. The following instances are supported:</p> <p><code>vsip_vor_i</code> <code>vsip_vor_si</code> <code>vsip_vor_bl</code></p>
<pre>void vsip_mor_P(const vsip_mview_P * A, const vsip_mview_P * B, const vsip_mview_P * R);</pre>	<p>Computes the bitwise inclusive or, by element, of two matrices. The following instances are supported:</p> <p><code>vsip_mor_i</code> <code>vsip_mor_si</code> <code>vsip_mor_bl</code></p>
<pre>void vsip_vxor_P(const vsip_vview_P * A, const vsip_vview_P * B, const vsip_vview_P * R);</pre>	<p>Computes the bitwise exclusive or, by element, of two vectors. The following instances are supported:</p> <p><code>vsip_vxor_i</code> <code>vsip_vxor_si</code> <code>vsip_vxor_bl</code></p>
<pre>void vsip_mxor_P(const vsip_mview_P * A, const vsip_mview_P * B, const vsip_mview_P * R);</pre>	<p>Computes the bitwise exclusive or, by element, of two matrices. The following instances are supported:</p> <p><code>vsip_mxor_i</code> <code>vsip_mxor_si</code> <code>vsip_mxor_bl</code></p>

6.8 Element Generation and Copy

Prototype	Description
<pre>void vsip_Dvcopy_P_P(const vsip_Dvview_P * A, const vsip_Dvview_P * R);</pre>	<p>Copy the source vector to the destination vector performing any necessary type conversion of the standard ANSI C scalar types.</p> <p>The following instances are supported:</p> <pre>vsip_vcopy_f_f vsip_vcopy_f_i vsip_vcopy_f_si vsip_vcopy_f_bl vsip_vcopy_i_f vsip_vcopy_i_i vsip_vcopy_i_si vsip_vcopy_i_vi vsip_vcopy_si_f vsip_vcopy_si_i vsip_vcopy_si_si vsip_vcopy_bl_f vsip_vcopy_bl_bl vsip_vcopy_vi_i vsip_vcopy_vi_vi vsip_vcopy_mi_mi vsip_cvcopy_f_f</pre>
<pre>void vsip_Dmcopy_P_P(const vsip_Dmview_P * A, const vsip_Dmview_P * R);</pre>	<p>Copy the source matrix to the destination matrix performing any necessary type conversion of the standard ANSI C scalar types.</p> <p>The following instances are supported:</p> <pre>vsip_mcopy_f_f vsip_mcopy_f_i vsip_mcopy_f_si vsip_mcopy_f_bl vsip_mcopy_i_f vsip_mcopy_i_i vsip_mcopy_i_si vsip_mcopy_si_f vsip_mcopy_si_i vsip_mcopy_si_si vsip_mcopy_bl_f vsip_mcopy_bl_bl vsip_cmcopy_f_f</pre>
<pre>void vsip_Dvfill_P(const vsip_Dscalar_P a, const vsip_Dvview_P * R);</pre>	<p>Fill a vector with a constant value.</p> <p>The following instances are supported:</p> <pre>vsip_vfill_f vsip_vfill_i vsip_vfill_si vsip_cvfill_f</pre>
<pre>void vsip_Dmfill_P(const vsip_Dscalar_P a, const vsip_Dmview_P * R);</pre>	<p>Fill a matrix with a constant value.</p> <p>The following instances are supported:</p> <pre>vsip_mfill_f vsip_mfill_i vsip_mfill_si vsip_cmfill_f</pre>

Prototype	Description
<pre>void vsip_vramp_P(const vsip_scalar_P alpha, const vsip_scalar_P beta, const vsip_vview_P * R);</pre>	<p>Computes a vector ramp by starting at an initial value and incrementing each successive element by the ramp step size.</p> <p>The following instances are supported:</p> <pre>vsip_vramp_f vsip_vramp_i vsip_vramp_si</pre>

6.9 Manipulation Operations

Prototype	Description
<pre>void vsip_vcmlpx_f(const vsip_vview_f * A, const vsip_vview_f * B, const vsip_cvview_f * R);</pre>	<p>Form a complex vector from two real vectors.</p>
<pre>void vsip_mcmlpx_f(const vsip_mview_f * A, const vsip_mview_f * B, const vsip_cmview_f * R);</pre>	<p>Form a complex matrix from two real matrices.</p>
<pre>void vsip_Dvgather_P(const vsip_Dvview_P * X, const vsip_vview_vi * I, const vsip_Dvview_P * Y);</pre>	<p>The gather operation selects elements of a source vector using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.</p> <p>The following instances are supported:</p> <pre>vsip_vgather_f vsip_vgather_i vsip_vgather_si vsip_cvgather_f</pre>
<pre>void vsip_Dmgather_P(const vsip_Dmview_P * X, const vsip_vview_mi * I, const vsip_Dvview_P * Y);</pre>	<p>The gather operation selects elements of a source vector/matrix using indices supplied by an index vector. The selected elements are placed sequentially in an output vector so that the output vector and the index vector are indexed the same.</p> <p>The following instances are supported:</p> <pre>vsip_mgather_f vsip_mgather_i vsip_mgather_si vsip_cmgather_f</pre>
<pre>void vsip_vimag_f(const vsip_cvview_f * A, const vsip_vview_f * R);</pre>	<p>Extract the imaginary part of a complex vector.</p>
<pre>void vsip_mimag_f(const vsip_cmview_f * A, const vsip_mview_f * R);</pre>	<p>Extract the imaginary part of a complex matrix.</p>

Prototype	Description
<pre>void vsip_vpolar_f(const vsip_cvview_f * A, const vsip_vview_f * R, const vsip_vview_f * P);</pre>	Convert a complex vector from rectangular to polar form. The polar data consists of a real vector containing the radius and a corresponding real vector containing the argument (angle) of the complex input data.
<pre>void vsip_mpolar_f(const vsip_cmview_f * A, const vsip_mvview_f * R, const vsip_mvview_f * P);</pre>	Convert a complex matrix from rectangular to polar form. The polar data consists of a real matrix containing the radius and a corresponding real matrix containing the argument (angle) of the complex input data.
<pre>void vsip_vreal_f(const vsip_cvview_f * A, const vsip_vview_f * R);</pre>	Extract the real part of a complex vector.
<pre>void vsip_mreal_f(const vsip_cmview_f * A, const vsip_mvview_f * R);</pre>	Extract the real part of a complex matrix.
<pre>void vsip_vrect_f(const vsip_vview_f * R, const vsip_vview_f * P, const vsip_cvview_f * A);</pre>	Convert a pair of real vectors from complex polar to complex rectangular form.
<pre>void vsip_mrect_f(const vsip_mvview_f * R, const vsip_mvview_f * P, const vsip_cmview_f * A);</pre>	Convert a pair of real matrices from complex polar to complex rectangular form.
<pre>void vsip_Dvscatter_P(const vsip_Dvview_P * X, const vsip_Dvview_P * Y, const vsip_vview_vi * I);</pre>	<p>The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector index is used to select a storage location in the output vector to store the element from the source vector.</p> <p>The following instances are supported:</p> <pre>vsip_vscatter_f vsip_vscatter_i vsip_vscatter_si vsip_cvscatter_f</pre>
<pre>void vsip_Dmscatter_P(const vsip_Dvview_P * X, const vsip_Dmvview_P * Y, const vsip_vview_mi * I);</pre>	<p>The scatter operation sequentially uses elements of a source vector and an index vector. The element of the vector/matrix index is used to select a storage location in the output vector/matrix to store the element from the source vector.</p> <p>The following instances are supported:</p> <pre>vsip_mscatter_f vsip_mscatter_i vsip_mscatter_si vsip_cmscatter_f</pre>

Prototype	Description
<pre>void vsip_Dvswap_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B);</pre>	<p>Swap elements between two vectors. The following instances are supported:</p> <pre>vsip_vswap_f vsip_vswap_i vsip_vswap_si vsip_cvswap_f</pre>
<pre>void vsip_Dmswap_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B);</pre>	<p>Swap elements between two matrices. The following instances are supported:</p> <pre>vsip_mswap_f vsip_mswap_i vsip_mswap_si vsip_cmswap_f</pre>

6.10 Extensions

Prototype	Description
<pre>vsip_scalar_f vsip_vcsummgval_f(const vsip_cvview_f * A);</pre>	<p>Returns the sum of the magnitudes of the elements of a complex vector.</p>

Chapter 7. Signal Processing Functions

7.1 FFT Functions

Prototype	Description
<pre>vsip_fft_f * vsip_ccfftip_create_f(const vsip_index length, const vsip_scalar_f scale, const vsip_fft_dir dir, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 1D FFT object.
<pre>vsip_fft_f * vsip_ccfftop_create_f(const vsip_index length, const vsip_scalar_f scale, const vsip_fft_dir dir, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 1D FFT object.
<pre>vsip_fft_f * vsip_crfftop_create_f(const vsip_index length, const vsip_scalar_f scale, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 1D FFT object.
<pre>vsip_fft_f * vsip_rcfftop_create_f(const vsip_index length, const vsip_scalar_f scale, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 1D FFT object.
<pre>int vsip_fft_destroy_f(vsip_fft_f * plan);</pre>	Destroy an FFT object.
<pre>void vsip_fft_getattr_f(const vsip_fft_f * plan, vsip_fft_attr_f * attr);</pre>	Return the attributes of an FFT object.
<pre>void vsip_ccfftip_f(const vsip_fft_f * plan, const vsip_cvview_f * xy);</pre>	Apply a complex-to-complex Fast Fourier Transform (FFT).
<pre>void vsip_ccfftop_f(const vsip_fft_f * plan, const vsip_cvview_f * x, const vsip_cvview_f * y);</pre>	Apply a complex-to-complex Fast Fourier Transform (FFT).
<pre>void vsip_crfftop_f(const vsip_fft_f * plan, const vsip_cvview_f * x, const vsip_vview_f * y);</pre>	Apply a complex-to-real Fast Fourier Transform (FFT).
<pre>void vsip_rcfftop_f(const vsip_fft_f * plan, const vsip_vview_f * x, const vsip_cvview_f * y);</pre>	Apply a real-to-complex Fast Fourier Transform (FFT).

Prototype	Description
<pre>vsip_fftm_f * vsip_ccfftmip_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_fft_dir dir, const vsip_major major, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 1D multiple FFT object.
<pre>vsip_fftm_f * vsip_ccfftmop_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_fft_dir dir, const vsip_major major, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 1D multiple FFT object.
<pre>vsip_fftm_f * vsip_crfftmop_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_major major, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 1D multiple FFT object.
<pre>vsip_fftm_f * vsip_rcfftmop_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_major major, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 1D multiple FFT object.
<pre>int vsip_fftm_destroy_f(vsip_fftm_f * plan);</pre>	Destroy an FFT object.
<pre>void vsip_fftm_getattr_f(const vsip_fftm_f * plan, vsip_fftm_attr_f * attr);</pre>	Return the attributes of an FFT object.
<pre>void vsip_ccfftmip_f(const vsip_fftm_f * plan, const vsip_cmview_f * XY);</pre>	Apply a multiple complex-to-complex Fast Fourier Transform (FFT).
<pre>void vsip_ccfftmop_f(const vsip_fftm_f * plan, const vsip_cmview_f * X, const vsip_cmview_f * Y);</pre>	Apply a multiple complex-to-complex Fast Fourier Transform (FFT).
<pre>void vsip_crfftmop_f(const vsip_fftm_f * plan, const vsip_cmview_f * X, const vsip_mview_f * Y);</pre>	Apply a multiple complex-to-real Fast Fourier Transform (FFT).
<pre>void vsip_rcfftmop_f(const vsip_fftm_f * plan, const vsip_mview_f * X, const vsip_cmview_f * Y);</pre>	Apply a multiple real-to-complex out of place Fast Fourier Transform (FFT).

Prototype	Description
<pre>vsip_fft2d_f * vsip_ccfft2dip_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_fft_dir dir, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 2D FFT object.
<pre>vsip_fft2d_f * vsip_ccfft2dop_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_fft_dir dir, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 2D FFT object.
<pre>vsip_fft2d_f * vsip_crfft2dop_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 2D FFT object.
<pre>vsip_fft2d_f * vsip_rcfft2dop_create_f(const vsip_index rows, const vsip_index cols, const vsip_scalar_f scale, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 2D FFT object.
<pre>int vsip_fft2d_destroy_f(vsip_fft2d_f * plan);</pre>	Destroy an FFT object.
<pre>void vsip_fft2d_getattr_f(const vsip_fft2d_f * plan, vsip_fft2d_attr_f * attr);</pre>	Return the attributes of an FFT object.
<pre>void vsip_ccfft2dip_f(const vsip_fft2d_f * plan, const vsip_cmview_f * XY);</pre>	Apply a complex-to-complex 2D Fast Fourier Transform (FFT).
<pre>void vsip_ccfft2dop_f(const vsip_fft2d_f * plan, const vsip_cmview_f * X, const vsip_cmview_f * Y);</pre>	Apply a complex-to-complex 2D Fast Fourier Transform (FFT).
<pre>void vsip_crfft2dop_f(const vsip_fft2d_f * plan, const vsip_cmview_f * X, const vsip_mview_f * Y);</pre>	Apply a complex-to-real 2D Fast Fourier Transform (FFT).
<pre>void vsip_rcfft2dop_f(const vsip_fft2d_f * plan, const vsip_mview_f * X, const vsip_cmview_f * Y);</pre>	Apply a real-to-complex 2D Fast Fourier Transform (FFT).

7.2 Convolution/Correlation Functions

Prototype	Description
<pre>vsip_conv1d_f * vsip_conv1d_create_f(const vsip_vview_f * kernel, const vsip_symmetry symm, const vsip_length N, const vsip_length D, const vsip_support_region support, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a decimated 1D convolution filter object.
<pre>int vsip_conv1d_destroy_f(vsip_conv1d_f * plan);</pre>	Destroy a 1D convolution object.
<pre>void vsip_conv1d_getattr_f(const vsip_conv1d_f * plan, vsip_conv1d_attr_f * attr);</pre>	Returns the attributes for a 1D convolution object.
<pre>void vsip_convolve1d_f(const vsip_conv1d_f * plan, const vsip_vview_f * x, const vsip_vview_f * y);</pre>	Compute a decimated real one-dimensional (1D) convolution of two vectors.
<pre>vsip_conv2d_f * vsip_conv2d_create_f(const vsip_mview_f * H, const vsip_symmetry symm, const vsip_length P, const vsip_length Q, const vsip_length D, const vsip_support_region support, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a decimated 2D convolution filter object.
<pre>int vsip_conv2d_destroy_f(vsip_conv2d_f * plan);</pre>	Destroy a 2D convolution object.
<pre>void vsip_conv2d_getattr_f(const vsip_conv2d_f * plan, vsip_conv2d_attr_f * attr);</pre>	Returns the attributes for a 2D convolution object.
<pre>void vsip_convolve2d_f(const vsip_conv2d_f * plan, const vsip_mview_f * x, const vsip_mview_f * y);</pre>	Compute a decimated real two-dimensional (2D) convolution of two matrices.
<pre>vsip_Dcorr1d_P * vsip_Dcorr1d_create_P(const vsip_length M, const vsip_length N, const vsip_support_region support, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a 1D correlation object. The following instances are supported: <code>vsip_corr1d_create_f</code> <code>vsip_ccorr1d_create_f</code>
<pre>int vsip_Dcorr1d_destroy_P(vsip_Dcorr1d_P * plan);</pre>	Destroy a 1D correlation object. The following instances are supported: <code>vsip_corr1d_destroy_f</code> <code>vsip_ccorr1d_destroy_f</code>

Prototype	Description
<pre>void vsip_Dcorr1d_getattr_P(const vsip_Dcorr1d_P * plan, vsip_Dcorr1d_attr_P * attr);</pre>	<p>Return the attributes for a 1D correlation object. The following instances are supported:</p> <p><code>vsip_corr1d_getattr_f</code> <code>vsip_ccorr1d_getattr_f</code></p>
<pre>void vsip_Dcorrelate1d_P(const vsip_Dcorr1d_P * plan, const vsip_bias bias, const vsip_Dvview_P * ref, const vsip_Dvview_P * x, const vsip_Dvview_P * y);</pre>	<p>Compute a real one-dimensional (1D) correlation of two vectors. The following instances are supported:</p> <p><code>vsip_correlate1d_f</code> <code>vsip_ccorrelate1d_f</code></p>
<pre>vsip_Dcorr2d_P * vsip_Dcorr2d_create_P(const vsip_length M, const vsip_length N, vsip_length P, vsip_length Q, const vsip_support_region support, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	<p>Create a 2D correlation object. The following instances are supported:</p> <p><code>vsip_corr2d_create_f</code> <code>vsip_ccorr2d_create_f</code></p>
<pre>int vsip_Dcorr2d_destroy_P(vsip_Dcorr2d_P * plan);</pre>	<p>Destroy a 2D correlation object. The following instances are supported:</p> <p><code>vsip_corr2d_destroy_f</code> <code>vsip_ccorr2d_destroy_f</code></p>
<pre>void vsip_Dcorr2d_getattr_P(const vsip_Dcorr2d_P * plan, vsip_Dcorr2d_attr_P * attr);</pre>	<p>Return the attributes for a 2D correlation object. The following instances are supported:</p> <p><code>vsip_corr2d_getattr_f</code> <code>vsip_ccorr2d_getattr_f</code></p>
<pre>void vsip_Dcorrelate2d_P(const vsip_Dcorr2d_P * plan, const vsip_bias bias, const vsip_Dmview_P * ref, const vsip_Dmview_P * x, const vsip_Dmview_P * y);</pre>	<p>Compute a two-dimensional (2D) correlation of two matrices. The following instances are supported:</p> <p><code>vsip_correlate2d_f</code> <code>vsip_ccorrelate2d_f</code></p>

7.3 Window Functions

Prototype	Description
<pre>vsip_vview_f * vsip_vcreate_blackman_f(const vsip_length N, const vsip_memory_hint hint);</pre>	<p>Create a vector with Blackman window weights.</p>
<pre>vsip_vview_f * vsip_vcreate_cheby_f(const vsip_length N, const vsip_scalar_f ripple, const vsip_memory_hint hint);</pre>	<p>Create a vector with Dolph-Chebyshev window weights.</p>
<pre>vsip_vview_f * vsip_vcreate_hanning_f(const vsip_length N, const vsip_memory_hint hint);</pre>	<p>Create a vector with Hanning window weights.</p>

Prototype	Description
<pre>vsip_vview_f * vsip_vcreate_kaiser_f(const vsip_length N, const vsip_scalar_f beta, const vsip_memory_hint hint);</pre>	Create a vector with Kaiser window weights.

7.4 Filter Functions

Prototype	Description
<pre>vsip_Dfir_P * vsip_Dfir_create_P(const vsip_Dvview_P * kernel, const vsip_symmetry symm, const vsip_length N, const vsip_length D, const vsip_obj_state state, const vsip_length ntimes, const vsip_alg_hint hint);</pre>	Create a decimated FIR filter object. The following instances are supported: <code>vsip_fir_create_f</code> <code>vsip_cfir_create_f</code>
<pre>int vsip_Dfir_destroy_P(vsip_Dfir_P * plan);</pre>	Destroy a FIR filter object. The following instances are supported: <code>vsip_fir_destroy_f</code> <code>vsip_cfir_destroy_f</code>
<pre>int vsip_Dfirflt_P(vsip_Dfir_P * plan, const vsip_Dvview_P * x, const vsip_Dvview_P * y);</pre>	FIR filter an input sequence and decimate the output. The following instances are supported: <code>vsip_firflt_f</code> <code>vsip_cfirflt_f</code>
<pre>void vsip_Dfir_getattr_P(const vsip_Dfir_P * plan, vsip_Dfir_attr_P * attr);</pre>	Return the attributes of a FIR filter object. The following instances are supported: <code>vsip_fir_getattr_f</code> <code>vsip_cfir_getattr_f</code>
<pre>void vsip_Dfir_reset_P(vsip_Dfir_P * fir);</pre>	Reset the state of a decimated FIR filter object. The following instances are supported: <code>vsip_fir_reset_f</code> <code>vsip_cfir_reset_f</code>

7.5 Miscellaneous Signal Processing Functions

Prototype	Description
<pre>void vsip_vhisto_f(const vsip_vview_f * A, const vsip_scalar_f min, const vsip_scalar_f max, const vsip_hist_opt opt, const vsip_vview_f * R);</pre>	Compute the histogram of a vector.

Chapter 8. Linear Algebra

8.1 Matrix and Vector Operations

Prototype	Description
<pre>void vsip_cmherm_f(const vsip_cmview_f * A, const vsip_cmview_f * R);</pre>	Complex Hermitian (conjugate transpose) of a matrix.
<pre>vsip_cscalar_f vsip_cvjdot_f(const vsip_cvview_f * A, const vsip_cvview_f * B);</pre>	Compute the conjugate inner (dot) product of two complex vectors.
<pre>void vsip_gemp_f(const vsip_scalar_f alpha, const vsip_mview_f * A, const vsip_mat_op Aop, const vsip_mview_f * B, const vsip_mat_op Bop, const vsip_scalar_f beta, const vsip_mview_f * R);</pre>	Calculate the general product of two matrices and accumulate.
<pre>void vsip_cgemp_f(const vsip_cscalar_f alpha, const vsip_cmview_f * A, const vsip_mat_op Aop, const vsip_cmview_f * B, const vsip_mat_op Bop, const vsip_cscalar_f beta, const vsip_cmview_f * R);</pre>	Calculate the general product of two matrices and accumulate.
<pre>void vsip_gems_f(const vsip_scalar_f alpha, const vsip_mview_f * A, const vsip_mat_op Aop, const vsip_scalar_f beta, const vsip_mview_f * C);</pre>	Calculate a general matrix sum.
<pre>void vsip_cgems_f(const vsip_cscalar_f alpha, const vsip_cmview_f * A, const vsip_mat_op Aop, const vsip_cscalar_f beta, const vsip_cmview_f * C);</pre>	Calculate a general matrix sum.
<pre>void vsip_Dmprod_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);</pre>	Calculate the product of two matrices. The following instances are supported: <pre>vsip_mprod_f vsip_mprod_i vsip_mprod_si vsip_cmprod_f vsip_cmprod_i vsip_cmprod_si</pre>

Prototype	Description
<pre>void vsip_cmprodh_P(const vsip_cmview_P * A, const vsip_cmview_P * B, const vsip_cmview_P * R);</pre>	<p>Calculate the product a complex matrix and the Hermitian of a complex matrix.</p> <p>The following instances are supported:</p> <pre>vsip_cmprodh_f vsip_cmprodh_i vsip_cmprodh_si</pre>
<pre>void vsip_cmprodj_P(const vsip_cmview_P * A, const vsip_cmview_P * B, const vsip_cmview_P * R);</pre>	<p>Calculate the product a complex matrix and the conjugate of a complex matrix.</p> <p>The following instances are supported:</p> <pre>vsip_cmprodj_f vsip_cmprodj_i vsip_cmprodj_si</pre>
<pre>void vsip_Dmprodt_P(const vsip_Dmview_P * A, const vsip_Dmview_P * B, const vsip_Dmview_P * R);</pre>	<p>Calculate the product of a matrix and the transpose of a matrix.</p> <p>The following instances are supported:</p> <pre>vsip_mprodt_f vsip_mprodt_i vsip_mprodt_si vsip_cmprodt_f vsip_cmprodt_i vsip_cmprodt_si</pre>
<pre>void vsip_Dmvprod_P(const vsip_Dmview_P * A, const vsip_Dvview_P * X, const vsip_Dvview_P * Y);</pre>	<p>Calculate a matrix–vector product.</p> <p>The following instances are supported:</p> <pre>vsip_mvprod_f vsip_mvprod_i vsip_mvprod_si vsip_cmvprod_f vsip_cmvprod_i vsip_cmvprod_si</pre>
<pre>void vsip_Dmtrans_P(const vsip_Dmview_P * A, const vsip_Dmview_P * R);</pre>	<p>Transpose a matrix.</p> <p>The following instances are supported:</p> <pre>vsip_mtrans_bl vsip_mtrans_f vsip_mtrans_i vsip_mtrans_si vsip_cmtrans_f vsip_cmtrans_i vsip_cmtrans_si</pre>
<pre>vsip_Dscalar_P vsip_Dvdot_P(const vsip_Dvview_P * A, const vsip_Dvview_P * B);</pre>	<p>Compute the inner (dot) product of two vectors.</p> <p>The following instances are supported:</p> <pre>vsip_vdot_f vsip_cvdot_f</pre>

Prototype	Description
<pre>void vsip_Dvmprod_P(const vsip_Dvview_P * X, const vsip_Dmview_P * A, const vsip_Dvview_P * Y);</pre>	<p>Calculate a vector–matrix product. The following instances are supported:</p> <pre>vsip_vmprod_f vsip_vmprod_i vsip_vmprod_si vsip_cvmprod_f vsip_cvmprod_i vsip_cvmprod_si</pre>
<pre>void vsip_vouter_f(const vsip_scalar_f alpha, const vsip_vview_f * X, const vsip_vview_f * Y, const vsip_vview_f * R);</pre>	Calculate the outer product of two vectors.
<pre>void vsip_cvouter_f(const vsip_cscalar_f alpha, const vsip_cvview_f * X, const vsip_cvview_f * Y, const vsip_cvview_f * R);</pre>	Calculate the outer product of two vectors.

8.2 Special Linear System Solvers

Prototype	Description
<pre>int vsip_covsol_f(const vsip_mview_f * A, const vsip_mview_f * XB);</pre>	Solve a covariance linear system problem.
<pre>int vsip_ccovsol_f(const vsip_cmview_f * A, const vsip_cmview_f * XB);</pre>	Solve a covariance linear system problem.
<pre>int vsip_llsqsol_f(const vsip_mview_f * A, const vsip_mview_f * XB);</pre>	Solve a linear least squares problem.
<pre>int vsip_cllsqsol_f(const vsip_cmview_f * A, const vsip_cmview_f * XB);</pre>	Solve a linear least squares problem.
<pre>int vsip_toepsol_f(const vsip_vview_f * T, const vsip_vview_f * B, const vsip_vview_f * W, const vsip_vview_f * X);</pre>	Solve a real symmetric positive definite Toeplitz linear system.
<pre>int vsip_ctoepsol_f(const vsip_cvview_f * T, const vsip_cvview_f * B, const vsip_cvview_f * W, const vsip_cvview_f * X);</pre>	Solve a complex Hermitian positive definite Toeplitz linear system.

8.3 General Square Linear System Solver

Prototype	Description
<pre>int vsip_Dlud_P(vsip_clu_P * lud, const vsip_Dmview_P * A);</pre>	<p>Compute an LU decomposition of a square matrix using partial pivoting. The following instances are supported:</p> <p><code>vsip_lud_f</code> <code>vsip_clud_f</code></p>
<pre>vsip_Dlu_P * vsip_Dlud_create_P(const vsip_length N);</pre>	<p>Create an LU decomposition object. The following instances are supported:</p> <p><code>vsip_lud_create_f</code> <code>vsip_clud_create_f</code></p>
<pre>int vsip_Dlud_destroy_P(vsip_Dlu_P * lud);</pre>	<p>Destroy an LU decomposition object. The following instances are supported:</p> <p><code>vsip_lud_destroy_f</code> <code>vsip_clud_destroy_f</code></p>
<pre>void vsip_Dlud_getattr_P(const vsip_Dlu_P * lud, vsip_Dlu_attr_P * attr);</pre>	<p>Returns the attributes of an LU decomposition object. The following instances are supported:</p> <p><code>vsip_lud_getattr_f</code> <code>vsip_clud_getattr_f</code></p>
<pre>int vsip_lusol_f(const vsip_clu_f * clud, const vsip_mat_op opA, const vsip_mview_f * XB);</pre>	<p>Solve a square linear system.</p>
<pre>int vsip_clusol_f(const vsip_clu_f * clud, const vsip_mat_op opA, const vsip_cmview_f * XB);</pre>	<p>Solve a square linear system.</p>

8.4 Symmetric Positive Definite Linear System Solver

Prototype	Description
<pre>int vsip_chold_f(vsip_cchol_f * chold, const vsip_mview_f * A);</pre>	<p>Compute a Cholesky decomposition of a symmetric positive definite matrix.</p>
<pre>int vsip_cchold_f(vsip_cchol_f * chold, const vsip_cmview_f * A);</pre>	<p>Compute a Cholesky decomposition of a Hermitian positive definite matrix.</p>
<pre>vsip_chol_f * vsip_chold_create_f(const vsip_mat_uplo uplo, const vsip_length n);</pre>	<p>Creates a Cholesky decomposition object.</p>
<pre>vsip_cchol_f * vsip_cchold_create_f(const vsip_mat_uplo uplo, const vsip_length n);</pre>	<p>Creates a Cholesky decomposition object.</p>

Prototype	Description
<pre>int vsip_Dchold_destroy_P(vsip_Dchol_P * chold);</pre>	<p>Destroy a Cholesky decomposition object. The following instances are supported:</p> <p><code>vsip_chold_destroy_f</code> <code>vsip_cchold_destroy_f</code></p>
<pre>void vsip_Dchold_getattr_P(const vsip_Dchol_P * chold, vsip_Dchol_attr_P * attr);</pre>	<p>Returns the attributes of a Cholesky decomposition object. The following instances are supported:</p> <p><code>vsip_chold_getattr_f</code> <code>vsip_cchold_getattr_f</code></p>
<pre>int vsip_cholsol_f(const vsip_cchol_f * chold, const vsip_mview_f * XB);</pre>	<p>Solve a symmetric positive definite linear system.</p>
<pre>int vsip_ccholsol_f(const vsip_cchol_f * chold, const vsip_cmview_f * XB);</pre>	<p>Solve a Hermitian positive definite linear system.</p>

8.5 Overdetermined Linear System Solver

Prototype	Description
<pre>int vsip_qrd_f(vsip_cqr_f * qrd, const vsip_mview_f * A);</pre>	<p>Compute a QR decomposition of a matrix .</p>
<pre>int vsip_cqrd_f(vsip_cqr_f * qrd, const vsip_cmview_f * A);</pre>	<p>Compute a QR decomposition of a matrix .</p>
<pre>vsip_qr_f * vsip_qrd_create_f(const vsip_length m, const vsip_length n, const vsip_qrd_qopt qopt);</pre>	<p>Create a QR decomposition object.</p>
<pre>vsip_cqr_f * vsip_cqrd_create_f(const vsip_length m, const vsip_length n, const vsip_qrd_qopt qopt);</pre>	<p>Create a QR decomposition object.</p>
<pre>int vsip_Dqrd_destroy_P(vsip_Dqr_P * qrd);</pre>	<p>Destroy a QR decomposition object. The following instances are supported:</p> <p><code>vsip_qrd_destroy_f</code> <code>vsip_cqrd_destroy_f</code></p>
<pre>void vsip_Dqrd_getattr_P(const vsip_Dqr_P * qrd, vsip_Dqr_attr_P * attr);</pre>	<p>Returns the attributes of a QR decomposition object. The following instances are supported:</p> <p><code>vsip_qrd_getattr_f</code> <code>vsip_cqrd_getattr_f</code></p>

Prototype	Description
<pre>int vsip_qrdprodq_f(const vsip_qr_f * qrd, const vsip_mat_op opQ, const vsip_mat_side apQ, const vsip_mview_f * C);</pre>	Multiply a matrix by the matrix Q from a QR decomposition.
<pre>int vsip_cqrdprodq_f(const vsip_cqr_f * qrd, const vsip_mat_op opQ, const vsip_mat_side apQ, const vsip_cmview_f * C);</pre>	Multiply a matrix by the matrix Q from a QR decomposition.
<pre>int vsip_qrdsolr_f(const vsip_qr_f * qrd, const vsip_mat_op OpR, const vsip_scalar_f alpha, const vsip_mview_f * XB);</pre>	Solve linear system based on the matrix R , from QR decomposition of the matrix A .
<pre>int vsip_cqrdsolr_f(const vsip_cqr_f * qrd, const vsip_mat_op OpR, const vsip_cscalar_f alpha, const vsip_cmview_f * XB);</pre>	Solve linear system based on the matrix R , from QR decomposition of the matrix A .
<pre>int vsip_qrsol_f(const vsip_qr_f * qrd, const vsip_qrd_prob prob, const vsip_mview_f * XB);</pre>	Solve either a linear covariance or linear least squares problem.
<pre>int vsip_cqrsol_f(const vsip_cqr_f * qrd, const vsip_qrd_prob prob, const vsip_cmview_f * XB);</pre>	Solve either a linear covariance or linear least squares problem.

8.6 Singular Value Decomposition

Prototype	Description
<pre>int vsip_Dsvd_P(vsip_Dsv_P * svd, const vsip_Dmview_P * A, const vsip_vvview_P * s);</pre>	Compute the singular value decomposition of a matrix. The following instances are supported: <code>vsip_svd_f</code> <code>vsip_csvd_f</code>
<pre>vsip_Dsv_P * vsip_Dsvd_create_P(const vsip_length M, const vsip_length N, const vsip_svd_uv Usave, const vsip_svd_uv Vsave);</pre>	Create the singular value decomposition (SVD) object. The following instances are supported: <code>vsip_svd_create_f</code> <code>vsip_csvd_create_f</code>
<pre>int vsip_Dsvd_destroy_P(vsip_Dsv_P * svd);</pre>	Destroy an SVD object. The following instances are supported: <code>vsip_svd_destroy_f</code> <code>vsip_csvd_destroy_f</code>

Prototype	Description
<pre>void vsip_Dsvd_getattr_P(const vsip_Dsv_P * svd, vsip_Dsv_attr_P * attr);</pre>	<p>Returns the attributes of an SVD object. The following instances are supported:</p> <p><code>vsip_svd_getattr_f</code> <code>vsip_csvd_getattr_f</code></p>
<pre>int vsip_Dsvdprodu_P(const vsip_Dsv_P * svd, const vsip_mat_op opU, const vsip_mat_side apU, const vsip_Dmview_P * C);</pre>	<p>Multiply a matrix by the matrix U from a singular value decomposition. The following instances are supported:</p> <p><code>vsip_svdprodu_f</code> <code>vsip_csvdprodu_f</code></p>
<pre>int vsip_Dsvdprodv_P(const vsip_Dsv_P * svd, const vsip_mat_op opU, const vsip_mat_side apU, const vsip_Dmview_P * C);</pre>	<p>Multiply a matrix by the matrix U from a singular value decomposition. The following instances are supported:</p> <p><code>vsip_svdprodv_f</code> <code>vsip_csvdprodv_f</code></p>
<pre>int vsip_Dsvdmatu_P(const vsip_Dsv_P * svd, const vsip_scalar_vi low, const vsip_scalar_vi high, const vsip_Dmview_P * C);</pre>	<p>Returns consecutive columns in the matrix U from a singular value decomposition. The following instances are supported:</p> <p><code>vsip_svdmatu_f</code> <code>vsip_csvdmatu_f</code></p>
<pre>int vsip_Dsvdmatv_P(const vsip_Dsv_P * svd, const vsip_scalar_vi low, const vsip_scalar_vi high, const vsip_Dmview_P * C);</pre>	<p>Returns consecutive columns in the matrix V from a singular value decomposition. The following instances are supported:</p> <p><code>vsip_svdmatv_f</code> <code>vsip_csvdmatv_f</code></p>

8.7 Extensions

Prototype	Description
<pre>void vsip_Dminvlu_P(const vsip_Dmview_P * A, const vsip_vview_i * V, const vsip_Dmview_P * R);</pre>	<p>Invert a square matrix using LU decomposition. The following instances are supported:</p> <p><code>vsip_minvlu_f</code> <code>vsip_cminvlu_f</code></p>

Chapter 9. Glossary

Admitted	<i>Block</i> state where the <i>data array</i> (memory) and associated <i>views</i> are available for VSIPL computations, and not available for user I/O or access.
Attribute	Characteristic or state of an object, such as <i>admitted / released, stride, or length</i> .
Binary Function	A function with two input arguments.
Block	A data storage abstraction representing contiguous data elements consisting of a <i>data array</i> and a VSIPL <i>block object</i> .
Block Object	Descriptor for a <i>data array</i> and its <i>attributes</i> , including a reference to the data array, the state of the block, data type and size.
Block Offset	The number of <i>elements</i> from the start of a <i>block</i> . A view with a block offset of zero starts at the beginning of the block.
Boolean	Used to represent the values of true and false, where false is always zero, and true is non-zero.
Bound	A <i>view</i> or <i>block</i> is bound to a <i>data array</i> if it references the data array.
Cloned View	An exact duplicate of a <i>view object</i> .
Column	Rightmost dimension in a <i>matrix</i> .
Column Stride	The number of <i>block elements</i> between successive elements within a <i>column</i> .
Complex Block	<i>Block</i> containing only complex <i>elements</i> . There are two formats for released complex blocks – <i>split</i> and <i>interleaved</i> . The complex data format for admitted complex blocks is not known to the user.
Conformant Views	<i>Views</i> that are the correct shape/size for a given computation.
const Object	An object that is not modified by the function, although data referenced by the const object may be modified.
Create	To allocate memory for an object and initialise it (if appropriate).
Data Array	Memory where data is stored.
Derived Block	A <i>real block</i> derived from a <i>complex block</i> . Note that the only way to create a derived block is to create a <i>derived view</i> of the real or complex component of a <i>split</i> complex view. In all other cases, retrieving the block from a view returns a reference to the original block.

Derived View	A derived view is a view created using a VSIPL function whose arguments include another view (a parent view). The derived view's data is some subset of the parent view's data. The data subset depends on the function call, and is physically co-located in memory with the parent view's data.
Destroy	To release the memory allocated to an object.
Development Library	An implementation of VSIPL that maximises error reporting at the possible expense of performance.
Domain	The set of all valid input values to a function.
Element	The atomic portion of data associated with a <i>block</i> or a <i>view</i> . For example, an element of a <i>complex block</i> of precision double is a complex number of precision double; for a view of type float an element is a single float number.
Hermitian Transpose	Conjugate transpose.
Hint	Information provided by the user to some VSIPL functions to aid optimization. Hints are optional and may be ignored by the implementation. Wrong hints may result in incorrect behavior.
In-Place	A type of algorithm implementation in which the memory used to hold the input to an algorithm is overwritten (completely or partially) with the output data. Often referred to in the context of an FFT algorithm.
Interleaved Complex	Storage format for <i>user data arrays</i> where the real and complex <i>element</i> components alternate in physical memory.
Kernel	The filter vector used in a FIR filter, or the vector or matrix used as the weights in a convolution.
Length	Number of <i>elements</i> in a view along a <i>view dimension</i> .
Matrix	A two-dimensional view.
Opaque	An opaque object may not be manipulated by simple assignment statements. Its attributes must be set/retrieved through access functions. All VSIPL objects are opaque.
Out-of-place	If none of the output views in a function call overlap the input views, the function is considered out-of-place.
Overlapped	Indicates that two or more <i>views</i> or <i>blocks</i> share one or more memory locations.
Production Library	A VSIPL implementation that maximises performance at the possible expense of not detecting user errors.
Range	Valid output values from a function.
Real Block	A <i>block</i> containing only real <i>elements</i> .
Region of Support	For neighborhood operations (i.e. FIR filtering, convolution), the non-zero values in the kernel, or the output. For example, a 3×3 FIR filter has a 'kernel region of support' of 3×3 .

Released	<i>Block</i> state where the associated <i>data array</i> is available for user I/O and application access, but not available for VSIPL computations.
Row	Left-most dimension of a <i>matrix</i> .
Row Stride	The number of <i>block</i> elements between successive elements within a <i>row</i> .
Split Complex	Storage format for released <i>complex blocks</i> where the real <i>element</i> components are stored in one physically contiguous <i>data array</i> , and the imaginary components are stored in a separate physically contiguous data array.
Stride	Distance between successive <i>elements</i> of the block data array in a <i>view</i> along a view dimension. Strides can be positive, negative, or zero.
Subview	A <i>derived view</i> that describes a subset of the data from the original view, and is the same type as the original view.
Tensor	An n -dimensional matrix. VSIPL only supports three-dimensional tensors (3-tensor). The three dimensions are referred to as X, Y and Z.
Ternary Function	A function with three input arguments.
Unary Function	A function with a single input argument.
User Block	A block which is associated with user data arrays. User blocks are created in the released state and may be admitted and released.
User Data Array	Memory that has been allocated by the application for the storage of data using some functionality not part of the VSIPL standard.
Vector	A one-dimensional <i>view</i> .
View	A portion of a <i>block</i> , and a <i>view object</i> describing it. The view object has structural information allowing the data to be interpreted as a one-, two- or three-dimensional array for arithmetic processing.
View Dimension	A <i>view</i> represents a one-, two-, or three-dimensional data organisation termed respectively a <i>vector</i> , <i>matrix</i> or <i>tensor</i> . A <i>view dimension</i> represents one of the standard directions of these data representations.
View Object	A description of a portion of a <i>block</i> , including structural information that allows the data to be interpreted as a one-, two- or three-dimensional array for arithmetic processing. Attributes of the <i>view object</i> include <i>offset</i> , <i>stride(s)</i> and <i>length(s)</i> .
VSIPL Block	<i>Block</i> referencing or <i>bound</i> to VSIPL data. A VSIPL block is created in the <i>admitted</i> state and may not be <i>released</i> .
VSIPL Data Array	Memory that has been allocated for the storage of data using some functionality that is part of the VSIPL standard.