

ReproBLAS

Generated by Doxygen 1.8.10

Thu Jan 21 2016 13:45:21



# Contents

<b>1</b>	<b>File Index</b>	<b>1</b>
1.1	File List	1
<b>2</b>	<b>File Documentation</b>	<b>3</b>
2.1	include/idxd.h File Reference	3
2.1.1	Detailed Description	10
2.1.2	Macro Definition Documentation	11
2.1.2.1	DIWIDTH	11
2.1.2.2	idxd_DICAPACITY	11
2.1.2.3	idxd_DIENDURANCE	11
2.1.2.4	idxd_DIMAXFOLD	12
2.1.2.5	idxd_DIMAXINDEX	12
2.1.2.6	idxd_DMCOMPRESSION	12
2.1.2.7	idxd_DMEXPANSION	12
2.1.2.8	idxd_SICAPACITY	13
2.1.2.9	idxd_SIENDURANCE	13
2.1.2.10	idxd_SIMAXFOLD	13
2.1.2.11	idxd_SIMAXINDEX	13
2.1.2.12	idxd_SMCOMPRESSION	14
2.1.2.13	idxd_SMEXPANSION	14
2.1.2.14	SIWIDTH	14
2.1.3	Typedef Documentation	14
2.1.3.1	double_complex_indexed	14
2.1.3.2	double_indexed	15
2.1.3.3	float_complex_indexed	15
2.1.3.4	float_indexed	15
2.1.4	Function Documentation	15
2.1.4.1	idxd_cciconv_sub(const int fold, const float_complex_indexed *X, void *conv)	15
2.1.4.2	idxd_ccmconv_sub(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, void *conv)	16
2.1.4.3	idxd_cialloc(const int fold)	17
2.1.4.4	idxd_cicadd(const int fold, const void *X, float_complex_indexed *Y)	17

2.1.4.5	<code>idxd_cicconv(const int fold, const void *X, float_complex_indexed *Y)</code>	18
2.1.4.6	<code>idxd_cicdeposit(const int fold, const void *X, float_complex_indexed *Y)</code>	19
2.1.4.7	<code>idxd_ciciadd(const int fold, const float_complex_indexed *X, float_complex_indexed *Y)</code>	19
2.1.4.8	<code>idxd_ciciaddv(const int fold, const int N, const float_complex_indexed *X, const int incX, float_complex_indexed *Y, const int incY)</code>	20
2.1.4.9	<code>idxd_ciciset(const int fold, const float_complex_indexed *X, float_complex_indexed *Y)</code>	20
2.1.4.10	<code>idxd_cicupdate(const int fold, const void *X, float_complex_indexed *Y)</code>	21
2.1.4.11	<code>idxd_cinegate(const int fold, float_complex_indexed *X)</code>	21
2.1.4.12	<code>idxd_cinum(const int fold)</code>	21
2.1.4.13	<code>idxd_ciprint(const int fold, const float_complex_indexed *X)</code>	22
2.1.4.14	<code>idxd_cirenorm(const int fold, float_complex_indexed *X)</code>	23
2.1.4.15	<code>idxd_cisetzero(const int fold, float_complex_indexed *X)</code>	23
2.1.4.16	<code>idxd_cisiset(const int fold, const float_indexed *X, float_complex_indexed *Y)</code>	23
2.1.4.17	<code>idxd_cisize(const int fold)</code>	24
2.1.4.18	<code>idxd_cisupdate(const int fold, const float X, float_complex_indexed *Y)</code>	24
2.1.4.19	<code>idxd_cmccadd(const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	25
2.1.4.20	<code>idxd_cmccconv(const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	26
2.1.4.21	<code>idxd_cmccdeposit(const int fold, const void *X, float *priY, const int incpriY)</code>	26
2.1.4.22	<code>idxd_cmcmadd(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	27
2.1.4.23	<code>idxd_cmcmset(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	27
2.1.4.24	<code>idxd_cmccupdate(const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	28
2.1.4.25	<code>idxd_cmccdenorm(const int fold, const float *priX)</code>	28
2.1.4.26	<code>idxd_cmccnegate(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	29
2.1.4.27	<code>idxd_cmccprint(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)</code>	29
2.1.4.28	<code>idxd_cmccrenorm(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	29
2.1.4.29	<code>idxd_cmccsetzero(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	30
2.1.4.30	<code>idxd_cmccsmset(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	30
2.1.4.31	<code>idxd_cmccsrescale(const int fold, const float X, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	31
2.1.4.32	<code>idxd_cmccsupupdate(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	31
2.1.4.33	<code>idxd_ddiconv(const int fold, const double_indexed *X)</code>	32

2.1.4.34	<code>idxd_ddmconv(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)</code>	32
2.1.4.35	<code>idxd_dialloc(const int fold)</code>	33
2.1.4.36	<code>idxd_dibound(const int fold, const int N, const double X, const double S)</code>	34
2.1.4.37	<code>idxd_didadd(const int fold, const double X, double_indexed *Y)</code>	34
2.1.4.38	<code>idxd_didconv(const int fold, const double X, double_indexed *Y)</code>	35
2.1.4.39	<code>idxd_diddeposit(const int fold, const double X, double_indexed *Y)</code>	36
2.1.4.40	<code>idxd_didiadd(const int fold, const double_indexed *X, double_indexed *Y)</code>	36
2.1.4.41	<code>idxd_didiaddsq(const int fold, const double scaleX, const double_indexed *X, const double scaleY, double_indexed *Y)</code>	37
2.1.4.42	<code>idxd_didiadv(const int fold, const int N, const double_indexed *X, const int incX, double_indexed *Y, const int incY)</code>	37
2.1.4.43	<code>idxd_didiset(const int fold, const double_indexed *X, double_indexed *Y)</code>	38
2.1.4.44	<code>idxd_didupdate(const int fold, const double X, double_indexed *Y)</code>	38
2.1.4.45	<code>idxd_dindex(const double X)</code>	38
2.1.4.46	<code>idxd_dinegate(const int fold, double_indexed *X)</code>	39
2.1.4.47	<code>idxd_dinum(const int fold)</code>	39
2.1.4.48	<code>idxd_diprint(const int fold, const double_indexed *X)</code>	39
2.1.4.49	<code>idxd_direnorm(const int fold, double_indexed *X)</code>	40
2.1.4.50	<code>idxd_disetzero(const int fold, double_indexed *X)</code>	40
2.1.4.51	<code>idxd_dsize(const int fold)</code>	40
2.1.4.52	<code>idxd_dmbins(const int X)</code>	41
2.1.4.53	<code>idxd_dmdadd(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	41
2.1.4.54	<code>idxd_dmdconv(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	42
2.1.4.55	<code>idxd_dmddeposit(const int fold, const double X, double *priY, const int incpriY)</code>	42
2.1.4.56	<code>idxd_dmdenorm(const int fold, const double *priX)</code>	43
2.1.4.57	<code>idxd_dmdmadd(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	43
2.1.4.58	<code>idxd_dmdmaddsq(const int fold, const double scaleX, const double *priX, const int incpriX, const double *carX, const int inccarX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	44
2.1.4.59	<code>idxd_dmdmset(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	44
2.1.4.60	<code>idxd_dmdrescale(const int fold, const double X, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	45
2.1.4.61	<code>idxd_dmdupdate(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	45
2.1.4.62	<code>idxd_dmindex(const double *priX)</code>	46
2.1.4.63	<code>idxd_dmindex0(const double *priX)</code>	46

2.1.4.64	<code>idxd_dmnegate(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	47
2.1.4.65	<code>idxd_dmprint(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)</code>	47
2.1.4.66	<code>idxd_dmrenorm(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	47
2.1.4.67	<code>idxd_dmsetzero(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	48
2.1.4.68	<code>idxd_dscale(const double X)</code>	48
2.1.4.69	<code>idxd_sialloc(const int fold)</code>	49
2.1.4.70	<code>idxd_sibound(const int fold, const int N, const float X, const float S)</code>	50
2.1.4.71	<code>idxd_sindex(const float X)</code>	50
2.1.4.72	<code>idxd_sinegate(const int fold, float_indexed *X)</code>	51
2.1.4.73	<code>idxd_sinum(const int fold)</code>	51
2.1.4.74	<code>idxd_siprint(const int fold, const float_indexed *X)</code>	51
2.1.4.75	<code>idxd_sirenorm(const int fold, float_indexed *X)</code>	52
2.1.4.76	<code>idxd_sisadd(const int fold, const float X, float_indexed *Y)</code>	52
2.1.4.77	<code>idxd_sisconv(const int fold, const float X, float_indexed *Y)</code>	52
2.1.4.78	<code>idxd_sisdeposit(const int fold, const float X, float_indexed *Y)</code>	53
2.1.4.79	<code>idxd_sisetzzero(const int fold, float_indexed *X)</code>	53
2.1.4.80	<code>idxd_sisiadd(const int fold, const float_indexed *X, float_indexed *Y)</code>	54
2.1.4.81	<code>idxd_sisiaddsq(const int fold, const float scaleX, const float_indexed *X, const float scaleY, float_indexed *Y)</code>	54
2.1.4.82	<code>idxd_sisiadv(const int fold, const int N, const float_indexed *X, const int incX, float_indexed *Y, const int incY)</code>	55
2.1.4.83	<code>idxd_sisiset(const int fold, const float_indexed *X, float_indexed *Y)</code>	55
2.1.4.84	<code>idxd_sisize(const int fold)</code>	55
2.1.4.85	<code>idxd_sisupdate(const int fold, const float X, float_indexed *Y)</code>	56
2.1.4.86	<code>idxd_smbins(const int X)</code>	56
2.1.4.87	<code>idxd_smdenorm(const int fold, const float *priX)</code>	56
2.1.4.88	<code>idxd_smindex(const float *priX)</code>	57
2.1.4.89	<code>idxd_smindex0(const float *priX)</code>	57
2.1.4.90	<code>idxd_smnegate(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	58
2.1.4.91	<code>idxd_smprint(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)</code>	58
2.1.4.92	<code>idxd_smrenorm(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	58
2.1.4.93	<code>idxd_smsadd(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	59
2.1.4.94	<code>idxd_smsconv(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	59
2.1.4.95	<code>idxd_smsdeposit(const int fold, const float X, float *priY, const int incpriY)</code>	60

2.1.4.96	<code>idxd_smsetzero(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	60
2.1.4.97	<code>idxd_smsmadd(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	60
2.1.4.98	<code>idxd_smsmaddsq(const int fold, const float scaleX, const float *priX, const int incpriX, const float *carX, const int inccarX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	61
2.1.4.99	<code>idxd_smsmset(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	61
2.1.4.100	<code>idxd_smsrescale(const int fold, const float X, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	62
2.1.4.101	<code>idxd_smsupdate(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	62
2.1.4.102	<code>idxd_ssacle(const float X)</code>	63
2.1.4.103	<code>idxd_ssiconv(const int fold, const float_indexed *X)</code>	63
2.1.4.104	<code>idxd_ssmconv(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)</code>	64
2.1.4.105	<code>idxd_uftp(const double X)</code>	64
2.1.4.106	<code>idxd_uftp(const float X)</code>	64
2.1.4.107	<code>idxd_zialloc(const int fold)</code>	65
2.1.4.108	<code>idxd_zidiset(const int fold, const double_indexed *X, double_complex_indexed *Y)</code>	65
2.1.4.109	<code>idxd_zidupdate(const int fold, const double X, double_complex_indexed *Y)</code>	66
2.1.4.110	<code>idxd_zinegate(const int fold, double_complex_indexed *X)</code>	67
2.1.4.111	<code>idxd_zinum(const int fold)</code>	67
2.1.4.112	<code>idxd_ziprint(const int fold, const double_complex_indexed *X)</code>	67
2.1.4.113	<code>idxd_zirenorm(const int fold, double_complex_indexed *X)</code>	68
2.1.4.114	<code>idxd_zisetzero(const int fold, double_complex_indexed *X)</code>	68
2.1.4.115	<code>idxd_zisize(const int fold)</code>	68
2.1.4.116	<code>idxd_zizadd(const int fold, const void *X, double_complex_indexed *Y)</code>	69
2.1.4.117	<code>idxd_zizconv(const int fold, const void *X, double_complex_indexed *Y)</code>	69
2.1.4.118	<code>idxd_zizdeposit(const int fold, const void *X, double_complex_indexed *Y)</code>	70
2.1.4.119	<code>idxd_ziziadd(const int fold, const double_complex_indexed *X, double_complex_indexed *Y)</code>	70
2.1.4.120	<code>idxd_ziziaddv(const int fold, const int N, const double_complex_indexed *X, const int incX, double_complex_indexed *Y, const int incY)</code>	70
2.1.4.121	<code>idxd_ziziset(const int fold, const double_complex_indexed *X, double_complex_indexed *Y)</code>	71
2.1.4.122	<code>idxd_zizupdate(const int fold, const void *X, double_complex_indexed *Y)</code>	71
2.1.4.123	<code>idxd_zmdenorm(const int fold, const double *priX)</code>	72
2.1.4.124	<code>idxd_zmdmset(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	72

2.1.4.125	<code>idxd_zmdrescale(const int fold, const double X, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	72
2.1.4.126	<code>idxd_zmdupdate(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	73
2.1.4.127	<code>idxd_zmnegate(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	73
2.1.4.128	<code>idxd_zmprint(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)</code>	74
2.1.4.129	<code>idxd_zmrenorm(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	74
2.1.4.130	<code>idxd_zmsetzero(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	75
2.1.4.131	<code>idxd_zmzadd(const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	75
2.1.4.132	<code>idxd_zmzconv(const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	75
2.1.4.133	<code>idxd_zmzdeposit(const int fold, const void *X, double *priY, const int incpriY)</code>	76
2.1.4.134	<code>idxd_zmzmadd(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	76
2.1.4.135	<code>idxd_zmzmset(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	77
2.1.4.136	<code>idxd_zmzupdate(const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	77
2.1.4.137	<code>idxd_zziconv_sub(const int fold, const double_complex_indexed *X, void *conv)</code>	78
2.1.4.138	<code>idxd_zzmconv_sub(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, void *conv)</code>	78
2.2	<code>include/idxdBLAS.h</code> File Reference	79
2.2.1	Detailed Description	82
2.2.2	Function Documentation	83
2.2.2.1	<code>idxdBLAS_camax_sub(const int N, const void *X, const int incX, void *amax)</code>	83
2.2.2.2	<code>idxdBLAS_camaxm_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *amaxm)</code>	83
2.2.2.3	<code>idxdBLAS_cicdotc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float_indexed *Z)</code>	84
2.2.2.4	<code>idxdBLAS_cicdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float_indexed *Z)</code>	85
2.2.2.5	<code>idxdBLAS_cicgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, float_complex_indexed *C, const int ldc)</code>	85
2.2.2.6	<code>idxdBLAS_cicgemv(const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, float_complex_indexed *Y, const int incY)</code>	86
2.2.2.7	<code>idxdBLAS_cicsum(const int fold, const int N, const void *X, const int incX, float_indexed *Y)</code>	87



2.2.2.8	<code>idxdBLAS_cmcdotc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)</code>	87
2.2.2.9	<code>idxdBLAS_cmcdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)</code>	88
2.2.2.10	<code>idxdBLAS_cmcsu(m(const int fold, const int N, const void *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	88
2.2.2.11	<code>idxdBLAS_damax(const int N, const double *X, const int incX)</code>	89
2.2.2.12	<code>idxdBLAS_damaxm(const int N, const double *X, const int incX, const double *Y, const int incY)</code>	89
2.2.2.13	<code>idxdBLAS_didasum(const int fold, const int N, const double *X, const int incX, double_indexed *Y)</code>	90
2.2.2.14	<code>idxdBLAS_diddot(const int fold, const int N, const double *X, const int incX, const double *Y, const int incY, double_indexed *Z)</code>	90
2.2.2.15	<code>idxdBLAS_didgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, double_indexed *C, const int ldc)</code>	91
2.2.2.16	<code>idxdBLAS_didgemv(const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, double_indexed *Y, const int incY)</code>	92
2.2.2.17	<code>idxdBLAS_didssq(const int fold, const int N, const double *X, const int incX, const double scaleY, double_indexed *Y)</code>	92
2.2.2.18	<code>idxdBLAS_didsum(const int fold, const int N, const double *X, const int incX, double_indexed *Y)</code>	93
2.2.2.19	<code>idxdBLAS_dizasum(const int fold, const int N, const void *X, const int incX, double_indexed *Y)</code>	93
2.2.2.20	<code>idxdBLAS_dizssq(const int fold, const int N, const void *X, const int incX, const double scaleY, double_indexed *Y)</code>	93
2.2.2.21	<code>idxdBLAS_dmdasum(const int fold, const int N, const double *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	94
2.2.2.22	<code>idxdBLAS_dmddot(const int fold, const int N, const double *X, const int incX, const double *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)</code>	94
2.2.2.23	<code>idxdBLAS_dmdssq(const int fold, const int N, const double *X, const int incX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	95
2.2.2.24	<code>idxdBLAS_dmdsum(const int fold, const int N, const double *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	95
2.2.2.25	<code>idxdBLAS_dmzasum(const int fold, const int N, const void *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	96
2.2.2.26	<code>idxdBLAS_dmzssq(const int fold, const int N, const void *X, const int incX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	96
2.2.2.27	<code>idxdBLAS_samax(const int N, const float *X, const int incX)</code>	97
2.2.2.28	<code>idxdBLAS_samaxm(const int N, const float *X, const int incX, const float *Y, const int incY)</code>	97
2.2.2.29	<code>idxdBLAS_sicasum(const int fold, const int N, const void *X, const int incX, float_indexed *Y)</code>	98

2.2.2.30	<code>idxdBLAS_sicssq(const int fold, const int N, const void *X, const int incX, const float scaleY, float_indexed *Y)</code>	98
2.2.2.31	<code>idxdBLAS_sisasum(const int fold, const int N, const float *X, const int incX, float_indexed *Y)</code>	99
2.2.2.32	<code>idxdBLAS_sisdot(const int fold, const int N, const float *X, const int incX, const float *Y, const int incY, float_indexed *Z)</code>	99
2.2.2.33	<code>idxdBLAS_sisgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, float_indexed *C, const int ldc)</code>	100
2.2.2.34	<code>idxdBLAS_sisgemv(const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, float_indexed *Y, const int incY)</code>	101
2.2.2.35	<code>idxdBLAS_sisssq(const int fold, const int N, const float *X, const int incX, const float scaleY, float_indexed *Y)</code>	101
2.2.2.36	<code>idxdBLAS_sissum(const int fold, const int N, const float *X, const int incX, float_indexed *Y)</code>	102
2.2.2.37	<code>idxdBLAS_smcasum(const int fold, const int N, const void *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	102
2.2.2.38	<code>idxdBLAS_smcssq(const int fold, const int N, const void *X, const int incX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	103
2.2.2.39	<code>idxdBLAS_smsasum(const int fold, const int N, const float *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	103
2.2.2.40	<code>idxdBLAS_smsdot(const int fold, const int N, const float *X, const int incX, const float *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)</code>	104
2.2.2.41	<code>idxdBLAS_smsssq(const int fold, const int N, const float *X, const int incX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	104
2.2.2.42	<code>idxdBLAS_smssum(const int fold, const int N, const float *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	105
2.2.2.43	<code>idxdBLAS_zamax_sub(const int N, const void *X, const int incX, void *amax)</code>	105
2.2.2.44	<code>idxdBLAS_zamaxm_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *amaxm)</code>	106
2.2.2.45	<code>idxdBLAS_zizdotc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double_indexed *Z)</code>	106
2.2.2.46	<code>idxdBLAS_zizdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double_indexed *Z)</code>	107
2.2.2.47	<code>idxdBLAS_zizgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, double_complex_indexed *C, const int ldc)</code>	107
2.2.2.48	<code>idxdBLAS_zizgemv(const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, double_complex_indexed *Y, const int incY)</code>	108
2.2.2.49	<code>idxdBLAS_zizsum(const int fold, const int N, const void *X, const int incX, double_indexed *Y)</code>	109
2.2.2.50	<code>idxdBLAS_zmzdotc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)</code>	109

2.2.2.51	<code>idxdBLAS_zmzdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)</code>	110
2.2.2.52	<code>idxdBLAS_zmzsum(const int fold, const int N, const void *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	110
<b>Index</b>		<b>113</b>



# Chapter 1

## File Index

### 1.1 File List

Here is a list of all documented files with brief descriptions:

include/ <a href="#">idxd.h</a>	
<a href="#">idxd.h</a> defines the indexed types and the lower level functions associated with their use . . . .	3
include/ <a href="#">idxdBLAS.h</a>	
<a href="#">idxdBLAS.h</a> defines BLAS Methods that operate on indexed types . . . . .	79
include/ <b>idxdMPI.h</b> . . . . .	??
include/ <b>reproBLAS.h</b> . . . . .	??



## Chapter 2

# File Documentation

### 2.1 include/idxd.h File Reference

[idxd.h](#) defines the indexed types and the lower level functions associated with their use.

```
#include <stddef.h>
#include <stdlib.h>
#include <float.h>
```

#### Macros

- `#define DIWIDTH 40`  
*Indexed double precision bin width.*
- `#define SIWIDTH 13`  
*Indexed single precision bin width.*
- `#define idxd_DIMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DIWIDTH) - 1)`  
*Indexed double precision maximum index.*
- `#define idxd_SIMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SIWIDTH) - 1)`  
*Indexed single precision maximum index.*
- `#define idxd_DIMAXFOLD (idxd_DIMAXINDEX + 1)`  
*The maximum double precision fold supported by the library.*
- `#define idxd_SIMAXFOLD (idxd_SIMAXINDEX + 1)`  
*The maximum single precision fold supported by the library.*
- `#define idxd_DIENDURANCE (1 << (DBL_MANT_DIG - DIWIDTH - 2))`  
*Indexed double precision deposit endurance.*
- `#define idxd_SIENDURANCE (1 << (FLT_MANT_DIG - SIWIDTH - 2))`  
*Indexed single precision deposit endurance.*
- `#define idxd_DICAPACITY (idxd_DIENDURANCE*(1.0/DBL_EPSILON - 1.0))`  
*Indexed double precision capacity.*
- `#define idxd_SICAPACITY (idxd_SIENDURANCE*(1.0/FLT_EPSILON - 1.0))`  
*Indexed single precision capacity.*
- `#define idxd_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DIWIDTH + 1)))`  
*Indexed double precision compression factor.*
- `#define idxd_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SIWIDTH + 1)))`  
*Indexed single precision compression factor.*
- `#define idxd_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DIWIDTH + 1)))`  
*Indexed double precision expansion factor.*
- `#define idxd_SMEXPANSION (1.0*(1 << (FLT_MANT_DIG - SIWIDTH + 1)))`  
*Indexed single precision expansion factor.*

## Typedefs

- typedef double [double\\_indexed](#)  
*The indexed double datatype.*
- typedef double [double\\_complex\\_indexed](#)  
*The indexed complex double datatype.*
- typedef float [float\\_indexed](#)  
*The indexed float datatype.*
- typedef float [float\\_complex\\_indexed](#)  
*The indexed complex float datatype.*

## Functions

- size\_t [idxd\\_disize](#) (const int fold)  
*indexed double precision size*
- size\_t [idxd\\_zisize](#) (const int fold)  
*indexed complex double precision size*
- size\_t [idxd\\_sisize](#) (const int fold)  
*indexed single precision size*
- size\_t [idxd\\_cisize](#) (const int fold)  
*indexed complex single precision size*
- [double\\_indexed](#) \* [idxd\\_dialloc](#) (const int fold)  
*indexed double precision allocation*
- [double\\_complex\\_indexed](#) \* [idxd\\_zialloc](#) (const int fold)  
*indexed complex double precision allocation*
- [float\\_indexed](#) \* [idxd\\_sialloc](#) (const int fold)  
*indexed single precision allocation*
- [float\\_complex\\_indexed](#) \* [idxd\\_cialloc](#) (const int fold)  
*indexed complex single precision allocation*
- int [idxd\\_dinum](#) (const int fold)  
*indexed double precision size*
- int [idxd\\_zinum](#) (const int fold)  
*indexed complex double precision size*
- int [idxd\\_sinum](#) (const int fold)  
*indexed single precision size*
- int [idxd\\_cinum](#) (const int fold)  
*indexed complex single precision size*
- double [idxd\\_dibound](#) (const int fold, const int N, const double X, const double S)  
*Get indexed double precision summation error bound.*
- float [idxd\\_sibound](#) (const int fold, const int N, const float X, const float S)  
*Get indexed single precision summation error bound.*
- const double \* [idxd\\_dmbins](#) (const int X)  
*Get indexed double precision reference bins.*
- const float \* [idxd\\_smbins](#) (const int X)  
*Get indexed single precision reference bins.*
- int [idxd\\_dindex](#) (const double X)  
*Get index of double precision.*
- int [idxd\\_dmindex](#) (const double \*priX)  
*Get index of manually specified indexed double precision.*
- int [idxd\\_dmindex0](#) (const double \*priX)  
*Check if index of manually specified indexed double precision is 0.*



- int `idxd_sindex` (const float X)  
*Get index of single precision.*
- int `idxd_smindex` (const float \*priX)  
*Get index of manually specified indexed single precision.*
- int `idxd_smindex0` (const float \*priX)  
*Check if index of manually specified indexed single precision is 0.*
- int `idxd_dmdenorm` (const int fold, const double \*priX)  
*Check if indexed type has denormal bits.*
- int `idxd_zmdenorm` (const int fold, const double \*priX)  
*Check if indexed type has denormal bits.*
- int `idxd_smdenorm` (const int fold, const float \*priX)  
*Check if indexed type has denormal bits.*
- int `idxd_cmdenorm` (const int fold, const float \*priX)  
*Check if indexed type has denormal bits.*
- void `idxd_diprint` (const int fold, const `double_indexed` \*X)  
*Print indexed double precision.*
- void `idxd_dmprint` (const int fold, const double \*priX, const int incpriX, const double \*carX, const int inccarX)  
*Print manually specified indexed double precision.*
- void `idxd_ziprint` (const int fold, const `double_complex_indexed` \*X)  
*Print indexed complex double precision.*
- void `idxd_zmprint` (const int fold, const double \*priX, const int incpriX, const double \*carX, const int inccarX)  
*Print manually specified indexed complex double precision.*
- void `idxd_siprint` (const int fold, const `float_indexed` \*X)  
*Print indexed single precision.*
- void `idxd_smprint` (const int fold, const float \*priX, const int incpriX, const float \*carX, const int inccarX)  
*Print manually specified indexed single precision.*
- void `idxd_ciprint` (const int fold, const `float_complex_indexed` \*X)  
*Print indexed complex single precision.*
- void `idxd_cmprint` (const int fold, const float \*priX, const int incpriX, const float \*carX, const int inccarX)  
*Print manually specified indexed complex single precision.*
- void `idxd_didiset` (const int fold, const `double_indexed` \*X, `double_indexed` \*Y)  
*Set indexed double precision ( $Y = X$ )*
- void `idxd_dmdmset` (const int fold, const double \*priX, const int incpriX, const double \*carX, const int inccarX, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Set manually specified indexed double precision ( $Y = X$ )*
- void `idxd_ziziset` (const int fold, const `double_complex_indexed` \*X, `double_complex_indexed` \*Y)  
*Set indexed complex double precision ( $Y = X$ )*
- void `idxd_zmzmset` (const int fold, const double \*priX, const int incpriX, const double \*carX, const int inccarX, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Set manually specified indexed complex double precision ( $Y = X$ )*
- void `idxd_zidiset` (const int fold, const `double_indexed` \*X, `double_complex_indexed` \*Y)  
*Set indexed complex double precision to indexed double precision ( $Y = X$ )*
- void `idxd_zmdmset` (const int fold, const double \*priX, const int incpriX, const double \*carX, const int inccarX, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Set manually specified indexed complex double precision to manually specified indexed double precision ( $Y = X$ )*
- void `idxd_sisiset` (const int fold, const `float_indexed` \*X, `float_indexed` \*Y)  
*Set indexed single precision ( $Y = X$ )*
- void `idxd_smsmset` (const int fold, const float \*priX, const int incpriX, const float \*carX, const int inccarX, float \*priY, const int incpriY, float \*carY, const int inccarY)  
*Set manually specified indexed single precision ( $Y = X$ )*
- void `idxd_ciciset` (const int fold, const `float_complex_indexed` \*X, `float_complex_indexed` \*Y)

- Set indexed complex single precision ( $Y = X$ )*
- void `idxd_cmcmset` (const int fold, const float \*priX, const int incpriX, const float \*carX, const int inccarX, float \*priY, const int incpriY, float \*carY, const int inccarY)
- Set manually specified indexed complex single precision ( $Y = X$ )*
- void `idxd_cisaset` (const int fold, const `float_indexed` \*X, `float_complex_indexed` \*Y)
- Set indexed complex single precision to indexed single precision ( $Y = X$ )*
- void `idxd_cmsmset` (const int fold, const float \*priX, const int incpriX, const float \*carX, const int inccarX, float \*priY, const int incpriY, float \*carY, const int inccarY)
- Set manually specified indexed complex single precision to manually specified indexed single precision ( $Y = X$ )*
- void `idxd_disetzero` (const int fold, `double_indexed` \*X)
- Set indexed double precision to 0 ( $X = 0$ )*
- void `idxd_dmsetzero` (const int fold, double \*priX, const int incpriX, double \*carX, const int inccarX)
- Set manually specified indexed double precision to 0 ( $X = 0$ )*
- void `idxd_zisetzero` (const int fold, `double_complex_indexed` \*X)
- Set indexed double precision to 0 ( $X = 0$ )*
- void `idxd_zmsetzero` (const int fold, double \*priX, const int incpriX, double \*carX, const int inccarX)
- Set manually specified indexed complex double precision to 0 ( $X = 0$ )*
- void `idxd_sisetzero` (const int fold, `float_indexed` \*X)
- Set indexed single precision to 0 ( $X = 0$ )*
- void `idxd_smsetzero` (const int fold, float \*priX, const int incpriX, float \*carX, const int inccarX)
- Set manually specified indexed single precision to 0 ( $X = 0$ )*
- void `idxd_cisetzero` (const int fold, `float_complex_indexed` \*X)
- Set indexed single precision to 0 ( $X = 0$ )*
- void `idxd_cmsetzero` (const int fold, float \*priX, const int incpriX, float \*carX, const int inccarX)
- Set manually specified indexed complex single precision to 0 ( $X = 0$ )*
- void `idxd_didiadd` (const int fold, const `double_indexed` \*X, `double_indexed` \*Y)
- Add indexed double precision ( $Y += X$ )*
- void `idxd_dmdmadd` (const int fold, const double \*priX, const int incpriX, const double \*carX, const int inccarX, double \*priY, const int incpriY, double \*carY, const int inccarY)
- Add manually specified indexed double precision ( $Y += X$ )*
- void `idxd_ziziadd` (const int fold, const `double_complex_indexed` \*X, `double_complex_indexed` \*Y)
- Add indexed complex double precision ( $Y += X$ )*
- void `idxd_zmzmadd` (const int fold, const double \*priX, const int incpriX, const double \*carX, const int inccarX, double \*priY, const int incpriY, double \*carY, const int inccarY)
- Add manually specified indexed complex double precision ( $Y += X$ )*
- void `idxd_sisiadd` (const int fold, const `float_indexed` \*X, `float_indexed` \*Y)
- Add indexed single precision ( $Y += X$ )*
- void `idxd_smsmadd` (const int fold, const float \*priX, const int incpriX, const float \*carX, const int inccarX, float \*priY, const int incpriY, float \*carY, const int inccarY)
- Add manually specified indexed single precision ( $Y += X$ )*
- void `idxd_ciciadd` (const int fold, const `float_complex_indexed` \*X, `float_complex_indexed` \*Y)
- Add indexed complex single precision ( $Y += X$ )*
- void `idxd_cmcmadd` (const int fold, const float \*priX, const int incpriX, const float \*carX, const int inccarX, float \*priY, const int incpriY, float \*carY, const int inccarY)
- Add manually specified indexed complex single precision ( $Y += X$ )*
- void `idxd_didiaddv` (const int fold, const int N, const `double_indexed` \*X, const int incX, `double_indexed` \*Y, const int incY)
- Add indexed double precision vectors ( $Y += X$ )*
- void `idxd_ziziaddv` (const int fold, const int N, const `double_complex_indexed` \*X, const int incX, `double_complex_indexed` \*Y, const int incY)
- Add indexed complex double precision vectors ( $Y += X$ )*

- void `idxd_sisiaddv` (const int fold, const int N, const `float_indexed` \*X, const int incX, `float_indexed` \*Y, const int incY)  
*Add indexed single precision vectors (Y += X)*
- void `idxd_ciciaddv` (const int fold, const int N, const `float_complex_indexed` \*X, const int incX, `float_complex_indexed` \*Y, const int incY)  
*Add indexed complex single precision vectors (Y += X)*
- void `idxd_didadd` (const int fold, const double X, `double_indexed` \*Y)  
*Add double precision to indexed double precision (Y += X)*
- void `idxd_dmdadd` (const int fold, const double X, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Add double precision to manually specified indexed double precision (Y += X)*
- void `idxd_zizadd` (const int fold, const void \*X, `double_complex_indexed` \*Y)  
*Add complex double precision to indexed complex double precision (Y += X)*
- void `idxd_zmzadd` (const int fold, const void \*X, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Add complex double precision to manually specified indexed complex double precision (Y += X)*
- void `idxd_sisadd` (const int fold, const float X, `float_indexed` \*Y)  
*Add single precision to indexed single precision (Y += X)*
- void `idxd_smsadd` (const int fold, const float X, float \*priY, const int incpriY, float \*carY, const int inccarY)  
*Add single precision to manually specified indexed single precision (Y += X)*
- void `idxd_cicadd` (const int fold, const void \*X, `float_complex_indexed` \*Y)  
*Add complex single precision to indexed complex single precision (Y += X)*
- void `idxd_cmcadd` (const int fold, const void \*X, float \*priY, const int incpriY, float \*carY, const int inccarY)  
*Add complex single precision to manually specified indexed complex single precision (Y += X)*
- void `idxd_didupdate` (const int fold, const double X, `double_indexed` \*Y)  
*Update indexed double precision with double precision (X -> Y)*
- void `idxd_dmdupdate` (const int fold, const double X, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Update manually specified indexed double precision with double precision (X -> Y)*
- void `idxd_zizupdate` (const int fold, const void \*X, `double_complex_indexed` \*Y)  
*Update indexed complex double precision with complex double precision (X -> Y)*
- void `idxd_zmzupdate` (const int fold, const void \*X, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Update manually specified indexed complex double precision with complex double precision (X -> Y)*
- void `idxd_zidupdate` (const int fold, const double X, `double_complex_indexed` \*Y)  
*Update indexed complex double precision with double precision (X -> Y)*
- void `idxd_zmdupdate` (const int fold, const double X, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Update manually specified indexed complex double precision with double precision (X -> Y)*
- void `idxd_sisupdate` (const int fold, const float X, `float_indexed` \*Y)  
*Update indexed single precision with single precision (X -> Y)*
- void `idxd_smsupdate` (const int fold, const float X, float \*priY, const int incpriY, float \*carY, const int inccarY)  
*Update manually specified indexed single precision with single precision (X -> Y)*
- void `idxd_cicupdate` (const int fold, const void \*X, `float_complex_indexed` \*Y)  
*Update indexed complex single precision with complex single precision (X -> Y)*
- void `idxd_cmcupdate` (const int fold, const void \*X, float \*priY, const int incpriY, float \*carY, const int inccarY)  
*Update manually specified indexed complex single precision with complex single precision (X -> Y)*
- void `idxd_cisupdate` (const int fold, const float X, `float_complex_indexed` \*Y)  
*Update indexed complex single precision with single precision (X -> Y)*
- void `idxd_cmsupdate` (const int fold, const float X, float \*priY, const int incpriY, float \*carY, const int inccarY)  
*Update manually specified indexed complex single precision with single precision (X -> Y)*
- void `idxd_diddeposit` (const int fold, const double X, `double_indexed` \*Y)

- Add double precision to suitably indexed indexed double precision ( $Y += X$ )*
- void `idxd_dmddeposit` (const int fold, const double X, double \*priY, const int incpriY)
- Add double precision to suitably indexed manually specified indexed double precision ( $Y += X$ )*
- void `idxd_zizdeposit` (const int fold, const void \*X, `double_complex_indexed` \*Y)
- Add complex double precision to suitably indexed manually specified indexed complex double precision ( $Y += X$ )*
- void `idxd_zmzdeposit` (const int fold, const void \*X, double \*priY, const int incpriY)
- Add complex double precision to suitably indexed manually specified indexed complex double precision ( $Y += X$ )*
- void `idxd_sisdeposit` (const int fold, const float X, `float_indexed` \*Y)
- Add single precision to suitably indexed indexed single precision ( $Y += X$ )*
- void `idxd_smsdeposit` (const int fold, const float X, float \*priY, const int incpriY)
- Add single precision to suitably indexed manually specified indexed single precision ( $Y += X$ )*
- void `idxd_cicdeposit` (const int fold, const void \*X, `float_complex_indexed` \*Y)
- Add complex single precision to suitably indexed manually specified indexed complex single precision ( $Y += X$ )*
- void `idxd_cmcddeposit` (const int fold, const void \*X, float \*priY, const int incpriY)
- Add complex single precision to suitably indexed manually specified indexed complex single precision ( $Y += X$ )*
- void `idxd_direnorm` (const int fold, `double_indexed` \*X)
- Renormalize indexed double precision.*
- void `idxd_dmrenorm` (const int fold, double \*priX, const int incpriX, double \*carX, const int inccarX)
- Renormalize manually specified indexed double precision.*
- void `idxd_zirenorm` (const int fold, `double_complex_indexed` \*X)
- Renormalize indexed complex double precision.*
- void `idxd_zmrenorm` (const int fold, double \*priX, const int incpriX, double \*carX, const int inccarX)
- Renormalize manually specified indexed complex double precision.*
- void `idxd_sirenorm` (const int fold, `float_indexed` \*X)
- Renormalize indexed single precision.*
- void `idxd_smrenorm` (const int fold, float \*priX, const int incpriX, float \*carX, const int inccarX)
- Renormalize manually specified indexed single precision.*
- void `idxd_cirenorm` (const int fold, `float_complex_indexed` \*X)
- Renormalize indexed complex single precision.*
- void `idxd_cmrenorm` (const int fold, float \*priX, const int incpriX, float \*carX, const int inccarX)
- Renormalize manually specified indexed complex single precision.*
- void `idxd_didconv` (const int fold, const double X, `double_indexed` \*Y)
- Convert double precision to indexed double precision ( $X \rightarrow Y$ )*
- void `idxd_dmdconv` (const int fold, const double X, double \*priY, const int incpriY, double \*carY, const int inccarY)
- Convert double precision to manually specified indexed double precision ( $X \rightarrow Y$ )*
- void `idxd_zizconv` (const int fold, const void \*X, `double_complex_indexed` \*Y)
- Convert complex double precision to indexed complex double precision ( $X \rightarrow Y$ )*
- void `idxd_zmzconv` (const int fold, const void \*X, double \*priY, const int incpriY, double \*carY, const int inccarY)
- Convert complex double precision to manually specified indexed complex double precision ( $X \rightarrow Y$ )*
- void `idxd_sisconv` (const int fold, const float X, `float_indexed` \*Y)
- Convert single precision to indexed single precision ( $X \rightarrow Y$ )*
- void `idxd_smsconv` (const int fold, const float X, float \*priY, const int incpriY, float \*carY, const int inccarY)
- Convert single precision to manually specified indexed single precision ( $X \rightarrow Y$ )*
- void `idxd_cicconv` (const int fold, const void \*X, `float_complex_indexed` \*Y)
- Convert complex single precision to indexed complex single precision ( $X \rightarrow Y$ )*
- void `idxd_cmccconv` (const int fold, const void \*X, float \*priY, const int incpriY, float \*carY, const int inccarY)
- Convert complex single precision to manually specified indexed complex single precision ( $X \rightarrow Y$ )*
- double `idxd_ddiconv` (const int fold, const `double_indexed` \*X)
- Convert indexed double precision to double precision ( $X \rightarrow Y$ )*

- double [idxd\\_ddmconv](#) (const int fold, const double \*priX, const int incpriX, const double \*carX, const int inccarX)  
*Convert manually specified indexed double precision to double precision ( $X \rightarrow Y$ )*
- void [idxd\\_zziconv\\_sub](#) (const int fold, const [double\\_complex\\_indexed](#) \*X, void \*conv)  
*Convert indexed complex double precision to complex double precision ( $X \rightarrow Y$ )*
- void [idxd\\_zzmconv\\_sub](#) (const int fold, const double \*priX, const int incpriX, const double \*carX, const int inccarX, void \*conv)  
*Convert manually specified indexed complex double precision to complex double precision ( $X \rightarrow Y$ )*
- float [idxd\\_ssiconv](#) (const int fold, const [float\\_indexed](#) \*X)  
*Convert indexed single precision to single precision ( $X \rightarrow Y$ )*
- float [idxd\\_ssmconv](#) (const int fold, const float \*priX, const int incpriX, const float \*carX, const int inccarX)  
*Convert manually specified indexed single precision to single precision ( $X \rightarrow Y$ )*
- void [idxd\\_cciconv\\_sub](#) (const int fold, const [float\\_complex\\_indexed](#) \*X, void \*conv)  
*Convert indexed complex single precision to complex single precision ( $X \rightarrow Y$ )*
- void [idxd\\_ccmconv\\_sub](#) (const int fold, const float \*priX, const int incpriX, const float \*carX, const int inccarX, void \*conv)  
*Convert manually specified indexed complex single precision to complex single precision ( $X \rightarrow Y$ )*
- void [idxd\\_dinegate](#) (const int fold, [double\\_indexed](#) \*X)  
*Negate indexed double precision ( $X = -X$ )*
- void [idxd\\_dmnegate](#) (const int fold, double \*priX, const int incpriX, double \*carX, const int inccarX)  
*Negate manually specified indexed double precision ( $X = -X$ )*
- void [idxd\\_zinegate](#) (const int fold, [double\\_complex\\_indexed](#) \*X)  
*Negate indexed complex double precision ( $X = -X$ )*
- void [idxd\\_zmnegate](#) (const int fold, double \*priX, const int incpriX, double \*carX, const int inccarX)  
*Negate manually specified indexed complex double precision ( $X = -X$ )*
- void [idxd\\_sinegate](#) (const int fold, [float\\_indexed](#) \*X)  
*Negate indexed single precision ( $X = -X$ )*
- void [idxd\\_smnegate](#) (const int fold, float \*priX, const int incpriX, float \*carX, const int inccarX)  
*Negate manually specified indexed single precision ( $X = -X$ )*
- void [idxd\\_cinegate](#) (const int fold, [float\\_complex\\_indexed](#) \*X)  
*Negate indexed complex single precision ( $X = -X$ )*
- void [idxd\\_cmnegate](#) (const int fold, float \*priX, const int incpriX, float \*carX, const int inccarX)  
*Negate manually specified indexed complex single precision ( $X = -X$ )*
- double [idxd\\_dscale](#) (const double X)  
*Get a reproducible double precision scale.*
- float [idxd\\_sscale](#) (const float X)  
*Get a reproducible single precision scale.*
- void [idxd\\_dmdrescale](#) (const int fold, const double X, const double scaleY, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*rescale manually specified indexed double precision sum of squares*
- void [idxd\\_zmdrescale](#) (const int fold, const double X, const double scaleY, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*rescale manually specified indexed complex double precision sum of squares*
- void [idxd\\_smsrescale](#) (const int fold, const float X, const float scaleY, float \*priY, const int incpriY, float \*carY, const int inccarY)  
*rescale manually specified indexed single precision sum of squares*
- void [idxd\\_cmsrescale](#) (const int fold, const float X, const float scaleY, float \*priY, const int incpriY, float \*carY, const int inccarY)  
*rescale manually specified indexed complex single precision sum of squares*
- double [idxd\\_dmdmaddsq](#) (const int fold, const double scaleX, const double \*priX, const int incpriX, const double \*carX, const int inccarX, const double scaleY, double \*priY, const int incpriY, double \*carY, const int inccarY)

*Add manually specified indexed double precision scaled sums of squares ( $Y += X$ )*

- double `idxd_didiaddsq` (const int fold, const double scaleX, const `double_indexed` \*X, const double scaleY, `double_indexed` \*Y)

*Add indexed double precision scaled sums of squares ( $Y += X$ )*

- float `idxd_smsmaddsq` (const int fold, const float scaleX, const float \*priX, const int incpriX, const float \*carX, const int inccarX, const float scaleY, float \*priY, const int incpriY, float \*carY, const int inccarY)

*Add manually specified indexed single precision scaled sums of squares ( $Y += X$ )*

- float `idxd_sisiaddsq` (const int fold, const float scaleX, const `float_indexed` \*X, const float scaleY, `float_indexed` \*Y)

*Add indexed single precision scaled sums of squares ( $Y += X$ )*

- double `idxd_ufp` (const double X)

*unit in the first place*

- float `idxd_ufpf` (const float X)

*unit in the first place*

### 2.1.1 Detailed Description

`idxd.h` defines the indexed types and the lower level functions associated with their use.

This header is modeled after `cblas.h`, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (`*void`)
- s - float (`float`)
- c - complex float (`*void`)
- di - indexed double (`double_indexed`)
- zi - indexed complex double (`double_complex_indexed`)
- si - indexed float (`float_indexed`)
- ci - indexed complex float (`float_complex_indexed`)
- dm - manually specified indexed double (`double, double`)
- zm - manually specified indexed complex double (`double, double`)
- sm - manually specified indexed float (`float, float`)
- cm - manually specified indexed complex float (`float, float`)

Throughout the library, complex types are specified via `*void` pointers. These routines will sometimes be suffixed by sub, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using indexed types is to obtain either more accurate or reproducible summation of floating point numbers. Indexed types are composed of several adjacent bins...

The parameter `fold` describes how many bins are used in the indexed types supplied to a subroutine. The maximum value for this parameter can be set in `config.h`. If you are unsure of what value to use for , we recommend

3. Note that the `fold` of indexed types must be the same for all indexed types that interact with each other. Operations on more than one indexed type assume all indexed types being operated upon have the same `fold`. Note that the `fold` of an indexed type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all indexed functions that you use. Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an indexed type themselves. If you do not know what these are, don't worry about the manually specified indexed types.

## 2.1.2 Macro Definition Documentation

### 2.1.2.1 `#define DIWIDTH 40`

Indexed double precision bin width.

bin width (in bits)

#### Author

Hong Diep Nguyen  
Peter Ahrens

#### Date

27 Apr 2015

### 2.1.2.2 `#define idxd_DICAPACITY (idxd_DIENDURANCE*(1.0/DBL_EPSILON - 1.0))`

Indexed double precision capacity.

The maximum number of double precision numbers that can be summed using indexed double precision. Applies also to indexed complex double precision.

#### Author

Peter Ahrens

#### Date

27 Apr 2015

### 2.1.2.3 `#define idxd_DIENDURANCE (1 << (DBL_MANT_DIG - DIWIDTH - 2))`

Indexed double precision deposit endurance.

The number of deposits that can be performed before a renorm is necessary. Applies also to indexed complex double precision.

#### Author

Hong Diep Nguyen  
Peter Ahrens

#### Date

27 Apr 2015

#### 2.1.2.4 `#define idxd_DIMAXFOLD (idxd_DIMAXINDEX + 1)`

The maximum double precision fold supported by the library.

##### Author

Peter Ahrens

##### Date

14 Jan 2016

#### 2.1.2.5 `#define idxd_DIMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DIWIDTH) - 1)`

Indexed double precision maximum index.

maximum index (inclusive)

##### Author

Peter Ahrens

##### Date

24 Jun 2015

#### 2.1.2.6 `#define idxd_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DIWIDTH + 1)))`

Indexed double precision compression factor.

This factor is used to scale down inputs before deposition into the bin of highest index

##### Author

Peter Ahrens

##### Date

19 May 2015

#### 2.1.2.7 `#define idxd_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DIWIDTH + 1)))`

Indexed double precision expansion factor.

This factor is used to scale up inputs after deposition into the bin of highest index

##### Author

Peter Ahrens

##### Date

19 May 2015



**2.1.2.8 #define idxd\_SICAPACITY (idxd\_SIENDURANCE\*(1.0/FLT\_EPSILON - 1.0))**

Indexed single precision capacity.

The maximum number of single precision numbers that can be summed using indexed single precision. Applies also to indexed complex double precision.

**Author**

Peter Ahrens

**Date**

27 Apr 2015

**2.1.2.9 #define idxd\_SIENDURANCE (1 << (FLT\_MANT\_DIG - SIWIDTH - 2))**

Indexed single precision deposit endurance.

The number of deposits that can be performed before a renorm is necessary. Applies also to indexed complex single precision.

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.2.10 #define idxd\_SIMAXFOLD (idxd\_SIMAXINDEX + 1)**

The maximum single precision fold supported by the library.

**Author**

Peter Ahrens

**Date**

14 Jan 2016

**2.1.2.11 #define idxd\_SIMAXINDEX (((FLT\_MAX\_EXP - FLT\_MIN\_EXP + FLT\_MANT\_DIG - 1)/SIWIDTH) - 1)**

Indexed single precision maximum index.

maximum index (inclusive)

**Author**

Peter Ahrens

**Date**

24 Jun 2015

2.1.2.12 `#define idxd_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SIWIDTH + 1)))`

Indexed single precision compression factor.

This factor is used to scale down inputs before deposition into the bin of highest index

**Author**

Peter Ahrens

**Date**

19 May 2015

2.1.2.13 `#define idxd_SMEXPANSION (1.0*(1 << (FLT_MANT_DIG - SIWIDTH + 1)))`

Indexed single precision expansion factor.

This factor is used to scale up inputs after deposition into the bin of highest index

**Author**

Peter Ahrens

**Date**

19 May 2015

2.1.2.14 `#define SIWIDTH 13`

Indexed single precision bin width.

bin width (in bits)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

## 2.1.3 Typedef Documentation

2.1.3.1 `typedef double double_complex_indexed`

The indexed complex double datatype.

To allocate a `double_complex_indexed`, call `idxd_zialloc()`

**Warning**

A `double_complex_indexed` is, under the hood, an array of `double`. Therefore, if you have defined an array of `double_complex_indexed`, you must index it by multiplying the index into the array by the number of underlying `double` that make up the `double_complex_indexed`. This number can be obtained by a call to `idxd_zinum()`

### 2.1.3.2 typedef double double\_indexed

The indexed double datatype.

To allocate a `double_indexed`, call `idxd_dialloc()`

#### Warning

A `double_indexed` is, under the hood, an array of `double`. Therefore, if you have defined an array of `double_indexed`, you must index it by multiplying the index into the array by the number of underlying `double` that make up the `double_indexed`. This number can be obtained by a call to `idxd_dinum()`

### 2.1.3.3 typedef float float\_complex\_indexed

The indexed complex float datatype.

To allocate a `float_complex_indexed`, call `idxd_cialloc()`

#### Warning

A `float_complex_indexed` is, under the hood, an array of `float`. Therefore, if you have defined an array of `float_complex_indexed`, you must index it by multiplying the index into the array by the number of underlying `float` that make up the `float_complex_indexed`. This number can be obtained by a call to `idxd_cinum()`

### 2.1.3.4 typedef float float\_indexed

The indexed float datatype.

To allocate a `float_indexed`, call `idxd_sialloc()`

#### Warning

A `float_indexed` is, under the hood, an array of `float`. Therefore, if you have defined an array of `float_indexed`, you must index it by multiplying the index into the array by the number of underlying `float` that make up the `float_indexed`. This number can be obtained by a call to `idxd_sinum()`

## 2.1.4 Function Documentation

### 2.1.4.1 void idxd\_cciconv\_sub ( const int fold, const float\_complex\_indexed \* X, void \* conv )

Convert indexed complex single precision to complex single precision (X -> Y)

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>conv</i>	scalar return

#### Author

Hong Diep Nguyen  
Peter Ahrens

#### Date

27 Apr 2015

2.1.4.2 void idxd\_ccmconv\_sub ( const int *fold*, const float \* *priX*, const int *incpriX*, const float \* *carX*, const int *inccarX*,  
void \* *conv* )

Convert manually specified indexed complex single precision to complex single precision (X -> Y)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>conv</i>	scalar return

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.3 `float_complex_indexed* idxd_cialloc ( const int fold )`

indexed complex single precision allocation

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

a freshly allocated indexed type. (free with `free()`)

## Author

Peter Ahrens

## Date

27 Apr 2015

2.1.4.4 `void idxd_cicadd ( const int fold, const void * X, float_complex_indexed * Y )`

Add complex single precision to indexed complex single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type Y

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.5 void `idxd_cicconv` ( const int *fold*, const void \* *X*, float\_complex\_indexed \* *Y* )

Convert complex single precision to indexed complex single precision (X -> Y)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.6 void idxd\_cicdeposit ( const int *fold*, const void \* *X*, float\_complex\_indexed \* *Y* )

Add complex single precision to suitably indexed manually specified indexed complex single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type Y where the index of Y is larger than the index of X

## Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd\\_cicupdate\(\)](#) on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call [idxd\\_cicdeposit\(\)](#) to deposit a maximum of [idxd\\_SIENDURANCE](#) elements into Y before renormalizing Y with [idxd\\_cirenorm\(\)](#). After any number of successive calls of [idxd\\_cicdeposit\(\)](#) on Y, you must renormalize Y with [idxd\\_cirenorm\(\)](#) before using any other function on Y.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

10 Jun 2015

2.1.4.7 void idxd\_ciciadd ( const int *fold*, const float\_complex\_indexed \* *X*, float\_complex\_indexed \* *Y* )

Add indexed complex single precision ( $Y += X$ )

Performs the operation  $Y += X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.8** void idxd\_ciciaddv ( const int *fold*, const int *N*, const float\_complex\_indexed \* *X*, const int *incX*, float\_complex\_indexed \* *Y*, const int *incY* )

Add indexed complex single precision vectors ( $Y += X$ )

Performs the operation  $Y += X$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	indexed vector X
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed vector Y
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

**Author**

Peter Ahrens

**Date**

25 Jun 2015

**2.1.4.9** void idxd\_ciciset ( const int *fold*, const float\_complex\_indexed \* *X*, float\_complex\_indexed \* *Y* )

Set indexed complex single precision ( $Y = X$ )

Performs the operation  $Y = X$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015



**2.1.4.10 void idxd\_cicupdate ( const int *fold*, const void \* *X*, float\_complex\_indexed \* *Y* )**

Update indexed complex single precision with complex single precision ( $X \rightarrow Y$ )

This method updates *Y* to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of *X* respectively.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.11 void idxd\_cinegate ( const int *fold*, float\_complex\_indexed \* *X* )**

Negate indexed complex single precision ( $X = -X$ )

Performs the operation  $X = -X$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.12 int idxd\_cinum ( const int *fold* )**

indexed complex single precision size

**Parameters**

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

**Returns**

the size (in `float`) of the indexed type

**Author**

Peter Ahrens

**Date**

27 Apr 2015

2.1.4.13 `void idxd_ciprint ( const int fold, const float_complex_indexed * X )`

Print indexed complex single precision.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.14 void idxd\_cirenorm ( const int *fold*, float\_complex\_indexed \* *X* )

Renormalize indexed complex single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.15 void idxd\_cisetzero ( const int *fold*, float\_complex\_indexed \* *X* )

Set indexed single precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.16 void idxd\_cisiset ( const int *fold*, const float\_indexed \* *X*, float\_complex\_indexed \* *Y* )

Set indexed complex single precision to indexed single precision ( $Y = X$ )

Performs the operation  $Y = X$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.17 size\_t idxd\_cisize ( const int *fold* )**

indexed complex single precision size

**Parameters**

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

**Returns**

the size (in bytes) of the indexed type

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.18 void idxd\_cisupdate ( const int *fold*, const float *X*, float\_complex\_indexed \* *Y* )**

Update indexed complex single precision with single precision ( $X \rightarrow Y$ )

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

2.1.4.19 void idxd\_cmcadd ( const int *fold*, const void \* *X*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

Add complex single precision to manually specified indexed complex single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type  $Y$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.20 `void idxd_cmconv ( const int fold, const void * X, float * priY, const int incpriY, float * carY, const int inccarY )`

Convert complex single precision to manually specified indexed complex single precision (X -> Y)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.21 `void idxd_cmcdposit ( const int fold, const void * X, float * priY, const int incpriY )`

Add complex single precision to suitably indexed manually specified indexed complex single precision (Y += X)

Performs the operation Y += X on an indexed type Y where the index of Y is larger than the index of X

## Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd\\_cmupdate\(\)](#) on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call [idxd\\_cmcdposit\(\)](#) to deposit a maximum of [idxd\\_SIENDURANCE](#) elements into Y before renormalizing Y with [idxd\\_cmrenorm\(\)](#). After any number of successive calls of [idxd\\_cmcdposit\(\)](#) on Y, you must renormalize Y with [idxd\\_cmrenorm\(\)](#) before using any other function on Y.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

10 Jun 2015

**2.1.4.22** void idxd\_cmcmadd ( const int *fold*, const float \* *priX*, const int *incpriX*, const float \* *carX*, const int *inccarX*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

Add manually specified indexed complex single precision ( $Y += X$ )

Performs the operation  $Y += X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.23** void idxd\_cmcmset ( const int *fold*, const float \* *priX*, const int *incpriX*, const float \* *carX*, const int *inccarX*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

Set manually specified indexed complex single precision ( $Y = X$ )

Performs the operation  $Y = X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.24** void idxd\_cmcupdate ( const int *fold*, const void \* *X*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

Update manually specified indexed complex single precision with complex single precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.25** int idxd\_cmdenorm ( const int *fold*, const float \* *priX* )

Check if indexed type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

**Parameters**

<i>priX</i>	X's primary vector
-------------	--------------------

**Returns**

>0 if x has denormal bits, 0 otherwise.



## Author

Peter Ahrens

## Date

23 Jun 2015

2.1.4.26 void idxd\_cmnegate ( const int *fold*, float \* *priX*, const int *incpriX*, float \* *carX*, const int *inccarX* )

Negate manually specified indexed complex single precision ( $X = -X$ )

Performs the operation  $X = -X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.27 void idxd\_cmprint ( const int *fold*, const float \* *priX*, const int *incpriX*, const float \* *carX*, const int *inccarX* )

Print manually specified indexed complex single precision.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.28 void idxd\_cmrenorm ( const int *fold*, float \* *priX*, const int *incpriX*, float \* *carX*, const int *inccarX* )

Renormalize manually specified indexed complex single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.29** void idxd\_cmsetzero ( const int *fold*, float \* *priX*, const int *incpriX*, float \* *carX*, const int *inccarX* )

Set manually specified indexed complex single precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.30** void idxd\_cmsmset ( const int *fold*, const float \* *priX*, const int *incpriX*, const float \* *carX*, const int *inccarX*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

Set manually specified indexed complex single precision to manually specified indexed single precision ( $Y = X$ )

Performs the operation  $Y = X$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector

<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.31** void idxd\_cmsrescale ( const int *fold*, const float *X*, const float *scaleY*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

rescale manually specified indexed complex single precision sum of squares

Rescale an indexed complex single precision sum of squares Y to Y' such that  $Y / (\text{scaleY} * \text{scaleY}) == Y' / (X * X)$  and `#idxd_smindex(Y) == idxd_sindex(1.0)`

Note that Y is assumed to have an index at least the index of 1.0, and that  $X \geq \text{scaleY}$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	Y's new scaleY ( $X == \text{\#idxd\_sscale}(Y)$ for some float Y) ( $X \geq \text{scaleY}$ )
<i>scaleY</i>	Y's current scaleY ( $\text{scaleY} == \text{\#idxd\_sscale}(Y)$ for some float Y) ( $X \geq \text{scaleY}$ )
<i>priY</i>	Y's primary vector ( <code>#idxd_smindex(Y) &gt;= idxd_sindex(1.0)</code> )
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Peter Ahrens

**Date**

19 Jun 2015

**2.1.4.32** void idxd\_cmsupdate ( const int *fold*, const float *X*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

Update manually specified indexed complex single precision with single precision ( $X \rightarrow Y$ )

This method updates Y to an index suitable for adding numbers with absolute value less than X

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X

<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

### 2.1.4.33 `double idxd_ddiconv ( const int fold, const double_indexed * X )`

Convert indexed double precision to double precision (X -> Y)

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

**Returns**

scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

### 2.1.4.34 `double idxd_ddmconv ( const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX )`

Convert manually specified indexed double precision to double precision (X -> Y)

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Returns**

scalar Y

**Author**

Peter Ahrens

**Date**

31 Jul 2015

2.1.4.35 `double_indexed*` idxd\_dialloc ( `const int fold` )

indexed double precision allocation

**Parameters**

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

**Returns**

a freshly allocated indexed type. (free with `free()`)

**Author**

Peter Ahrens

**Date**

27 Apr 2015

#### 2.1.4.36 `double idxd_dibound ( const int fold, const int N, const double X, const double S )`

Get indexed double precision summation error bound.

This is a bound on the absolute error of a summation using indexed types

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	the number of double precision floating point summands
<i>X</i>	the summand of maximum absolute value
<i>S</i>	the value of the sum computed using indexed types

**Returns**

error bound

**Author**

Peter Ahrens

**Date**

31 Jul 2015

#### 2.1.4.37 `void idxd_didadd ( const int fold, const double X, double_indexed * Y )`

Add double precision to indexed double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type  $Y$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar $X$
<i>Y</i>	indexed scalar $Y$

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

2.1.4.38 void idxd\_didconv ( const int *fold*, const double *X*, double\_indexed \* *Y* )

Convert double precision to indexed double precision ( $X \rightarrow Y$ )

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.39 `void idxd_diddeposit ( const int fold, const double X, double_indexed * Y )`

Add double precision to suitably indexed indexed double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type *Y* where the index of *Y* is larger than the index of *X*

## Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd\\_didupdate\(\)](#) on *Y* with the maximum absolute value of all future elements you wish to deposit in *Y*, you can call [idxd\\_diddeposit\(\)](#) to deposit a maximum of [idxd\\_DIENDURANCE](#) elements into *Y* before renormalizing *Y* with [idxd\\_direnorm\(\)](#). After any number of successive calls of [idxd\\_diddeposit\(\)](#) on *Y*, you must renormalize *Y* with [idxd\\_direnorm\(\)](#) before using any other function on *Y*.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

10 Jun 2015

2.1.4.40 `void idxd_didiadd ( const int fold, const double_indexed * X, double_indexed * Y )`

Add indexed double precision ( $Y += X$ )

Performs the operation  $Y += X$

## Parameters

<i>fold</i>	the fold of the indexed types
-------------	-------------------------------



<i>X</i>	indexed scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.41** `double idxd_didiaddsq ( const int fold, const double scaleX, const double_indexed * X, const double scaleY, double_indexed * Y )`

Add indexed double precision scaled sums of squares ( $Y += X$ )

Performs the operation  $Y += X$ , where *X* and *Y* represent scaled sums of squares.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>scaleX</i>	scale of <i>X</i> ( $scaleX == \#idxd\_dscale(Z)$ for some <code>double Z</code> )
<i>X</i>	indexed scalar <i>X</i>
<i>scaleY</i>	scale of <i>Y</i> ( $scaleY == \#idxd\_dscale(Z)$ for some <code>double Z</code> )
<i>Y</i>	indexed scalar <i>Y</i>

**Returns**

updated scale of *Y*

**Author**

Peter Ahrens

**Date**

2 Dec 2015

**2.1.4.42** `void idxd_didiadv ( const int fold, const int N, const double_indexed * X, const int incX, double_indexed * Y, const int incY )`

Add indexed double precision vectors ( $Y += X$ )

Performs the operation  $Y += X$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	indexed vector <i>X</i>
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)

<i>Y</i>	indexed vector <i>Y</i>
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)

**Author**

Peter Ahrens

**Date**

25 Jun 2015

#### 2.1.4.43 void idxd\_didiset ( const int *fold*, const double\_indexed \* *X*, double\_indexed \* *Y* )

Set indexed double precision ( $Y = X$ )

Performs the operation  $Y = X$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

#### 2.1.4.44 void idxd\_didupdate ( const int *fold*, const double *X*, double\_indexed \* *Y* )

Update indexed double precision with double precision ( $X \rightarrow Y$ )

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

#### 2.1.4.45 int idxd\_dindex ( const double *X* )

Get index of double precision.

The index of a non-indexed type is the smallest index an indexed type would need to have to sum it reproducibly. Higher indices correspond to smaller bins.

## Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

## Returns

*X*'s index

## Author

Peter Ahrens  
Hong Diep Nguyen

## Date

19 Jun 2015

2.1.4.46 void idxd\_dinegate ( const int *fold*, double\_indexed \* *X* )

Negate indexed double precision ( $X = -X$ )

Performs the operation  $X = -X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.47 int idxd\_dinum ( const int *fold* )

indexed double precision size

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

the size (in `double`) of the indexed type

## Author

Peter Ahrens

## Date

27 Apr 2015

2.1.4.48 void idxd\_diprint ( const int *fold*, const double\_indexed \* *X* )

Print indexed double precision.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.49 void idxd\_direnorm ( const int *fold*, double\_indexed \* *X* )**

Renormalize indexed double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.50 void idxd\_disetzero ( const int *fold*, double\_indexed \* *X* )**

Set indexed double precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.51 size\_t idxd\_disize ( const int *fold* )**

indexed double precision size

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

the size (in bytes) of the indexed type

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.52 `const double* idxd_dmbins ( const int X )`

Get indexed double precision reference bins.

returns a pointer to the bins corresponding to the given index

## Parameters

<i>X</i>	index
----------	-------

## Returns

pointer to constant double precision bins of index X

## Author

Peter Ahrens  
Hong Diep Nguyen

## Date

19 Jun 2015

2.1.4.53 `void idxd_dmdadd ( const int fold, const double X, double * priY, const int incpriY, double * carY, const int inccarY )`

Add double precision to manually specified indexed double precision (Y += X)

Performs the operation Y += X on an indexed type Y

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)

<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.54** `void idxd_dmdconv ( const int fold, const double X, double * priY, const int incpriY, double * carY, const int inccarY )`

Convert double precision to manually specified indexed double precision ( $X \rightarrow Y$ )

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

30 Apr 2015

**2.1.4.55** `void idxd_dmddeposit ( const int fold, const double X, double * priY, const int incpriY )`

Add double precision to suitably indexed manually specified indexed double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type Y where the index of Y is larger than the index of X

**Note**

This routine was provided as a means of allowing the you to optimize your code. After you have called `idxd_dmdupdate()` on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call `idxd_dmddeposit()` to deposit a maximum of `idxd_DIENDURANCE` elements into Y before renormalizing Y with `idxd_dmrenorm()`. After any number of successive calls of `idxd_dmddeposit()` on Y, you must renormalize Y with `idxd_dmrenorm()` before using any other function on Y.

**Parameters**

<i>fold</i>	the fold of the indexed types
-------------	-------------------------------

<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

10 Jun 2015

**2.1.4.56 int idxd\_dmdenorm ( const int *fold*, const double \* *priX* )**

Check if indexed type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

**Parameters**

<i>priX</i>	X's primary vector
-------------	--------------------

**Returns**

>0 if x has denormal bits, 0 otherwise.

**Author**

Peter Ahrens

**Date**

23 Jun 2015

**2.1.4.57 void idxd\_dmdmadd ( const int *fold*, const double \* *priX*, const int *incpriX*, const double \* *carX*, const int *inccarX*, double \* *priY*, const int *incpriY*, double \* *carY*, const int *inccarY* )**

Add manually specified indexed double precision (Y += X)

Performs the operation Y += X

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)

<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.58** `double idxd_dmdmaddsq ( const int fold, const double scaleX, const double * priX, const int incpriX, const double * carX, const int inccarX, const double scaleY, double * priY, const int incpriY, double * carY, const int inccarY )`

Add manually specified indexed double precision scaled sums of squares ( $Y += X$ )

Performs the operation  $Y += X$ , where X and Y represent scaled sums of squares.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>scaleX</i>	scale of X ( $scaleX == \#idxd\_dscale(Z)$ for some <code>double Z</code> )
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>scaleY</i>	scale of Y ( $scaleY == \#idxd\_dscale(Z)$ for some <code>double Z</code> )
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Returns**

updated scale of Y

**Author**

Peter Ahrens

**Date**

1 Jun 2015

**2.1.4.59** `void idxd_dmdmset ( const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX, double * priY, const int incpriY, double * carY, const int inccarY )`

Set manually specified indexed double precision ( $Y = X$ )

Performs the operation  $Y = X$



## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.60** `void idxd_dmdrescale ( const int fold, const double X, const double scaleY, double * priY, const int incpriY, double * carY, const int inccarY )`

rescale manually specified indexed double precision sum of squares

Rescale an indexed double precision sum of squares Y to Y' such that  $Y / (\text{scaleY} * \text{scaleY}) == Y' / (X * X)$

Note that Y is assumed to have an index at least the index of 1.0, and that  $X \geq \text{scaleY}$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	Y's new scaleY ( $X == \text{\#idxd\_dscale}(Y)$ for some double Y) ( $X \geq \text{scaleY}$ )
<i>scaleY</i>	Y's current scaleY ( $\text{scaleY} == \text{\#idxd\_dscale}(Y)$ for some double Y) ( $X \geq \text{scaleY}$ )
<i>priY</i>	Y's primary vector ( $\text{\#idxd\_dmindex}(Y) \geq \text{idxd\_dindex}(1.0)$ )
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Peter Ahrens

## Date

19 Jun 2015

**2.1.4.61** `void idxd_dmdupdate ( const int fold, const double X, double * priY, const int incpriY, double * carY, const int inccarY )`

Update manually specified indexed double precision with double precision ( $X \rightarrow Y$ )

This method updates Y to an index suitable for adding numbers with absolute value less than X

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

5 May 2015

2.1.4.62 `int idxd_dminindex ( const double * priX )`

Get index of manually specified indexed double precision.

The index of an indexed type is the bin that it corresponds to. Higher indices correspond to smaller bins.

## Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

## Returns

X's index

## Author

Peter Ahrens  
Hong Diep Nguyen

## Date

23 Sep 2015

2.1.4.63 `int idxd_dminindex0 ( const double * priX )`

Check if index of manually specified indexed double precision is 0.

A quick check to determine if the index is 0

## Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

## Returns

>0 if x has index 0, 0 otherwise.

## Author

Peter Ahrens

## Date

19 May 2015

**2.1.4.64** void idxd\_dmnegate ( const int *fold*, double \* *priX*, const int *incpriX*, double \* *carX*, const int *inccarX* )

Negate manually specified indexed double precision ( $X = -X$ )

Performs the operation  $X = -X$

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

#### Author

Hong Diep Nguyen  
Peter Ahrens

#### Date

27 Apr 2015

**2.1.4.65** void idxd\_dmprint ( const int *fold*, const double \* *priX*, const int *incpriX*, const double \* *carX*, const int *inccarX* )

Print manually specified indexed double precision.

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

#### Author

Hong Diep Nguyen  
Peter Ahrens

#### Date

27 Apr 2015

**2.1.4.66** void idxd\_dmrenorm ( const int *fold*, double \* *priX*, const int *incpriX*, double \* *carX*, const int *inccarX* )

Renormalize manually specified indexed double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector

<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

23 Sep 2015

**2.1.4.67** `void idxd_dmsetzero ( const int fold, double * priX, const int incpriX, double * carX, const int inccarX )`

Set manually specified indexed double precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.68** `double idxd_dscale ( const double X )`

Get a reproducible double precision scale.

For a given X, the smallest Y such that  $\#idxd\_dindex(X) == \#idxd\_dindex(Y)$

Perhaps the most useful property of this number is that,  $1.0 \leq X/Y < 2^{\text{DIWIDTH}}$

**Parameters**

<i>X</i>	double precision number to be scaled
----------	--------------------------------------

**Returns**

reproducible scaling factor

**Author**

Peter Ahrens

**Date**

19 Jun 2015

2.1.4.69 `float_indexed*` idxd\_sialloc ( `const int fold` )

indexed single precision allocation

**Parameters**

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

**Returns**

a freshly allocated indexed type. (free with `free()`)

**Author**

Peter Ahrens

**Date**

27 Apr 2015

#### 2.1.4.70 `float idxd_sibound ( const int fold, const int N, const float X, const float S )`

Get indexed single precision summation error bound.

This is a bound on the absolute error of a summation using indexed types

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	the number of single precision floating point summands
<i>X</i>	the summand of maximum absolute value
<i>S</i>	the value of the sum computed using indexed types

**Returns**

error bound

**Author**

Peter Ahrens

**Date**

31 Jul 2015

**Author**

Peter Ahrens  
Hong Diep Nguyen

**Date**

21 May 2015

#### 2.1.4.71 `int idxd_sindex ( const float X )`

Get index of single precision.

The index of a non-indexed type is the smallest index an indexed type would need to have to sum it reproducibly. Higher indices correspond to smaller bins.

## Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

## Returns

*X*'s index

## Author

Peter Ahrens  
Hong Diep Nguyen

## Date

19 Jun 2015

2.1.4.72 void idxd\_sinegate ( const int *fold*, float\_indexed \* *X* )

Negate indexed single precision ( $X = -X$ )

Performs the operation  $X = -X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.73 int idxd\_sinum ( const int *fold* )

indexed single precision size

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

the size (in `float`) of the indexed type

## Author

Peter Ahrens

## Date

27 Apr 2015

2.1.4.74 void idxd\_siprint ( const int *fold*, const float\_indexed \* *X* )

Print indexed single precision.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.75 void idxd\_sirenorm ( const int *fold*, float\_indexed \* *X* )

Renormalize indexed single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.76 void idxd\_sisadd ( const int *fold*, const float *X*, float\_indexed \* *Y* )

Add single precision to indexed single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type *Y*

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.77 void idxd\_sisconv ( const int *fold*, const float *X*, float\_indexed \* *Y* )

Convert single precision to indexed single precision ( $X \rightarrow Y$ )



## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.78 void idxd\_sisdeposit ( const int *fold*, const float *X*, float\_indexed \* *Y* )

Add single precision to suitably indexed indexed single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type *Y* where the index of *Y* is larger than the index of *X*

## Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd\\_sisupdate\(\)](#) on *Y* with the maximum absolute value of all future elements you wish to deposit in *Y*, you can call [idxd\\_sisdeposit\(\)](#) to deposit a maximum of [idxd\\_SIENDURANCE](#) elements into *Y* before renormalizing *Y* with [idxd\\_sirenorm\(\)](#). After any number of successive calls of [idxd\\_sisdeposit\(\)](#) on *Y*, you must renormalize *Y* with [idxd\\_sirenorm\(\)](#) before using any other function on *Y*.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

10 Jun 2015

2.1.4.79 void idxd\_sisetzzero ( const int *fold*, float\_indexed \* *X* )

Set indexed single precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

## Parameters

<i>fold</i>	the fold of the indexed types
-------------	-------------------------------

<i>X</i>	indexed scalar <i>X</i>
----------	-------------------------

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.80** `void idxd_sisiadd ( const int fold, const float_indexed * X, float_indexed * Y )`

Add indexed single precision ( $Y += X$ )

Performs the operation  $Y += X$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.81** `float idxd_sisiaddsq ( const int fold, const float scaleX, const float_indexed * X, const float scaleY, float_indexed * Y )`

Add indexed single precision scaled sums of squares ( $Y += X$ )

Performs the operation  $Y += X$ , where *X* and *Y* represent scaled sums of squares.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>scaleX</i>	scale of <i>X</i> ( $scaleX == \#idxd\_sscale(Z)$ for some <code>float Z</code> )
<i>X</i>	indexed scalar <i>X</i>
<i>scaleY</i>	scale of <i>Y</i> ( $scaleY == \#idxd\_sscale(Z)$ for some <code>float Z</code> )
<i>Y</i>	indexed scalar <i>Y</i>

**Returns**

updated scale of *Y*

**Author**

Peter Ahrens

**Date**

2 Dec 2015

**2.1.4.82** void idxd\_sisaddv ( const int *fold*, const int *N*, const float\_indexed \* *X*, const int *incX*, float\_indexed \* *Y*, const int *incY* )

Add indexed single precision vectors ( $Y += X$ )

Performs the operation  $Y += X$

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	indexed vector <i>X</i>
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed vector <i>Y</i>
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)

#### Author

Peter Ahrens

#### Date

25 Jun 2015

**2.1.4.83** void idxd\_sisiset ( const int *fold*, const float\_indexed \* *X*, float\_indexed \* *Y* )

Set indexed single precision ( $Y = X$ )

Performs the operation  $Y = X$

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

#### Author

Hong Diep Nguyen  
Peter Ahrens

#### Date

27 Apr 2015

**2.1.4.84** size\_t idxd\_sisize ( const int *fold* )

indexed single precision size

#### Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

#### Returns

the size (in bytes) of the indexed type

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.85** void idxd\_sisupdate ( const int *fold*, const float *X*, float\_indexed \* *Y* )

Update indexed single precision with single precision ( $X \rightarrow Y$ )

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.86** const float\* idxd\_smbins ( const int *X* )

Get indexed single precision reference bins.

returns a pointer to the bins corresponding to the given index

**Parameters**

<i>X</i>	index
----------	-------

**Returns**

pointer to constant single precision bins of index *X*

**Author**

Peter Ahrens  
Hong Diep Nguyen

**Date**

19 Jun 2015

**2.1.4.87** int idxd\_smdenorm ( const int *fold*, const float \* *priX* )

Check if indexed type has denormal bits.

A quick check to determine if calculations involving *X* cannot be performed with "denormals are zero"

## Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

## Returns

>0 if x has denormal bits, 0 otherwise.

## Author

Peter Ahrens

## Date

23 Jun 2015

**2.1.4.88 int idxd\_sminindex ( const float \* *priX* )**

Get index of manually specified indexed single precision.

The index of an indexed type is the bin that it corresponds to. Higher indices correspond to smaller bins.

## Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

## Returns

X's index

## Author

Peter Ahrens  
Hong Diep Nguyen

## Date

23 Sep 2015

**2.1.4.89 int idxd\_sminindex0 ( const float \* *priX* )**

Check if index of manually specified indexed single precision is 0.

A quick check to determine if the index is 0

## Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

## Returns

>0 if x has index 0, 0 otherwise.

## Author

Peter Ahrens

## Date

19 May 2015

**2.1.4.90** void idxd\_smnegate ( const int *fold*, float \* *priX*, const int *incpriX*, float \* *carX*, const int *inccarX* )

Negate manually specified indexed single precision ( $X = -X$ )

Performs the operation  $X = -X$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.91** void idxd\_smprint ( const int *fold*, const float \* *priX*, const int *incpriX*, const float \* *carX*, const int *inccarX* )

Print manually specified indexed single precision.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.92** void idxd\_smrenorm ( const int *fold*, float \* *priX*, const int *incpriX*, float \* *carX*, const int *inccarX* )

Renormalize manually specified indexed single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector

<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

23 Sep 2015

**2.1.4.93** void idxd\_smsadd ( const int *fold*, const float *X*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

Add single precision to manually specified indexed single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type  $Y$

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar $X$
<i>priY</i>	$Y$ 's primary vector
<i>incpriY</i>	stride within $Y$ 's primary vector (use every incpriY'th element)
<i>carY</i>	$Y$ 's carry vector
<i>inccarY</i>	stride within $Y$ 's carry vector (use every inccarY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.94** void idxd\_smsconv ( const int *fold*, const float *X*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

Convert single precision to manually specified indexed single precision ( $X \rightarrow Y$ )

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar $X$
<i>priY</i>	$Y$ 's primary vector
<i>incpriY</i>	stride within $Y$ 's primary vector (use every incpriY'th element)
<i>carY</i>	$Y$ 's carry vector
<i>inccarY</i>	stride within $Y$ 's carry vector (use every inccarY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

30 Apr 2015

#### 2.1.4.95 void idxd\_smsdeposit ( const int *fold*, const float *X*, float \* *priY*, const int *incpriY* )

Add single precision to suitably indexed manually specified indexed single precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type  $Y$  where the index of  $Y$  is larger than the index of  $X$

##### Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd\\_smsupdate\(\)](#) on  $Y$  with the maximum absolute value of all future elements you wish to deposit in  $Y$ , you can call [idxd\\_smsdeposit\(\)](#) to deposit a maximum of [idxd\\_SIENDURANCE](#) elements into  $Y$  before renormalizing  $Y$  with [idxd\\_smrenorm\(\)](#). After any number of successive calls of [idxd\\_smsdeposit\(\)](#) on  $Y$ , you must renormalize  $Y$  with [idxd\\_smrenorm\(\)](#) before using any other function on  $Y$ .

##### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar $X$
<i>priY</i>	$Y$ 's primary vector
<i>incpriY</i>	stride within $Y$ 's primary vector (use every $incpriY$ 'th element)

##### Author

Hong Diep Nguyen  
Peter Ahrens

##### Date

10 Jun 2015

#### 2.1.4.96 void idxd\_smsetzero ( const int *fold*, float \* *priX*, const int *incpriX*, float \* *carX*, const int *inccarX* )

Set manually specified indexed single precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

##### Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	$X$ 's primary vector
<i>incpriX</i>	stride within $X$ 's primary vector (use every $incpriX$ 'th element)
<i>carX</i>	$X$ 's carry vector
<i>inccarX</i>	stride within $X$ 's carry vector (use every $inccarX$ 'th element)

##### Author

Hong Diep Nguyen  
Peter Ahrens

##### Date

27 Apr 2015

#### 2.1.4.97 void idxd\_smsmadd ( const int *fold*, const float \* *priX*, const int *incpriX*, const float \* *carX*, const int *inccarX*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

Add manually specified indexed single precision ( $Y += X$ )

Performs the operation  $Y += X$



## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.98** `float idxd_smsmaddsq ( const int fold, const float scaleX, const float * priX, const int incpriX, const float * carX, const int inccarX, const float scaleY, float * priY, const int incpriY, float * carY, const int inccarY )`

Add manually specified indexed single precision scaled sums of squares ( $Y += X$ )

Performs the operation  $Y += X$ , where X and Y represent scaled sums of squares.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>scaleX</i>	scale of X ( $scaleX == \#idxd\_sscale(Z)$ for some <code>float Z</code> )
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>scaleY</i>	scale of Y ( $scaleY == \#idxd\_sscale(Z)$ for some <code>double Z</code> )
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Returns

updated scale of Y

## Author

Peter Ahrens

## Date

1 Jun 2015

**2.1.4.99** `void idxd_smsmset ( const int fold, const float * priX, const int incpriX, const float * carX, const int inccarX, float * priY, const int incpriY, float * carY, const int inccarY )`

Set manually specified indexed single precision ( $Y = X$ )

Performs the operation  $Y = X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.100** void `idxd_smsrescale` ( const int *fold*, const float *X*, const float *scaleY*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

rescale manually specified indexed single precision sum of squares

Rescale an indexed single precision sum of squares Y to Y' such that  $Y / (\text{scaleY} * \text{scaleY}) == Y' / (X * X)$

Note that Y is assumed to have an index at least the index of 1.0, and that  $X \geq \text{scaleY}$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	Y's new scaleY ( $X == \text{idxd\_sscale}(Y)$ for some float Y) ( $X \geq \text{scaleY}$ )
<i>scaleY</i>	Y's current scaleY ( $\text{scaleY} == \text{idxd\_sscale}(Y)$ for some float Y) ( $X \geq \text{scaleY}$ )
<i>priY</i>	Y's primary vector ( $\text{idxd\_smindex}(Y) \geq \text{idxd\_sindex}(1.0)$ )
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Peter Ahrens

## Date

1 Jun 2015

**2.1.4.101** void `idxd_smsupdate` ( const int *fold*, const float *X*, float \* *priY*, const int *incpriY*, float \* *carY*, const int *inccarY* )

Update manually specified indexed single precision with single precision ( $X \rightarrow Y$ )

This method updates Y to an index suitable for adding numbers with absolute value less than X

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

5 May 2015

## 2.1.4.102 float idxd\_sscales ( const float X )

Get a reproducible single precision scale.

For a given X, the smallest Y such that #idxd\_sindex(X) == #idxd\_sindex(Y)

Perhaps the most useful property of this number is that if the bin epsilon of X's bin is normalized,  $1.0 \leq X * (1.0/Y)$   
 $< 2^{\text{SIWIDTH}}$

## Parameters

<i>X</i>	single precision number to be scaled
----------	--------------------------------------

## Returns

reproducible scaling factor

## Author

Peter Ahrens

## Date

19 Jun 2015

## 2.1.4.103 float idxd\_ssiconv ( const int fold, const float\_indexed \* X )

Convert indexed single precision to single precision (X -> Y)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Returns

scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.104** `float idxd_ssmconv ( const int fold, const float * priX, const int incpriX, const float * carX, const int inccarX )`

Convert manually specified indexed single precision to single precision (X -> Y)

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Returns**

scalar Y

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.105** `double idxd_ufp ( double X )`

unit in the first place

This method returns just the implicit 1 in the mantissa of a `double`

**Parameters**

<i>X</i>	scalar X
----------	----------

**Returns**

unit in the first place

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.106** `float idxd_ufpf ( float X )`

unit in the first place

This method returns just the implicit 1 in the mantissa of a `float`

## Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

## Returns

unit in the first place

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.107 **double\_complex\_indexed\*** idxd\_zialloc ( const int *fold* )

indexed complex double precision allocation

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

a freshly allocated indexed type. (free with `free()`)

## Author

Peter Ahrens

## Date

27 Apr 2015

2.1.4.108 **void** idxd\_zidiset ( const int *fold*, const **double\_indexed** \* *X*, **double\_complex\_indexed** \* *Y* )

Set indexed complex double precision to indexed double precision ( $Y = X$ )

Performs the operation  $Y = X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.109 `void idxd_zidupdate ( const int fold, const double X, double_complex_indexed * Y )`

Update indexed complex double precision with double precision ( $X \rightarrow Y$ )

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	indexed scalar <i>Y</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.110 void idxd\_zinegate ( const int *fold*, double\_complex\_indexed \* *X* )

Negate indexed complex double precision ( $X = -X$ )

Performs the operation  $X = -X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar <i>X</i>

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.111 int idxd\_zinum ( const int *fold* )

indexed complex double precision size

## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

the size (in `double`) of the indexed type

## Author

Peter Ahrens

## Date

27 Apr 2015

2.1.4.112 void idxd\_ziprint ( const int *fold*, const double\_complex\_indexed \* *X* )

Print indexed complex double precision.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.113 void idxd\_zirenorm ( const int *fold*, double\_complex\_indexed \* *X* )**

Renormalize indexed complex double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.114 void idxd\_zissetzero ( const int *fold*, double\_complex\_indexed \* *X* )**

Set indexed double precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.115 size\_t idxd\_zisize ( const int *fold* )**

indexed complex double precision size



## Parameters

<i>fold</i>	the fold of the indexed type
-------------	------------------------------

## Returns

the size (in bytes) of the indexed type

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.116** void idxd\_zizadd ( const int *fold*, const void \* *X*, double\_complex\_indexed \* *Y* )

Add complex double precision to indexed complex double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type  $Y$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar $X$
<i>Y</i>	indexed scalar $Y$

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.117** void idxd\_zizconv ( const int *fold*, const void \* *X*, double\_complex\_indexed \* *Y* )

Convert complex double precision to indexed complex double precision ( $X \rightarrow Y$ )

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar $X$
<i>Y</i>	indexed scalar $Y$

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

#### 2.1.4.118 void idxd\_zizdeposit ( const int *fold*, const void \* *X*, double\_complex\_indexed \* *Y* )

Add complex double precision to suitably indexed manually specified indexed complex double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type  $Y$  where the index of  $Y$  is larger than the index of  $X$

##### Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [idxd\\_zizupdate\(\)](#) on  $Y$  with the maximum absolute value of all future elements you wish to deposit in  $Y$ , you can call [idxd\\_zizdeposit\(\)](#) to deposit a maximum of [idxd\\_DIENDURANCE](#) elements into  $Y$  before renormalizing  $Y$  with [idxd\\_zirenorm\(\)](#). After any number of successive calls of [idxd\\_zizdeposit\(\)](#) on  $Y$ , you must renormalize  $Y$  with [idxd\\_zirenorm\(\)](#) before using any other function on  $Y$ .

##### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar $X$
<i>priY</i>	$Y$ 's primary vector
<i>incpriY</i>	stride within $Y$ 's primary vector (use every $incpriY$ 'th element)

##### Author

Hong Diep Nguyen  
Peter Ahrens

##### Date

10 Jun 2015

#### 2.1.4.119 void idxd\_ziziadd ( const int *fold*, const double\_complex\_indexed \* *X*, double\_complex\_indexed \* *Y* )

Add indexed complex double precision ( $Y += X$ )

Performs the operation  $Y += X$

##### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar $X$
<i>Y</i>	indexed scalar $Y$

##### Author

Hong Diep Nguyen  
Peter Ahrens

##### Date

27 Apr 2015

#### 2.1.4.120 void idxd\_ziziaddv ( const int *fold*, const int *N*, const double\_complex\_indexed \* *X*, const int *incX*, double\_complex\_indexed \* *Y*, const int *incY* )

Add indexed complex double precision vectors ( $Y += X$ )

Performs the operation  $Y += X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	indexed vector X
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed vector Y
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

## Author

Peter Ahrens

## Date

25 Jun 2015

**2.1.4.121** void idxd\_ziziset ( const int *fold*, const double\_complex\_indexed \* *X*, double\_complex\_indexed \* *Y* )

Set indexed complex double precision ( $Y = X$ )

Performs the operation  $Y = X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.122** void idxd\_zizupdate ( const int *fold*, const void \* *X*, double\_complex\_indexed \* *Y* )

Update indexed complex double precision with complex double precision ( $X \rightarrow Y$ )

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>Y</i>	indexed scalar Y

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

#### 2.1.4.123 `int idxd_zmdenorm ( const int fold, const double * priX )`

Check if indexed type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

##### Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

##### Returns

>0 if x has denormal bits, 0 otherwise.

##### Author

Peter Ahrens

##### Date

23 Jun 2015

#### 2.1.4.124 `void idxd_zmdmset ( const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX, double * priY, const int incpriY, double * carY, const int inccarY )`

Set manually specified indexed complex double precision to manually specified indexed double precision ( $Y = X$ )

Performs the operation  $Y = X$

##### Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

##### Author

Hong Diep Nguyen  
Peter Ahrens

##### Date

27 Apr 2015

#### 2.1.4.125 `void idxd_zmdrescale ( const int fold, const double X, const double scaleY, double * priY, const int incpriY, double * carY, const int inccarY )`

rescale manually specified indexed complex double precision sum of squares

Rescale an indexed complex double precision sum of squares Y to Y' such that  $Y / (scaleY * scaleY) == Y' / (X * X)$  and `#idxd_dmindex(Y) == idxd_dindex(1.0)`

Note that Y is assumed to have an index at least the the index of 1.0, and that  $X \geq scaleY$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	Y's new scaleY ( $X == \#idxd\_dscale(Y)$ for some <code>double Y</code> ) ( $X \geq scaleY$ )
<i>scaleY</i>	Y's current scaleY ( $scaleY == \#idxd\_dscale(Y)$ for some <code>double Y</code> ) ( $X \geq scaleY$ )
<i>priY</i>	Y's primary vector ( $\#idxd\_dindex(Y) \geq idxd\_dindex(1.0)$ )
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

## Author

Peter Ahrens

## Date

1 Jun 2015

2.1.4.126 `void idxd_zmdupdate ( const int fold, const double X, double * priY, const int incpriY, double * carY, const int inccarY )`

Update manually specified indexed complex double precision with double precision ( $X \rightarrow Y$ )

This method updates Y to an index suitable for adding numbers with absolute value less than X

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.127 `void idxd_zmnegate ( const int fold, double * priX, const int incpriX, double * carX, const int inccarX )`

Negate manually specified indexed complex double precision ( $X = -X$ )

Performs the operation  $X = -X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector

<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

2.1.4.128 `void idxd_zmprint ( const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX )`

Print manually specified indexed complex double precision.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

2.1.4.129 `void idxd_zmrenorm ( const int fold, double * priX, const int incpriX, double * carX, const int inccarX )`

Renormalize manually specified indexed complex double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

**2.1.4.130** void idxd\_zmsetzero ( const int *fold*, double \* *priX*, const int *incpriX*, double \* *carX*, const int *inccarX* )

Set manually specified indexed complex double precision to 0 ( $X = 0$ )

Performs the operation  $X = 0$

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

#### Author

Hong Diep Nguyen  
Peter Ahrens

#### Date

27 Apr 2015

**2.1.4.131** void idxd\_zmzadd ( const int *fold*, const void \* *X*, double \* *priY*, const int *incpriY*, double \* *carY*, const int *inccarY* )

Add complex double precision to manually specified indexed complex double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type Y

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

#### Author

Hong Diep Nguyen  
Peter Ahrens

#### Date

27 Apr 2015

**2.1.4.132** void idxd\_zmzconv ( const int *fold*, const void \* *X*, double \* *priY*, const int *incpriY*, double \* *carY*, const int *inccarY* )

Convert complex double precision to manually specified indexed complex double precision ( $X \rightarrow Y$ )

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

27 Apr 2015

#### 2.1.4.133 void `idxd_zmzdeposit` ( const int *fold*, const void \* *X*, double \* *priY*, const int *incpriY* )

Add complex double precision to suitably indexed manually specified indexed complex double precision ( $Y += X$ )

Performs the operation  $Y += X$  on an indexed type Y where the index of Y is larger than the index of X

**Note**

This routine was provided as a means of allowing the you to optimize your code. After you have called `idxd_zmzupdate()` on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call `idxd_zmzdeposit()` to deposit a maximum of `idxd_DIENDURANCE` elements into Y before renormalizing Y with `idxd_zmrenorm()`. After any number of successive calls of `idxd_zmzdeposit()` on Y, you must renormalize Y with `idxd_zmrenorm()` before using any other function on Y.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)

**Author**

Hong Diep Nguyen  
Peter Ahrens

**Date**

10 Jun 2015

#### 2.1.4.134 void `idxd_zmzmadd` ( const int *fold*, const double \* *priX*, const int *incpriX*, const double \* *carX*, const int *inccarX*, double \* *priY*, const int *incpriY*, double \* *carY*, const int *inccarY* )

Add manually specified indexed complex double precision ( $Y += X$ )

Performs the operation  $Y += X$



## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.135** void idxd\_zmzmset ( const int *fold*, const double \* *priX*, const int *incpriX*, const double \* *carX*, const int *inccarX*, double \* *priY*, const int *incpriY*, double \* *carY*, const int *inccarY* )

Set manually specified indexed complex double precision ( $Y = X$ )

Performs the operation  $Y = X$

## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

**2.1.4.136** void idxd\_zmzupdate ( const int *fold*, const void \* *X*, double \* *priY*, const int *incpriY*, double \* *carY*, const int *inccarY* )

Update manually specified indexed complex double precision with complex double precision ( $X \rightarrow Y$ )

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.137 `void idxd_zziconv_sub ( const int fold, const double_complex_indexed * X, void * conv )`

Convert indexed complex double precision to complex double precision (X -> Y)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>X</i>	indexed scalar X
<i>conv</i>	scalar return

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

2.1.4.138 `void idxd_zzmconv_sub ( const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX, void * conv )`

Convert manually specified indexed complex double precision to complex double precision (X -> Y)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>conv</i>	scalar return

## Author

Hong Diep Nguyen  
Peter Ahrens

## Date

27 Apr 2015

## 2.2 include/idxdBLAS.h File Reference

`idxdBLAS.h` defines BLAS Methods that operate on indexed types.

```
#include "idxd.h"
#include "reproBLAS.h"
#include <complex.h>
```

### Functions

- float `idxdBLAS_samax` (const int N, const float \*X, const int incX)  
*Find maximum absolute value in vector of single precision.*
- double `idxdBLAS_damax` (const int N, const double \*X, const int incX)  
*Find maximum absolute value in vector of double precision.*
- void `idxdBLAS_camax_sub` (const int N, const void \*X, const int incX, void \*amax)  
*Find maximum magnitude in vector of complex single precision.*
- void `idxdBLAS_zamax_sub` (const int N, const void \*X, const int incX, void \*amax)  
*Find maximum magnitude in vector of complex double precision.*
- float `idxdBLAS_samaxm` (const int N, const float \*X, const int incX, const float \*Y, const int incY)  
*Find maximum absolute value pairwise product between vectors of single precision.*
- double `idxdBLAS_damaxm` (const int N, const double \*X, const int incX, const double \*Y, const int incY)  
*Find maximum absolute value pairwise product between vectors of double precision.*
- void `idxdBLAS_camaxm_sub` (const int N, const void \*X, const int incX, const void \*Y, const int incY, void \*amaxm)  
*Find maximum magnitude pairwise product between vectors of complex single precision.*
- void `idxdBLAS_zamaxm_sub` (const int N, const void \*X, const int incX, const void \*Y, const int incY, void \*amaxm)  
*Find maximum magnitude pairwise product between vectors of complex double precision.*
- void `idxdBLAS_didsum` (const int fold, const int N, const double \*X, const int incX, `double_indexed` \*Y)  
*Add to indexed double precision Y the sum of double precision vector X.*
- void `idxdBLAS_dmdsum` (const int fold, const int N, const double \*X, const int incX, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Add to manually specified indexed double precision Y the sum of double precision vector X.*
- void `idxdBLAS_didasum` (const int fold, const int N, const double \*X, const int incX, `double_indexed` \*Y)  
*Add to indexed double precision Y the absolute sum of double precision vector X.*
- void `idxdBLAS_dmdasum` (const int fold, const int N, const double \*X, const int incX, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Add to manually specified indexed double precision Y the absolute sum of double precision vector X.*
- double `idxdBLAS_didssq` (const int fold, const int N, const double \*X, const int incX, const double scaleY, `double_indexed` \*Y)  
*Add to scaled indexed double precision Y the scaled sum of squares of elements of double precision vector X.*
- double `idxdBLAS_dmdssq` (const int fold, const int N, const double \*X, const int incX, const double scaleY, double \*priY, const int incpriY, double \*carY, const int inccarY)  
*Add to scaled manually specified indexed double precision Y the scaled sum of squares of elements of double precision vector X.*
- void `idxdBLAS_diddot` (const int fold, const int N, const double \*X, const int incX, const double \*Y, const int incY, `double_indexed` \*Z)  
*Add to indexed double precision Z the dot product of double precision vectors X and Y.*
- void `idxdBLAS_dmddot` (const int fold, const int N, const double \*X, const int incX, const double \*Y, const int incY, double \*manZ, const int incmanZ, double \*carZ, const int inccarZ)  
*Add to manually specified indexed double precision Z the dot product of double precision vectors X and Y.*
- void `idxdBLAS_zizsum` (const int fold, const int N, const void \*X, const int incX, `double_indexed` \*Y)

*Add to indexed complex double precision Y the sum of complex double precision vector X.*

- void `idxdBLAS_zmzsum` (const int fold, const int N, const void \*X, const int incX, double \*priY, const int incpriY, double \*carY, const int inccarY)

*Add to manually specified indexed complex double precision Y the sum of complex double precision vector X.*

- void `idxdBLAS_dizasum` (const int fold, const int N, const void \*X, const int incX, `double_indexed` \*Y)

*Add to indexed double precision Y the absolute sum of complex double precision vector X.*

- void `idxdBLAS_dmzasum` (const int fold, const int N, const void \*X, const int incX, double \*priY, const int incpriY, double \*carY, const int inccarY)

*Add to manually specified indexed double precision Y the absolute sum of complex double precision vector X.*

- double `idxdBLAS_dizssq` (const int fold, const int N, const void \*X, const int incX, const double scaleY, `double_indexed` \*Y)

*Add to scaled indexed double precision Y the scaled sum of squares of elements of complex double precision vector X.*

- double `idxdBLAS_dmzssq` (const int fold, const int N, const void \*X, const int incX, const double scaleY, double \*priY, const int incpriY, double \*carY, const int inccarY)

*Add to scaled manually specified indexed double precision Y the scaled sum of squares of elements of complex double precision vector X.*

- void `idxdBLAS_zizdotu` (const int fold, const int N, const void \*X, const int incX, const void \*Y, const int incY, `double_indexed` \*Z)

*Add to indexed complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.*

- void `idxdBLAS_zmzdotu` (const int fold, const int N, const void \*X, const int incX, const void \*Y, const int incY, double \*manZ, const int incmanZ, double \*carZ, const int inccarZ)

*Add to manually specified indexed complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.*

- void `idxdBLAS_zizdotc` (const int fold, const int N, const void \*X, const int incX, const void \*Y, const int incY, `double_indexed` \*Z)

*Add to indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y.*

- void `idxdBLAS_zmzdotc` (const int fold, const int N, const void \*X, const int incX, const void \*Y, const int incY, double \*manZ, const int incmanZ, double \*carZ, const int inccarZ)

*Add to manually specified indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y.*

- void `idxdBLAS_sisum` (const int fold, const int N, const float \*X, const int incX, `float_indexed` \*Y)

*Add to indexed single precision Y the sum of single precision vector X.*

- void `idxdBLAS_smssum` (const int fold, const int N, const float \*X, const int incX, float \*priY, const int incpriY, float \*carY, const int inccarY)

*Add to manually specified indexed single precision Y the sum of single precision vector X.*

- void `idxdBLAS_sisasum` (const int fold, const int N, const float \*X, const int incX, `float_indexed` \*Y)

*Add to indexed single precision Y the absolute sum of single precision vector X.*

- void `idxdBLAS_smsasum` (const int fold, const int N, const float \*X, const int incX, float \*priY, const int incpriY, float \*carY, const int inccarY)

*Add to manually specified indexed single precision Y the absolute sum of double precision vector X.*

- float `idxdBLAS_sisssq` (const int fold, const int N, const float \*X, const int incX, const float scaleY, `float_indexed` \*Y)

*Add to scaled indexed single precision Y the scaled sum of squares of elements of single precision vector X.*

- float `idxdBLAS_smsssq` (const int fold, const int N, const float \*X, const int incX, const float scaleY, float \*priY, const int incpriY, float \*carY, const int inccarY)

*Add to scaled manually specified indexed single precision Y the scaled sum of squares of elements of single precision vector X.*

- void `idxdBLAS_sisdot` (const int fold, const int N, const float \*X, const int incX, const float \*Y, const int incY, `float_indexed` \*Z)

*Add to indexed single precision Z the dot product of single precision vectors X and Y.*

- void `idxdBLAS_smsdot` (const int fold, const int N, const float \*X, const int incX, const float \*Y, const int incY, float \*manZ, const int incmanZ, float \*carZ, const int inccarZ)

- Add to manually specified indexed single precision Z the dot product of single precision vectors X and Y.*
- void `idxdBLAS_cicsum` (const int fold, const int N, const void \*X, const int incX, [float\\_indexed](#) \*Y)
- Add to indexed complex single precision Y the sum of complex single precision vector X.*
- void `idxdBLAS_cmcsun` (const int fold, const int N, const void \*X, const int incX, float \*priY, const int incpriY, float \*carY, const int inccarY)
- Add to manually specified indexed complex single precision Y the sum of complex single precision vector X.*
- void `idxdBLAS_sicasun` (const int fold, const int N, const void \*X, const int incX, [float\\_indexed](#) \*Y)
- Add to indexed single precision Y the absolute sum of complex single precision vector X.*
- void `idxdBLAS_smcasun` (const int fold, const int N, const void \*X, const int incX, float \*priY, const int incpriY, float \*carY, const int inccarY)
- Add to manually specified indexed single precision Y the absolute sum of complex single precision vector X.*
- float `idxdBLAS_sicssq` (const int fold, const int N, const void \*X, const int incX, const float scaleY, [float\\_indexed](#) \*Y)
- Add to scaled indexed single precision Y the scaled sum of squares of elements of complex single precision vector X.*
- float `idxdBLAS_smcssq` (const int fold, const int N, const void \*X, const int incX, const float scaleY, float \*priY, const int incpriY, float \*carY, const int inccarY)
- Add to scaled manually specified indexed single precision Y the scaled sum of squares of elements of complex single precision vector X.*
- void `idxdBLAS_cicdotu` (const int fold, const int N, const void \*X, const int incX, const void \*Y, const int incY, [float\\_indexed](#) \*Z)
- Add to indexed complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.*
- void `idxdBLAS_cmcdotu` (const int fold, const int N, const void \*X, const int incX, const void \*Y, const int incY, float \*manZ, const int incmanZ, float \*carZ, const int inccarZ)
- Add to manually specified indexed complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.*
- void `idxdBLAS_cicdotc` (const int fold, const int N, const void \*X, const int incX, const void \*Y, const int incY, [float\\_indexed](#) \*Z)
- Add to indexed complex single precision Z the conjugated dot product of complex single precision vectors X and Y.*
- void `idxdBLAS_cmcdotc` (const int fold, const int N, const void \*X, const int incX, const void \*Y, const int incY, float \*manZ, const int incmanZ, float \*carZ, const int inccarZ)
- Add to manually specified indexed complex single precision Z the conjugated dot product of complex single precision vectors X and Y.*
- void `idxdBLAS_didgemv` (const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double \*A, const int lda, const double \*X, const int incX, [double\\_indexed](#) \*Y, const int incY)
- Add to indexed double precision vector Y the matrix-vector product of double precision matrix A and double precision vector X.*
- void `idxdBLAS_didgemm` (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double \*A, const int lda, const double \*B, const int ldb, [double\\_indexed](#) \*C, const int ldc)
- Add to indexed double precision matrix C the matrix-matrix product of double precision matrices A and B.*
- void `idxdBLAS_sisgemv` (const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float \*A, const int lda, const float \*X, const int incX, [float\\_indexed](#) \*Y, const int incY)
- Add to indexed single precision vector Y the matrix-vector product of single precision matrix A and single precision vector X.*
- void `idxdBLAS_sisgemm` (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float \*A, const int lda, const float \*B, const int ldb, [float\\_indexed](#) \*C, const int ldc)
- Add to indexed single precision matrix C the matrix-matrix product of single precision matrices A and B.*
- void `idxdBLAS_zizgemv` (const int fold, const char Order, const char TransA, const int M, const int N, const void \*alpha, const void \*A, const int lda, const void \*X, const int incX, [double\\_complex\\_indexed](#) \*Y, const int incY)
- Add to indexed complex double precision vector Y the matrix-vector product of complex double precision matrix A and complex double precision vector X.*

- void [idxdBLAS\\_zizgemv](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void \*alpha, const void \*A, const int lda, const void \*B, const int ldb, [double\\_complex\\_indexed](#) \*C, const int ldc)

*Add to indexed complex double precision matrix C the matrix-matrix product of complex double precision matrices A and B.*

- void [idxdBLAS\\_cicgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const void \*alpha, const void \*A, const int lda, const void \*X, const int incX, [float\\_complex\\_indexed](#) \*Y, const int incY)

*Add to indexed complex single precision vector Y the matrix-vector product of complex single precision matrix A and complex single precision vector X.*

- void [idxdBLAS\\_cicgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void \*alpha, const void \*A, const int lda, const void \*B, const int ldb, [float\\_complex\\_indexed](#) \*C, const int ldc)

*Add to indexed complex single precision matrix C the matrix-matrix product of complex single precision matrices A and B.*

## 2.2.1 Detailed Description

[idxdBLAS.h](#) defines BLAS Methods that operate on indexed types.

This header is modeled after cblas.h, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (`*void`)
- s - float (`float`)
- c - complex float (`*void`)
- di - indexed double ([double\\_indexed](#))
- zi - indexed complex double ([double\\_complex\\_indexed](#))
- si - indexed float ([float\\_indexed](#))
- ci - indexed complex float ([float\\_complex\\_indexed](#))
- dm - manually specified indexed double (`double, double`)
- zm - manually specified indexed complex double (`double, double`)
- sm - manually specified indexed float (`float, float`)
- cm - manually specified indexed complex float (`float, float`)

Throughout the library, complex types are specified via `*void` pointers. These routines will sometimes be suffixed by sub, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using indexed types is to obtain either more accurate or reproducible summation of floating point numbers. Indexed types are composed of several adjacent bins...

The parameter `fold` describes how many bins are used in the indexed types supplied to a subroutine. The maximum value for this parameter can be set in `config.h`. If you are unsure of what value to use for , we recommend

3. Note that the `fold` of indexed types must be the same for all indexed types that interact with each other. Operations on more than one indexed type assume all indexed types being operated upon have the same `fold`. Note that the `fold` of an indexed type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all indexed functions that you use. Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an indexed type themselves. If you do not know what these are, don't worry about the manually specified indexed types.

## 2.2.2 Function Documentation

### 2.2.2.1 `void idxdBLAS_camax_sub ( const int N, const void * X, const int incX, void * amax )`

Find maximum magnitude in vector of complex single precision.

Returns the magnitude of the element of maximum magnitude in an array.

#### Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>amax</i>	scalar return

#### Author

Peter Ahrens

#### Date

15 Jan 2016

### 2.2.2.2 `void idxdBLAS_camaxm_sub ( const int N, const void * X, const int incX, const void * Y, const int incY, void * amaxm )`

Find maximum magnitude pairwise product between vectors of complex single precision.

Returns the magnitude of the pairwise product of maximum magnitude between X and Y.

#### Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>amaxm</i>	scalar return

#### Author

Peter Ahrens

#### Date

15 Jan 2016

2.2.2.3 `void idxdBLAS_cicdotc ( const int fold, const int N, const void * X, const int incX, const void * Y, const int incY,  
float_complex_indexed * Z )`

Add to indexed complex single precision *Z* the conjugated dot product of complex single precision vectors *X* and *Y*.

Add to *Z* the indexed sum of the pairwise products of *X* and conjugated *Y*.



## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>indexed</i>	scalar <i>Z</i>

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.4** void idxdBLAS\_cicdotu ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, const void \* *Y*, const int *incY*, float\_complex\_indexed \* *Z* )

Add to indexed complex single precision *Z* the unconjugated dot product of complex single precision vectors *X* and *Y*.

Add to *Z* the indexed sum of the pairwise products of *X* and *Y*.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>indexed</i>	scalar <i>Z</i>

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.5** void idxdBLAS\_cicgemm ( const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void \* *alpha*, const void \* *A*, const int *lda*, const void \* *B*, const int *ldb*, float\_complex\_indexed \* *C*, const int *ldc* )

Add to indexed complex single precision matrix *C* the matrix-matrix product of complex single precision matrices *A* and *B*.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C$ ,

where  $\text{op}(X)$  is one of

$\text{op}(X) = X$  or  $\text{op}(X) = X^{**}T$  or  $\text{op}(X) = X^{**}H$ ,

$\alpha$  is a scalar, *A* and *B* are matrices with  $\text{op}(A)$  an *M* by *K* matrix and  $\text{op}(B)$  a *K* by *N* matrix, and *C* is an indexed *M* by *N* matrix.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	complex single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	indexed complex single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

## Author

Peter Ahrens

## Date

18 Jan 2016

**2.2.2.6** void idxdBLAS\_cicgemv ( const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void \* *alpha*, const void \* *A*, const int *lda*, const void \* *X*, const int *incX*, float\_complex\_indexed \* *Y*, const int *incY* )

Add to indexed complex single precision vector Y the matrix-vector product of complex single precision matrix A and complex single precision vector X.

Performs one of the matrix-vector operations

$y := \alpha A x + \beta y$  or  $y := \alpha A^T x + \beta y$  or  $y := \alpha A^H x + \beta y$ ,

where alpha and beta are scalars, x is a vector, y is an indexed vector, and A is an M by N matrix.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha

<i>A</i>	complex single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex single precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	indexed complex single precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

**Author**

Peter Ahrens

**Date**

18 Jan 2016

**2.2.2.7** void idxdBLAS\_cicsum ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, float\_complex\_indexed \* *Y* )

Add to indexed complex single precision Y the sum of complex single precision vector X.

Add to Y the indexed sum of X.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>indexed</i>	scalar Y

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.8** void idxdBLAS\_cmcdotc ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, const void \* *Y*, const int *incY*, float \* *manZ*, const int *incmanZ*, float \* *carZ*, const int *inccarZ* )

Add to manually specified indexed complex single precision Z the conjugated dot product of complex single precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and conjugated Y.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector

<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.9** void idxdBLAS\_cmcdotu ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, const void \* *Y*, const int *incY*, float \* *manZ*, const int *incmanZ*, float \* *carZ*, const int *inccarZ* )

Add to manually specified indexed complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and Y.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.10** void idxdBLAS\_cmcsu ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, float \* *manY*, const int *incmanY*, float \* *carY*, const int *inccarY* )

Add to manually specified indexed complex single precision Y the sum of complex single precision vector X.

Add to Y the indexed sum of X.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

## Author

Peter Ahrens

## Date

15 Jan 2016

2.2.2.11 `double idxdBLAS_damax ( const int N, const double * X, const int incX )`

Find maximum absolute value in vector of double precision.

Returns the absolute value of the element of maximum absolute value in an array.

## Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

## Returns

absolute maximum value of X

## Author

Peter Ahrens

## Date

15 Jan 2016

2.2.2.12 `double idxdBLAS_damaxm ( const int N, const double * X, const int incX, const double * Y, const int incY )`

Find maximum absolute value pairwise product between vectors of double precision.

Returns the absolute value of the pairwise product of maximum absolute value between X and Y.

## Parameters

<i>N</i>	vector length
----------	---------------

<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

**Returns**

absolute maximum value multiple of X and Y

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.13** void idxdBLAS\_didasum ( const int *fold*, const int *N*, const double \* *X*, const int *incX*, double\_indexed \* *Y* )

Add to indexed double precision Y the absolute sum of double precision vector X.

Add to Y the indexed sum of absolute values of elements in X.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar Y

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.14** void idxdBLAS\_diddot ( const int *fold*, const int *N*, const double \* *X*, const int *incX*, const double \* *Y*, const int *incY*, double\_indexed \* *Z* )

Add to indexed double precision Z the dot product of double precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and Y.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>Z</i>	indexed scalar <i>Z</i>

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.15** void idxdBLAS\_didgemm ( const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const double *alpha*, const double \* *A*, const int *lda*, const double \* *B*, const int *ldb*, double\_indexed \* *C*, const int *ldc* )

Add to indexed double precision matrix *C* the matrix-matrix product of double precision matrices *A* and *B*.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C$ ,

where  $\text{op}(X)$  is one of

$\text{op}(X) = X$  or  $\text{op}(X) = X^{**T}$ ,

$\alpha$  is a scalar, *A* and *B* are matrices with  $\text{op}(A)$  an *M* by *K* matrix and  $\text{op}(B)$  a *K* by *N* matrix, and *C* is an indexed *M* by *N* matrix.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose <i>A</i> before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose <i>B</i> before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix <i>C</i> .
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix <i>C</i> .
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$ .
<i>alpha</i>	scalar $\alpha$
<i>A</i>	double precision matrix of dimension ( <i>ma</i> , <i>lda</i> ) in row-major or ( <i>lda</i> , <i>na</i> ) in column-major. ( <i>ma</i> , <i>na</i> ) is ( <i>M</i> , <i>K</i> ) if <i>A</i> is not transposed and ( <i>K</i> , <i>M</i> ) otherwise.
<i>lda</i>	the first dimension of <i>A</i> as declared in the calling program. <i>lda</i> must be at least <i>na</i> in row major or <i>ma</i> in column major.
<i>B</i>	double precision matrix of dimension ( <i>mb</i> , <i>ldb</i> ) in row-major or ( <i>ldb</i> , <i>nb</i> ) in column-major. ( <i>mb</i> , <i>nb</i> ) is ( <i>K</i> , <i>N</i> ) if <i>B</i> is not transposed and ( <i>N</i> , <i>K</i> ) otherwise.
<i>ldb</i>	the first dimension of <i>B</i> as declared in the calling program. <i>ldb</i> must be at least <i>nb</i> in row major or <i>mb</i> in column major.
<i>C</i>	indexed double precision matrix of dimension ( <i>M</i> , <i>ldc</i> ) in row-major or ( <i>ldc</i> , <i>N</i> ) in column-major.
<i>ldc</i>	the first dimension of <i>C</i> as declared in the calling program. <i>ldc</i> must be at least <i>N</i> in row major or <i>M</i> in column major.

**Author**

Peter Ahrens

**Date**

18 Jan 2016

**2.2.2.16** void idxdBLAS\_didgemv ( const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const double *alpha*, const double \* *A*, const int *lda*, const double \* *X*, const int *incX*, double\_indexed \* *Y*, const int *incY* )

Add to indexed double precision vector *Y* the matrix-vector product of double precision matrix *A* and double precision vector *X*.

Performs one of the matrix-vector operations

$y := \alpha * A * x + y$  or  $y := \alpha * A^T * x + y$ ,

where  $\alpha$  is a scalar,  $x$  is a vector,  $y$  is an indexed vector, and  $A$  is an  $M$  by  $N$  matrix.

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose <i>A</i> before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix <i>A</i>
<i>N</i>	number of columns of matrix <i>A</i>
<i>alpha</i>	scalar $\alpha$
<i>A</i>	double precision matrix of dimension ( $M$ , $lda$ ) in row-major or ( $lda$ , $N$ ) in column-major
<i>lda</i>	the first dimension of <i>A</i> as declared in the calling program
<i>X</i>	double precision vector of at least size $N$ if not transposed or size $M$ otherwise
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed double precision vector <i>Y</i> of at least size $M$ if not transposed or size $N$ otherwise
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)

#### Author

Peter Ahrens

#### Date

18 Jan 2016

**2.2.2.17** double idxdBLAS\_didssq ( const int *fold*, const int *N*, const double \* *X*, const int *incX*, const double *scaleY*, double\_indexed \* *Y* )

Add to scaled indexed double precision *Y* the scaled sum of squares of elements of double precision vector *X*.

Add to *Y* the scaled indexed sum of the squares of each element of *X*. The scaling of each square is performed using [idxd\\_dscale\(\)](#)

#### Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of <i>Y</i>
<i>Y</i>	indexed scalar <i>Y</i>

#### Returns

the new scaling factor of *Y*

#### Author

Peter Ahrens



## Date

18 Jan 2016

**2.2.2.18** void idxdBLAS\_didsum ( const int *fold*, const int *N*, const double \* *X*, const int *incX*, double\_indexed \* *Y* )

Add to indexed double precision *Y* the sum of double precision vector *X*.

Add to *Y* the indexed sum of *X*.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar <i>Y</i>

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.19** void idxdBLAS\_dizasum ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, double\_indexed \* *Y* )

Add to indexed double precision *Y* the absolute sum of complex double precision vector *X*.

Add to *Y* the indexed sum of magnitudes of elements of *X*.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar <i>Y</i>

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.20** double idxdBLAS\_dizssq ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, const double *scaleY*, double\_indexed \* *Y* )

Add to scaled indexed double precision *Y* the scaled sum of squares of elements of complex double precision vector *X*.

Add to *Y* the scaled indexed sum of the squares of each element of *X*. The scaling of each square is performed using [idxd\\_dscale\(\)](#)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>Y</i>	indexed scalar Y

## Returns

the new scaling factor of Y

## Author

Peter Ahrens

## Date

18 Jan 2016

**2.2.2.21** `void idxdBLAS_dmdasum ( const int fold, const int N, const double * X, const int incX, double * manY, const int incmanY, double * carY, const int inccarY )`

Add to manually specified indexed double precision Y the absolute sum of double precision vector X.

Add to Y the indexed sum of absolute values of elements in X.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.22** `void idxdBLAS_dmddot ( const int fold, const int N, const double * X, const int incX, const double * Y, const int incY, double * manZ, const int incmanZ, double * carZ, const int inccarZ )`

Add to manually specified indexed double precision Z the dot product of double precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and Y.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.23** `double idxdBLAS_dmdssq ( const int fold, const int N, const double * X, const int incX, const double scaleY, double * manY, const int incmanY, double * carY, const int inccarY )`

Add to scaled manually specified indexed double precision Y the scaled sum of squares of elements of double precision vector X.

Add to Y the scaled indexed sum of the squares of each element of X. The scaling of each square is performed using [idxd\\_dscale\(\)](#)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

## Returns

the new scaling factor of Y

## Author

Peter Ahrens

## Date

18 Jan 2016

**2.2.2.24** `void idxdBLAS_dmdsum ( const int fold, const int N, const double * X, const int incX, double * manY, const int incmanY, double * carY, const int inccarY )`

Add to manually specified indexed double precision Y the sum of double precision vector X.

Set Y to the indexed sum of X.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.25** void idxdBLAS\_dmzasum ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, double \* *manY*, const int *incmanY*, double \* *carY*, const int *inccarY* )

Add to manually specified indexed double precision Y the absolute sum of complex double precision vector X.

Add to Y the indexed sum of magnitudes of elements of X.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.26** double idxdBLAS\_dmzssq ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, const double *scaleY*, double \* *manY*, const int *incmanY*, double \* *carY*, const int *inccarY* )

Add to scaled manually specified indexed double precision Y the scaled sum of squares of elements of complex double precision vector X.

Add to Y the scaled indexed sum of the squares of each element of X. The scaling of each square is performed using [idxd\\_dscale\(\)](#)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

## Returns

the new scaling factor of Y

## Author

Peter Ahrens

## Date

18 Jan 2016

### 2.2.2.27 float idxdBLAS\_samax ( const int *N*, const float \* *X*, const int *incX* )

Find maximum absolute value in vector of single precision.

Returns the absolute value of the element of maximum absolute value in an array.

## Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

## Returns

absolute maximum value of X

## Author

Peter Ahrens

## Date

15 Jan 2016

### 2.2.2.28 float idxdBLAS\_samaxm ( const int *N*, const float \* *X*, const int *incX*, const float \* *Y*, const int *incY* )

Find maximum absolute value pairwise product between vectors of single precision.

Returns the absolute value of the pairwise product of maximum absolute value between X and Y.

## Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

## Returns

absolute maximum value multiple of X and Y

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.29** void idxdBLAS\_sicasum ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, float\_indexed \* *Y* )

Add to indexed single precision Y the absolute sum of complex single precision vector X.

Add to Y the indexed sum of magnitudes of elements of X.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar Y

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.30** float idxdBLAS\_sicssq ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, const float *scaleY*, float\_indexed \* *Y* )

Add to scaled indexed single precision Y the scaled sum of squares of elements of complex single precision vector X.

Add to Y the scaled indexed sum of the squares of each element of X. The scaling of each square is performed using [idxd\\_sscale\(\)](#)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>Y</i>	indexed scalar Y

**Returns**

the new scaling factor of Y

**Author**

Peter Ahrens

**Date**

18 Jan 2016

**2.2.2.31** void idxdBLAS\_sisasum ( const int *fold*, const int *N*, const float \* *X*, const int *incX*, float\_indexed \* *Y* )

Add to indexed single precision Y the absolute sum of single precision vector X.

Add to Y the indexed sum of absolute values of elements in X.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar Y

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.32** void idxdBLAS\_sisdot ( const int *fold*, const int *N*, const float \* *X*, const int *incX*, const float \* *Y*, const int *incY*, float\_indexed \* *Z* )

Add to indexed single precision Z the dot product of single precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and Y.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length

<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>Z</i>	indexed scalar Z

**Author**

Peter Ahrens

**Date**

15 Jan 2016

2.2.2.33 void idxdBLAS\_sisgemm ( const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const float *alpha*, const float \* *A*, const int *lda*, const float \* *B*, const int *ldb*, float\_indexed \* *C*, const int *ldc* )

Add to indexed single precision matrix C the matrix-matrix product of single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C$ ,

where  $\text{op}(X)$  is one of

$\text{op}(X) = X$  or  $\text{op}(X) = X^{**T}$ ,

$\alpha$  is a scalar, A and B are matrices with  $\text{op}(A)$  an M by K matrix and  $\text{op}(B)$  a K by N matrix, and C is an indexed M by N matrix.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$ .
<i>alpha</i>	scalar alpha
<i>A</i>	single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	indexed single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

**Author**

Peter Ahrens



## Date

18 Jan 2016

**2.2.2.34** void idxdBLAS\_sisgemv ( const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const float *alpha*, const float \* *A*, const int *lda*, const float \* *X*, const int *incX*, float\_indexed \* *Y*, const int *incY* )

Add to indexed single precision vector *Y* the matrix-vector product of single precision matrix *A* and single precision vector *X*.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$  or  $y := \alpha * A^T * x + \beta * y$ ,

where  $\alpha$  and  $\beta$  are scalars,  $x$  is a vector,  $y$  is an indexed vector, and  $A$  is an  $M$  by  $N$  matrix.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose <i>A</i> before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix <i>A</i>
<i>N</i>	number of columns of matrix <i>A</i>
<i>alpha</i>	scalar $\alpha$
<i>A</i>	single precision matrix of dimension ( $M$ , $lda$ ) in row-major or ( $lda$ , $N$ ) in column-major
<i>lda</i>	the first dimension of <i>A</i> as declared in the calling program
<i>X</i>	single precision vector of at least size $N$ if not transposed or size $M$ otherwise
<i>incX</i>	<i>X</i> vector stride (use every $incX$ 'th element)
<i>beta</i>	scalar $\beta$
<i>Y</i>	indexed single precision vector <i>Y</i> of at least size $M$ if not transposed or size $N$ otherwise
<i>incY</i>	<i>Y</i> vector stride (use every $incY$ 'th element)

## Author

Peter Ahrens

## Date

18 Jan 2016

**2.2.2.35** float idxdBLAS\_sissq ( const int *fold*, const int *N*, const float \* *X*, const int *incX*, const float *scaleY*, float\_indexed \* *Y* )

Add to scaled indexed single precision *Y* the scaled sum of squares of elements of single precision vector *X*.

Add to *Y* the scaled indexed sum of the squares of each element of *X*. The scaling of each square is performed using [idxd\\_sscales\(\)](#)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector

<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y

**Returns**

the new scaling factor of Y

**Author**

Peter Ahrens

**Date**

18 Jan 2016

**2.2.2.36** void idxdBLAS\_sisum ( const int *fold*, const int *N*, const float \* *X*, const int *incX*, float\_indexed \* *Y* )

Add to indexed single precision Y the sum of single precision vector X.

Add to Y the indexed sum of X.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	indexed scalar Y

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.37** void idxdBLAS\_smcasum ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, float \* *manY*, const int *incmanY*, float \* *carY*, const int *inccarY* )

Add to manually specified indexed single precision Y the absolute sum of complex single precision vector X.

Add to Y the indexed sum of magnitudes of elements of X.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)

<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.38** `float idxdBLAS_smcssq ( const int fold, const int N, const void * X, const int incX, const float scaleY, float * manY, const int incmanY, float * carY, const int inccarY )`

Add to scaled manually specified indexed single precision Y the scaled sum of squares of elements of complex single precision vector X.

Add to Y the scaled indexed sum of the squares of each element of X. The scaling of each square is performed using [idxd\\_sscales\(\)](#)

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Returns**

the new scaling factor of Y

**Author**

Peter Ahrens

**Date**

18 Jan 2016

**2.2.2.39** `void idxdBLAS_smsasum ( const int fold, const int N, const float * X, const int incX, float * manY, const int incmanY, float * carY, const int inccarY )`

Add to manually specified indexed single precision Y the absolute sum of double precision vector X.

Add to Y to the indexed sum of absolute values of elements in X.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.40** void idxdBLAS\_smsdot ( const int *fold*, const int *N*, const float \* *X*, const int *incX*, const float \* *Y*, const int *incY*, float \* *manZ*, const int *incmanZ*, float \* *carZ*, const int *inccarZ* )

Add to manually specified indexed single precision Z the dot product of single precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and Y.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every incY'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every incpriZ'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every inccarZ'th element)

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.41** float idxdBLAS\_smsssq ( const int *fold*, const int *N*, const float \* *X*, const int *incX*, const float *scaleY*, float \* *manY*, const int *incmanY*, float \* *carY*, const int *inccarY* )

Add to scaled manually specified indexed single precision Y the scaled sum of squares of elements of single precision vector X.

Add to Y the scaled indexed sum of the squares of each element of X. The scaling of each square is performed using [idxd\\_sscales\(\)](#)

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Returns

the new scaling factor of Y

## Author

Peter Ahrens

## Date

18 Jan 2016

**2.2.2.42** void idxdBLAS\_ssmsum ( const int *fold*, const int *N*, const float \* *X*, const int *incX*, float \* *manY*, const int *incmanY*, float \* *carY*, const int *inccarY* )

Add to manually specified indexed single precision Y the sum of single precision vector X.

Add to Y the indexed sum of X.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.43** void idxdBLAS\_zamax\_sub ( const int *N*, const void \* *X*, const int *incX*, void \* *amax* )

Find maximum magnitude in vector of complex double precision.

Returns the magnitude of the element of maximum magnitude in an array.

## Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>amax</i>	scalar return

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.44** void idxdBLAS\_zamaxm\_sub ( const int *N*, const void \* *X*, const int *incX*, const void \* *Y*, const int *incY*, void \* *amaxm* )

Find maximum magnitude pairwise product between vectors of complex double precision.

Returns the magnitude of the pairwise product of maximum magnitude between X and Y.

## Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>amaxm</i>	scalar return

## Author

Peter Ahrens

## Date

15 Jan 2016

**2.2.2.45** void idxdBLAS\_zizdotc ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, const void \* *Y*, const int *incY*, double\_complex\_indexed \* *Z* )

Add to indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and conjugated Y.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>scalar</i>	return <i>Z</i>

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.46** void idxdBLAS\_zizdotu ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, const void \* *Y*, const int *incY*, double\_complex\_indexed \* *Z* )

Add to indexed complex double precision *Z* the unconjugated dot product of complex double precision vectors *X* and *Y*.

Add to *Z* the indexed sum of the pairwise products of *X* and *Y*.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>indexed</i>	scalar <i>Z</i>

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.47** void idxdBLAS\_zizgemm ( const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void \* *alpha*, const void \* *A*, const int *lda*, const void \* *B*, const int *ldb*, double\_complex\_indexed \* *C*, const int *ldc* )

Add to indexed complex double precision matrix *C* the matrix-matrix product of complex double precision matrices *A* and *B*.

Performs one of the matrix-matrix operations

$$C := \alpha * \text{op}(A) * \text{op}(B) + C,$$

where *op*(*X*) is one of

$$\text{op}(X) = X \text{ or } \text{op}(X) = X^{**T} \text{ or } \text{op}(X) = X^{**H},$$

*alpha* is a scalar, *A* and *B* are matrices with *op*(*A*) an *M* by *K* matrix and *op*(*B*) a *K* by *N* matrix, and *C* is an indexed *M* by *N* matrix.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	indexed complex double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

## Author

Peter Ahrens

## Date

18 Jan 2016

```
2.2.2.48 void idxdBLAS_zizgemv ( const int fold, const char Order, const char TransA, const int M, const int N, const void *
    alpha, const void * A, const int lda, const void * X, const int incX, double_complex_indexed * Y, const int incY
    )
```

Add to indexed complex double precision vector Y the matrix-vector product of complex double precision matrix A and complex double precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$  or  $y := \alpha * A ** T * x + \beta * y$  or  $y := \alpha * A ** H * x + \beta * y$ ,

where alpha and beta are scalars, x is a vector, y is an indexed vector, and A is an M by N matrix.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A



<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	indexed complex double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

**Author**

Peter Ahrens

**Date**

18 Jan 2016

**2.2.2.49** void idxdBLAS\_zizsum ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, double\_complex\_indexed \* *Y* )

Add to indexed complex double precision Y the sum of complex double precision vector X.

Add to Y the indexed sum of X.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>indexed</i>	scalar Y

**Author**

Peter Ahrens

**Date**

15 Jan 2016

**2.2.2.50** void idxdBLAS\_zmzdotc ( const int *fold*, const int *N*, const void \* *X*, const int *incX*, const void \* *Y*, const int *incY*, double \* *manZ*, const int *incmanZ*, double \* *carZ*, const int *inccarZ* )

Add to manually specified indexed complex double precision Z the conjugated dot product of complex double precision vectors X and Y.

Add to Z the indexed sum of the pairwise products of X and conjugated Y.

**Parameters**

<i>fold</i>	the fold of the indexed types
-------------	-------------------------------

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

**Author**

Peter Ahrens

**Date**

15 Jan 2016

2.2.2.51 `void idxdBLAS_zmzdotu ( const int fold, const int N, const void * X, const int incX, const void * Y, const int incY, double * manZ, const int incmanZ, double * carZ, const int inccarZ )`

Add to manually specified indexed complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.

Add to Z to the indexed sum of the pairwise products of X and Y.

**Parameters**

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

**Author**

Peter Ahrens

**Date**

15 Jan 2016

2.2.2.52 `void idxdBLAS_zmzsum ( const int fold, const int N, const void * X, const int incX, double * manY, const int incmanY, double * carY, const int inccarY )`

Add to manually specified indexed complex double precision Y the sum of complex double precision vector X.

Add to Y the indexed sum of X.

## Parameters

<i>fold</i>	the fold of the indexed types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

## Author

Peter Ahrens

## Date

15 Jan 2016



# Index

## DIWIDTH

[idxd.h, 11](#)

## double\_complex\_indexed

[idxd.h, 14](#)

## double\_indexed

[idxd.h, 14](#)

## float\_complex\_indexed

[idxd.h, 15](#)

## float\_indexed

[idxd.h, 15](#)

## idxd.h

[DIWIDTH, 11](#)

[double\\_complex\\_indexed, 14](#)

[double\\_indexed, 14](#)

[float\\_complex\\_indexed, 15](#)

[float\\_indexed, 15](#)

[idxd\\_DICAPACITY, 11](#)

[idxd\\_DIENDURANCE, 11](#)

[idxd\\_DIMAXFOLD, 11](#)

[idxd\\_DIMAXINDEX, 12](#)

[idxd\\_DMCOMPRESSION, 12](#)

[idxd\\_DMEXPANSION, 12](#)

[idxd\\_SICAPACITY, 12](#)

[idxd\\_SIENDURANCE, 13](#)

[idxd\\_SIMAXFOLD, 13](#)

[idxd\\_SIMAXINDEX, 13](#)

[idxd\\_SMCOMPRESSION, 13](#)

[idxd\\_SMEXPANSION, 14](#)

[idxd\\_ccconv\\_sub, 15](#)

[idxd\\_ccmconv\\_sub, 15](#)

[idxd\\_cialloc, 17](#)

[idxd\\_cicadd, 17](#)

[idxd\\_cicconv, 17](#)

[idxd\\_cicdeposit, 19](#)

[idxd\\_ciciadd, 19](#)

[idxd\\_ciciaddv, 20](#)

[idxd\\_ciciset, 20](#)

[idxd\\_cicupdate, 20](#)

[idxd\\_cinegate, 21](#)

[idxd\\_cinum, 21](#)

[idxd\\_ciprint, 21](#)

[idxd\\_cirenorm, 23](#)

[idxd\\_cisetzero, 23](#)

[idxd\\_cisiset, 23](#)

[idxd\\_cisize, 24](#)

[idxd\\_cisupdate, 24](#)

[idxd\\_cmadd, 24](#)

[idxd\\_cmconv, 26](#)

[idxd\\_cmcddeposit, 26](#)

[idxd\\_cmcmadd, 27](#)

[idxd\\_cmcmset, 27](#)

[idxd\\_cmcupdate, 28](#)

[idxd\\_cmddenorm, 28](#)

[idxd\\_cmnegate, 29](#)

[idxd\\_cmprint, 29](#)

[idxd\\_cmrenorm, 29](#)

[idxd\\_cmsetzero, 30](#)

[idxd\\_cmsmset, 30](#)

[idxd\\_cmsrescale, 31](#)

[idxd\\_cmsupdate, 31](#)

[idxd\\_ddconv, 32](#)

[idxd\\_ddmconv, 32](#)

[idxd\\_dialloc, 32](#)

[idxd\\_dibound, 34](#)

[idxd\\_didadd, 34](#)

[idxd\\_didconv, 34](#)

[idxd\\_diddeposit, 36](#)

[idxd\\_didiadd, 36](#)

[idxd\\_didiaddsq, 37](#)

[idxd\\_didiaddv, 37](#)

[idxd\\_didiset, 38](#)

[idxd\\_didupdate, 38](#)

[idxd\\_dindex, 38](#)

[idxd\\_dinegate, 39](#)

[idxd\\_dinum, 39](#)

[idxd\\_diprint, 39](#)

[idxd\\_direnorm, 40](#)

[idxd\\_disetzero, 40](#)

[idxd\\_disize, 40](#)

[idxd\\_dmbins, 41](#)

[idxd\\_dmdadd, 41](#)

[idxd\\_dmdconv, 42](#)

[idxd\\_dmddeposit, 42](#)

[idxd\\_dmdddenorm, 43](#)

[idxd\\_dmdmadd, 43](#)

[idxd\\_dmdmaddsq, 44](#)

[idxd\\_dmdmset, 44](#)

[idxd\\_dmdrescale, 45](#)

[idxd\\_dmdupdate, 45](#)

[idxd\\_dmindex, 46](#)

[idxd\\_dmindex0, 46](#)

[idxd\\_dmnegate, 46](#)

[idxd\\_dmprint, 47](#)

[idxd\\_dmrenorm, 47](#)

[idxd\\_dmsetzero, 48](#)

[idxd\\_dscale, 48](#)

[idxd\\_sialloc, 48](#)

idxd\_sibound, 50  
 idxd\_sindex, 50  
 idxd\_sinegate, 51  
 idxd\_sinum, 51  
 idxd\_siprint, 51  
 idxd\_sirenorm, 52  
 idxd\_sisadd, 52  
 idxd\_sisconv, 52  
 idxd\_sisdeposit, 53  
 idxd\_sisetzzero, 53  
 idxd\_sisiadd, 54  
 idxd\_sisiaddsq, 54  
 idxd\_sisiadv, 54  
 idxd\_sisiset, 55  
 idxd\_sisize, 55  
 idxd\_sisupdate, 56  
 idxd\_smbins, 56  
 idxd\_smdenorm, 56  
 idxd\_smindex, 57  
 idxd\_smindex0, 57  
 idxd\_smnegate, 57  
 idxd\_smprint, 58  
 idxd\_smrenorm, 58  
 idxd\_smsadd, 59  
 idxd\_smsconv, 59  
 idxd\_smsdeposit, 59  
 idxd\_smsetzero, 60  
 idxd\_smsmadd, 60  
 idxd\_smsmaddsq, 61  
 idxd\_smsmset, 61  
 idxd\_smsrescale, 62  
 idxd\_smsupdate, 62  
 idxd\_ssacle, 63  
 idxd\_ssiconv, 63  
 idxd\_ssmconv, 64  
 idxd\_ufp, 64  
 idxd\_ufpf, 64  
 idxd\_zialloc, 65  
 idxd\_zidiset, 65  
 idxd\_zidupdate, 65  
 idxd\_zinegate, 67  
 idxd\_zinum, 67  
 idxd\_ziprint, 67  
 idxd\_zirenorm, 68  
 idxd\_zisetzzero, 68  
 idxd\_zisize, 68  
 idxd\_zizadd, 69  
 idxd\_zizconv, 69  
 idxd\_zizdeposit, 69  
 idxd\_ziziadd, 70  
 idxd\_ziziadv, 70  
 idxd\_ziziset, 71  
 idxd\_zizupdate, 71  
 idxd\_zmdenorm, 71  
 idxd\_zmdmset, 72  
 idxd\_zmdrescale, 72  
 idxd\_zmdupdate, 73  
 idxd\_zmnegate, 73  
 idxd\_zmprint, 74  
 idxd\_zmrenorm, 74  
 idxd\_zmsetzero, 74  
 idxd\_zmzadd, 75  
 idxd\_zmzconv, 75  
 idxd\_zmzdeposit, 76  
 idxd\_zmzmadd, 76  
 idxd\_zmzmset, 77  
 idxd\_zmzupdate, 77  
 idxd\_zziconv\_sub, 78  
 idxd\_zzmconv\_sub, 78  
 SIWIDTH, 14  
 idxd\_DICAPACITY  
     idxd.h, 11  
 idxd\_DIENDURANCE  
     idxd.h, 11  
 idxd\_DIMAXFOLD  
     idxd.h, 11  
 idxd\_DIMAXINDEX  
     idxd.h, 12  
 idxd\_DMCOMPRESSION  
     idxd.h, 12  
 idxd\_DMEXPANSION  
     idxd.h, 12  
 idxd\_SICAPACITY  
     idxd.h, 12  
 idxd\_SIENDURANCE  
     idxd.h, 13  
 idxd\_SIMAXFOLD  
     idxd.h, 13  
 idxd\_SIMAXINDEX  
     idxd.h, 13  
 idxd\_SMCOMPRESSION  
     idxd.h, 13  
 idxd\_SMEXPANSION  
     idxd.h, 14  
 idxd\_cciconv\_sub  
     idxd.h, 15  
 idxd\_ccmconv\_sub  
     idxd.h, 15  
 idxd\_cialloc  
     idxd.h, 17  
 idxd\_cicadd  
     idxd.h, 17  
 idxd\_cicconv  
     idxd.h, 17  
 idxd\_cicdeposit  
     idxd.h, 19  
 idxd\_ciciadd  
     idxd.h, 19  
 idxd\_ciciadv  
     idxd.h, 20  
 idxd\_ciciset  
     idxd.h, 20  
 idxd\_cicupdate  
     idxd.h, 20  
 idxd\_cinegate  
     idxd.h, 21

idxd\_cinum  
  idxd.h, [21](#)

idxd\_ciprint  
  idxd.h, [21](#)

idxd\_cirenorm  
  idxd.h, [23](#)

idxd\_cisetzero  
  idxd.h, [23](#)

idxd\_cisiset  
  idxd.h, [23](#)

idxd\_cisize  
  idxd.h, [24](#)

idxd\_cisupdate  
  idxd.h, [24](#)

idxd\_cmcadd  
  idxd.h, [24](#)

idxd\_cmconv  
  idxd.h, [26](#)

idxd\_cmdeposit  
  idxd.h, [26](#)

idxd\_cmcmadd  
  idxd.h, [27](#)

idxd\_cmcmset  
  idxd.h, [27](#)

idxd\_cmcupdate  
  idxd.h, [28](#)

idxd\_cmddenorm  
  idxd.h, [28](#)

idxd\_cmnegate  
  idxd.h, [29](#)

idxd\_cmprint  
  idxd.h, [29](#)

idxd\_cmrenorm  
  idxd.h, [29](#)

idxd\_cmsetzero  
  idxd.h, [30](#)

idxd\_cmsmset  
  idxd.h, [30](#)

idxd\_cmsrescale  
  idxd.h, [31](#)

idxd\_cmsupdate  
  idxd.h, [31](#)

idxd\_ddiconv  
  idxd.h, [32](#)

idxd\_ddmconv  
  idxd.h, [32](#)

idxd\_dialloc  
  idxd.h, [32](#)

idxd\_dibound  
  idxd.h, [34](#)

idxd\_didadd  
  idxd.h, [34](#)

idxd\_didconv  
  idxd.h, [34](#)

idxd\_diddeposit  
  idxd.h, [36](#)

idxd\_didiadd  
  idxd.h, [36](#)

idxd\_didiaddsq  
  idxd.h, [37](#)

idxd\_didiadv  
  idxd.h, [37](#)

idxd\_didiset  
  idxd.h, [38](#)

idxd\_didupdate  
  idxd.h, [38](#)

idxd\_dindex  
  idxd.h, [38](#)

idxd\_dinegate  
  idxd.h, [39](#)

idxd\_dinum  
  idxd.h, [39](#)

idxd\_diprint  
  idxd.h, [39](#)

idxd\_direnorm  
  idxd.h, [40](#)

idxd\_disetzero  
  idxd.h, [40](#)

idxd\_disize  
  idxd.h, [40](#)

idxd\_dmbins  
  idxd.h, [41](#)

idxd\_dmdadd  
  idxd.h, [41](#)

idxd\_dmdconv  
  idxd.h, [42](#)

idxd\_dmddeposit  
  idxd.h, [42](#)

idxd\_dmdenorm  
  idxd.h, [43](#)

idxd\_dmdmadd  
  idxd.h, [43](#)

idxd\_dmdmaddsq  
  idxd.h, [44](#)

idxd\_dmdmset  
  idxd.h, [44](#)

idxd\_dmdrescale  
  idxd.h, [45](#)

idxd\_dmdupdate  
  idxd.h, [45](#)

idxd\_dmindex  
  idxd.h, [46](#)

idxd\_dmindex0  
  idxd.h, [46](#)

idxd\_dmnegate  
  idxd.h, [46](#)

idxd\_dmprint  
  idxd.h, [47](#)

idxd\_dmrenorm  
  idxd.h, [47](#)

idxd\_dmsetzero  
  idxd.h, [48](#)

idxd\_dscale  
  idxd.h, [48](#)

idxd\_sialloc  
  idxd.h, [48](#)

idxd_sibound	idxd.h, <a href="#">50</a>	idxd_smsmset	idxd.h, <a href="#">61</a>
idxd_sindex	idxd.h, <a href="#">50</a>	idxd_smsrescale	idxd.h, <a href="#">62</a>
idxd_sinegate	idxd.h, <a href="#">51</a>	idxd_smsupdate	idxd.h, <a href="#">62</a>
idxd_sinum	idxd.h, <a href="#">51</a>	idxd_sscales	idxd.h, <a href="#">63</a>
idxd_siprint	idxd.h, <a href="#">51</a>	idxd_ssiconv	idxd.h, <a href="#">63</a>
idxd_sirenorm	idxd.h, <a href="#">52</a>	idxd_ssmconv	idxd.h, <a href="#">64</a>
idxd_sisadd	idxd.h, <a href="#">52</a>	idxd_uftp	idxd.h, <a href="#">64</a>
idxd_sisconv	idxd.h, <a href="#">52</a>	idxd_uftp	idxd.h, <a href="#">64</a>
idxd_sisdeposit	idxd.h, <a href="#">53</a>	idxd_zialloc	idxd.h, <a href="#">65</a>
idxd_sisetzero	idxd.h, <a href="#">53</a>	idxd_zidiset	idxd.h, <a href="#">65</a>
idxd_sisiadd	idxd.h, <a href="#">54</a>	idxd_zidupdate	idxd.h, <a href="#">65</a>
idxd_sisiaddsq	idxd.h, <a href="#">54</a>	idxd_zinegate	idxd.h, <a href="#">67</a>
idxd_sisiaddv	idxd.h, <a href="#">54</a>	idxd_zinum	idxd.h, <a href="#">67</a>
idxd_sisiset	idxd.h, <a href="#">55</a>	idxd_ziprint	idxd.h, <a href="#">67</a>
idxd_sisize	idxd.h, <a href="#">55</a>	idxd_zirenorm	idxd.h, <a href="#">68</a>
idxd_sisupdate	idxd.h, <a href="#">56</a>	idxd_zisetzero	idxd.h, <a href="#">68</a>
idxd_smbins	idxd.h, <a href="#">56</a>	idxd_zisize	idxd.h, <a href="#">68</a>
idxd_smdenorm	idxd.h, <a href="#">56</a>	idxd_zizadd	idxd.h, <a href="#">69</a>
idxd_smindex	idxd.h, <a href="#">57</a>	idxd_zizconv	idxd.h, <a href="#">69</a>
idxd_smindex0	idxd.h, <a href="#">57</a>	idxd_zizdeposit	idxd.h, <a href="#">69</a>
idxd_smnegate	idxd.h, <a href="#">57</a>	idxd_ziziadd	idxd.h, <a href="#">70</a>
idxd_smprint	idxd.h, <a href="#">58</a>	idxd_ziziaddv	idxd.h, <a href="#">70</a>
idxd_smrenorm	idxd.h, <a href="#">58</a>	idxd_ziziset	idxd.h, <a href="#">71</a>
idxd_smsadd	idxd.h, <a href="#">59</a>	idxd_zizupdate	idxd.h, <a href="#">71</a>
idxd_smsconv	idxd.h, <a href="#">59</a>	idxd_zmdenorm	idxd.h, <a href="#">71</a>
idxd_smsdeposit	idxd.h, <a href="#">59</a>	idxd_zmdmset	idxd.h, <a href="#">72</a>
idxd_smsetzero	idxd.h, <a href="#">60</a>	idxd_zmdrescale	idxd.h, <a href="#">72</a>
idxd_smsmadd	idxd.h, <a href="#">60</a>	idxd_zmdupdate	idxd.h, <a href="#">73</a>
idxd_smsmaddsq	idxd.h, <a href="#">61</a>	idxd_zmnegate	idxd.h, <a href="#">73</a>



idxd\_zmprint  
     idxd.h, 74  
 idxd\_zmrenorm  
     idxd.h, 74  
 idxd\_zmsetzero  
     idxd.h, 74  
 idxd\_zmzadd  
     idxd.h, 75  
 idxd\_zmzconv  
     idxd.h, 75  
 idxd\_zmzdeposit  
     idxd.h, 76  
 idxd\_zmzmadd  
     idxd.h, 76  
 idxd\_zmzmset  
     idxd.h, 77  
 idxd\_zmzupdate  
     idxd.h, 77  
 idxd\_zziconv\_sub  
     idxd.h, 78  
 idxd\_zzmconv\_sub  
     idxd.h, 78  
 idxdBLAS.h  
     idxdBLAS\_camax\_sub, 83  
     idxdBLAS\_camaxm\_sub, 83  
     idxdBLAS\_cicdotc, 83  
     idxdBLAS\_cicdotu, 85  
     idxdBLAS\_cicgemm, 85  
     idxdBLAS\_cicgemv, 86  
     idxdBLAS\_cicsum, 87  
     idxdBLAS\_cmcdotc, 87  
     idxdBLAS\_cmcdotu, 88  
     idxdBLAS\_cmcsu, 88  
     idxdBLAS\_damax, 89  
     idxdBLAS\_damaxm, 89  
     idxdBLAS\_didasu, 90  
     idxdBLAS\_diddot, 90  
     idxdBLAS\_didgemm, 91  
     idxdBLAS\_didgemv, 91  
     idxdBLAS\_didssq, 92  
     idxdBLAS\_didsum, 93  
     idxdBLAS\_dizasum, 93  
     idxdBLAS\_dizssq, 93  
     idxdBLAS\_dmdasum, 94  
     idxdBLAS\_dmddot, 94  
     idxdBLAS\_dmdssq, 95  
     idxdBLAS\_dmdsum, 95  
     idxdBLAS\_dmzasum, 96  
     idxdBLAS\_dmzssq, 96  
     idxdBLAS\_samax, 97  
     idxdBLAS\_samaxm, 97  
     idxdBLAS\_sicasum, 98  
     idxdBLAS\_sicssq, 98  
     idxdBLAS\_sisasum, 99  
     idxdBLAS\_sisdot, 99  
     idxdBLAS\_sisgemm, 100  
     idxdBLAS\_sisgemv, 101  
     idxdBLAS\_sissq, 101  
     idxdBLAS\_sissu, 102  
     idxdBLAS\_smcasum, 102  
     idxdBLAS\_smcssq, 103  
     idxdBLAS\_smsasum, 103  
     idxdBLAS\_smsdot, 104  
     idxdBLAS\_smssq, 104  
     idxdBLAS\_smssu, 105  
     idxdBLAS\_zamax\_sub, 105  
     idxdBLAS\_zamaxm\_sub, 106  
     idxdBLAS\_zizdotc, 106  
     idxdBLAS\_zizdotu, 107  
     idxdBLAS\_zizgemm, 107  
     idxdBLAS\_zizgemv, 108  
     idxdBLAS\_zizsum, 109  
     idxdBLAS\_zmzdotc, 109  
     idxdBLAS\_zmzdotu, 110  
     idxdBLAS\_zmzsum, 110  
 idxdBLAS\_camax\_sub  
     idxdBLAS.h, 83  
 idxdBLAS\_camaxm\_sub  
     idxdBLAS.h, 83  
 idxdBLAS\_cicdotc  
     idxdBLAS.h, 83  
 idxdBLAS\_cicdotu  
     idxdBLAS.h, 85  
 idxdBLAS\_cicgemm  
     idxdBLAS.h, 85  
 idxdBLAS\_cicgemv  
     idxdBLAS.h, 86  
 idxdBLAS\_cicsum  
     idxdBLAS.h, 87  
 idxdBLAS\_cmcdotc  
     idxdBLAS.h, 87  
 idxdBLAS\_cmcdotu  
     idxdBLAS.h, 88  
 idxdBLAS\_cmcsu  
     idxdBLAS.h, 88  
 idxdBLAS\_damax  
     idxdBLAS.h, 89  
 idxdBLAS\_damaxm  
     idxdBLAS.h, 89  
 idxdBLAS\_didasu  
     idxdBLAS.h, 90  
 idxdBLAS\_diddot  
     idxdBLAS.h, 90  
 idxdBLAS\_didgemm  
     idxdBLAS.h, 91  
 idxdBLAS\_didgemv  
     idxdBLAS.h, 91  
 idxdBLAS\_didssq  
     idxdBLAS.h, 92  
 idxdBLAS\_didsum  
     idxdBLAS.h, 93  
 idxdBLAS\_dizasum  
     idxdBLAS.h, 93  
 idxdBLAS\_dizssq  
     idxdBLAS.h, 93  
 idxdBLAS\_dmdasum

- idxdBLAS.h, [94](#)
- idxdBLAS\_dmdot
  - idxdBLAS.h, [94](#)
- idxdBLAS\_dmdssq
  - idxdBLAS.h, [95](#)
- idxdBLAS\_dmdsum
  - idxdBLAS.h, [95](#)
- idxdBLAS\_dmzasum
  - idxdBLAS.h, [96](#)
- idxdBLAS\_dmzssq
  - idxdBLAS.h, [96](#)
- idxdBLAS\_samax
  - idxdBLAS.h, [97](#)
- idxdBLAS\_samaxm
  - idxdBLAS.h, [97](#)
- idxdBLAS\_sicasum
  - idxdBLAS.h, [98](#)
- idxdBLAS\_sicssq
  - idxdBLAS.h, [98](#)
- idxdBLAS\_sisasum
  - idxdBLAS.h, [99](#)
- idxdBLAS\_sisdot
  - idxdBLAS.h, [99](#)
- idxdBLAS\_sisgemm
  - idxdBLAS.h, [100](#)
- idxdBLAS\_sisgemv
  - idxdBLAS.h, [101](#)
- idxdBLAS\_sisssq
  - idxdBLAS.h, [101](#)
- idxdBLAS\_sissum
  - idxdBLAS.h, [102](#)
- idxdBLAS\_smcasum
  - idxdBLAS.h, [102](#)
- idxdBLAS\_smcssq
  - idxdBLAS.h, [103](#)
- idxdBLAS\_smsasum
  - idxdBLAS.h, [103](#)
- idxdBLAS\_smsdot
  - idxdBLAS.h, [104](#)
- idxdBLAS\_smsssq
  - idxdBLAS.h, [104](#)
- idxdBLAS\_smssum
  - idxdBLAS.h, [105](#)
- idxdBLAS\_zamax\_sub
  - idxdBLAS.h, [105](#)
- idxdBLAS\_zamaxm\_sub
  - idxdBLAS.h, [106](#)
- idxdBLAS\_zizdotc
  - idxdBLAS.h, [106](#)
- idxdBLAS\_zizdotu
  - idxdBLAS.h, [107](#)
- idxdBLAS\_zizgemm
  - idxdBLAS.h, [107](#)
- idxdBLAS\_zizgemv
  - idxdBLAS.h, [108](#)
- idxdBLAS\_zizsum
  - idxdBLAS.h, [109](#)
- idxdBLAS\_zmzdotc
  - idxdBLAS.h, [109](#)
- idxdBLAS\_zmzdotu
  - idxdBLAS.h, [110](#)
- idxdBLAS\_zmzsum
  - idxdBLAS.h, [110](#)
- include/idxd.h, [3](#)
- include/idxdBLAS.h, [79](#)
- SIWIDTH
  - idxd.h, [14](#)