

Signal/Geometry Processing Library (SPL)
2.0.6

Generated by Doxygen 1.8.9.1

Fri Sep 15 2017 08:19:45

Contents

1	Signal/Geometry Processing Library (SPL) Reference Manual (Version 2.0.6)	1
1.1	Introduction	1
1.2	License	1
1.3	Reporting Bugs	1
2	Getting Started	3
2.1	Installation	3
2.2	Organization of the Manual	5
3	Frequently Asked Questions (FAQ)	7
4	Known Bugs	9
5	Module Index	11
5.1	Modules	11
6	Hierarchical Index	13
6.1	Class Hierarchy	13
7	Class Index	15
7.1	Class List	15
8	File Index	17
8.1	File List	17
9	Module Documentation	19
9.1	Arrays and Sequences	19
9.1.1	Detailed Description	19
9.2	One-Dimensional Arrays	20
9.2.1	Detailed Description	22
9.2.2	Typedef Documentation	22
9.2.2.1	IntArray1	22

9.2.2.2	RealArray1	22
9.2.3	Function Documentation	22
9.2.3.1	Array1	22
9.2.3.2	Array1	22
9.2.3.3	Array1	22
9.2.3.4	Array1	22
9.2.3.5	begin	22
9.2.3.6	begin	23
9.2.3.7	dump	23
9.2.3.8	end	23
9.2.3.9	end	23
9.2.3.10	fill	23
9.2.3.11	getSize	23
9.2.3.12	isShared	23
9.2.3.13	isSharedWith	23
9.2.3.14	load	23
9.2.3.15	max	23
9.2.3.16	min	24
9.2.3.17	operator"!="	24
9.2.3.18	operator()	24
9.2.3.19	operator()	24
9.2.3.20	operator*="	24
9.2.3.21	operator*="	24
9.2.3.22	operator+="	24
9.2.3.23	operator+="	24
9.2.3.24	operator-="	24
9.2.3.25	operator-="	24
9.2.3.26	operator/="	25
9.2.3.27	operator/="	25
9.2.3.28	operator<<	25
9.2.3.29	operator="	25
9.2.3.30	operator="	25
9.2.3.31	operator=="	25
9.2.3.32	operator>>	25
9.2.3.33	output	25
9.2.3.34	resize	25
9.2.3.35	resize	25

9.2.3.36	save	26
9.2.3.37	sum	26
9.2.3.38	swap	26
9.2.3.39	~Array1	26
9.3	Two-Dimensional Arrays	27
9.3.1	Detailed Description	29
9.3.2	Typedef Documentation	29
9.3.2.1	IntArray2	29
9.3.2.2	RealArray2	30
9.3.3	Function Documentation	30
9.3.3.1	Array2	30
9.3.3.2	Array2	30
9.3.3.3	Array2	30
9.3.3.4	Array2	30
9.3.3.5	begin	30
9.3.3.6	begin	30
9.3.3.7	colBegin	30
9.3.3.8	colBegin	30
9.3.3.9	colEnd	30
9.3.3.10	colEnd	30
9.3.3.11	dump	31
9.3.3.12	end	31
9.3.3.13	end	31
9.3.3.14	fill	31
9.3.3.15	fliplr	31
9.3.3.16	flipud	31
9.3.3.17	getHeight	31
9.3.3.18	getSize	31
9.3.3.19	getWidth	31
9.3.3.20	isShared	31
9.3.3.21	isSharedWith	32
9.3.3.22	load	32
9.3.3.23	max	32
9.3.3.24	min	32
9.3.3.25	operator"!="	32
9.3.3.26	operator()	32
9.3.3.27	operator()	32

9.3.3.28	operator()	32
9.3.3.29	operator()	32
9.3.3.30	operator*= operator*=	32
9.3.3.31	operator*=	33
9.3.3.32	operator+=	33
9.3.3.33	operator+=	33
9.3.3.34	operator-=	33
9.3.3.35	operator-=	33
9.3.3.36	operator/=	33
9.3.3.37	operator/=	33
9.3.3.38	operator<<	33
9.3.3.39	operator=	33
9.3.3.40	operator==	33
9.3.3.41	operator>>	33
9.3.3.42	output	34
9.3.3.43	resize	34
9.3.3.44	resize	34
9.3.3.45	rowBegin	34
9.3.3.46	rowBegin	34
9.3.3.47	rowEnd	34
9.3.3.48	rowEnd	34
9.3.3.49	save	34
9.3.3.50	sum	34
9.3.3.51	swap	34
9.3.3.52	transpose	35
9.3.3.53	unshare	35
9.3.3.54	~Array2	35
9.4	One-Dimensional Sequences	36
9.4.1	Detailed Description	39
9.4.2	Typedef Documentation	39
9.4.2.1	IntSequence1	39
9.4.2.2	RealSequence1	39
9.4.3	Function Documentation	39
9.4.3.1	add	39
9.4.3.2	approxEqual	39
9.4.3.3	begin	40
9.4.3.4	begin	40

9.4.3.5	convolve	40
9.4.3.6	downsample	40
9.4.3.7	end	40
9.4.3.8	end	40
9.4.3.9	fill	40
9.4.3.10	getArray	41
9.4.3.11	getEndInd	41
9.4.3.12	getSize	41
9.4.3.13	getStartInd	41
9.4.3.14	isShared	41
9.4.3.15	max	41
9.4.3.16	min	41
9.4.3.17	operator"!=	41
9.4.3.18	operator()	41
9.4.3.19	operator()	42
9.4.3.20	operator*	42
9.4.3.21	operator*	42
9.4.3.22	operator*	42
9.4.3.23	operator*==	42
9.4.3.24	operator*==	42
9.4.3.25	operator+	42
9.4.3.26	operator+	42
9.4.3.27	operator+	43
9.4.3.28	operator+=	43
9.4.3.29	operator+=	43
9.4.3.30	operator-	43
9.4.3.31	operator-	43
9.4.3.32	operator-=	43
9.4.3.33	operator-=	43
9.4.3.34	operator/	44
9.4.3.35	operator/	44
9.4.3.36	operator/=	44
9.4.3.37	operator/=	44
9.4.3.38	operator<<	44
9.4.3.39	operator=	44
9.4.3.40	operator==	44
9.4.3.41	operator>>	45

9.4.3.42	polyphaseJoin	45
9.4.3.43	polyphaseSplit	45
9.4.3.44	Sequence1	45
9.4.3.45	Sequence1	45
9.4.3.46	Sequence1	45
9.4.3.47	Sequence1	45
9.4.3.48	Sequence1	46
9.4.3.49	Sequence1	46
9.4.3.50	subsequence	46
9.4.3.51	sum	46
9.4.3.52	swapArray	46
9.4.3.53	translate	46
9.4.3.54	translate	46
9.4.3.55	upsample	46
9.4.3.56	~Sequence1	47
9.5	Two-Dimensional Sequences	48
9.5.1	Detailed Description	52
9.5.2	Typedef Documentation	52
9.5.2.1	IntSequence2	52
9.5.2.2	RealSequence2	52
9.5.3	Function Documentation	52
9.5.3.1	add	52
9.5.3.2	approxEqual	52
9.5.3.3	begin	52
9.5.3.4	begin	52
9.5.3.5	colBegin	52
9.5.3.6	colBegin	52
9.5.3.7	colEnd	53
9.5.3.8	colEnd	53
9.5.3.9	convolve	53
9.5.3.10	convolveSeparable	53
9.5.3.11	downsample	53
9.5.3.12	end	53
9.5.3.13	end	53
9.5.3.14	fill	53
9.5.3.15	getArray	53
9.5.3.16	getEndX	53

9.5.3.17	getEndY	54
9.5.3.18	getHeight	54
9.5.3.19	getSize	54
9.5.3.20	getStartX	54
9.5.3.21	getStartY	54
9.5.3.22	getWidth	54
9.5.3.23	isShared	54
9.5.3.24	max	54
9.5.3.25	min	54
9.5.3.26	operator"!="	54
9.5.3.27	operator()	54
9.5.3.28	operator()	55
9.5.3.29	operator*	55
9.5.3.30	operator*	55
9.5.3.31	operator*	55
9.5.3.32	operator*="	55
9.5.3.33	operator*="	55
9.5.3.34	operator+	55
9.5.3.35	operator+	55
9.5.3.36	operator+	56
9.5.3.37	operator+="	56
9.5.3.38	operator+="	56
9.5.3.39	operator-	56
9.5.3.40	operator-	56
9.5.3.41	operator-="	56
9.5.3.42	operator-="	56
9.5.3.43	operator/	56
9.5.3.44	operator/	57
9.5.3.45	operator/=	57
9.5.3.46	operator/=	57
9.5.3.47	operator<<	57
9.5.3.48	operator=	57
9.5.3.49	operator=="	57
9.5.3.50	operator>>	57
9.5.3.51	output	57
9.5.3.52	polyphaseJoin	58
9.5.3.53	polyphaseSplit	58

9.5.3.54	rowBegin	58
9.5.3.55	rowBegin	58
9.5.3.56	rowEnd	58
9.5.3.57	rowEnd	58
9.5.3.58	Sequence2	58
9.5.3.59	Sequence2	58
9.5.3.60	Sequence2	59
9.5.3.61	Sequence2	59
9.5.3.62	Sequence2	59
9.5.3.63	Sequence2	59
9.5.3.64	subsequence	59
9.5.3.65	sum	59
9.5.3.66	swapArray	59
9.5.3.67	translate	59
9.5.3.68	translate	59
9.5.3.69	upsample	60
9.5.3.70	upsample	60
9.5.3.71	~Sequence2	60
9.6	Bit Stream I/O	61
9.6.1	Detailed Description	62
9.6.2	Function Documentation	62
9.6.2.1	clearIoStateBits	62
9.6.2.2	clearReadCount	62
9.6.2.3	clearWriteCount	62
9.6.2.4	getIoState	62
9.6.2.5	getReadCount	62
9.6.2.6	getReadLimit	62
9.6.2.7	getWriteCount	62
9.6.2.8	getWriteLimit	62
9.6.2.9	isEof	63
9.6.2.10	isLimit	63
9.6.2.11	isOkay	63
9.6.2.12	setIoState	63
9.6.2.13	setIoStateBits	63
9.6.2.14	setReadLimit	63
9.6.2.15	setWriteLimit	63
9.7	Audio and Image Codecs	64

9.7.1	Detailed Description	64
9.8	Audio Codecs	65
9.8.1	Detailed Description	65
9.8.2	Function Documentation	65
9.8.2.1	loadAudioFile	65
9.8.2.2	saveAudioFile	65
9.9	Image Codecs	66
9.9.1	Detailed Description	66
9.9.2	Function Documentation	66
9.9.2.1	decodePbm	66
9.9.2.2	decodePgm	67
9.9.2.3	decodePnm	67
9.9.2.4	decodePpm	67
9.9.2.5	encodePbm	67
9.9.2.6	encodePgm	67
9.9.2.7	encodePnm	68
9.9.2.8	encodePpm	68
9.10	Filter Design	69
9.10.1	Detailed Description	69
9.10.2	Function Documentation	69
9.10.2.1	bandpassFilter	69
9.10.2.2	highpassFilter	69
9.10.2.3	lowpassFilter	70
9.11	CPU and Memory Utilization	71
9.11.1	Detailed Description	71
9.11.2	Function Documentation	71
9.11.2.1	getCurrentMemUsage	71
9.11.2.2	getPeakMemUsage	71
9.12	Math Utilities	72
9.12.1	Detailed Description	72
9.12.2	Function Documentation	72
9.12.2.1	absVal	72
9.12.2.2	ceilDiv	73
9.12.2.3	clip	73
9.12.2.4	degToRad	73
9.12.2.5	floorDiv	73
9.12.2.6	mod	73

9.12.2.7	radToDeg	73
9.12.2.8	roundTowardZeroDiv	73
9.12.2.9	signum	73
9.12.2.10	sinc	73
9.12.2.11	sqr	74
9.13	CGAL Utilities	75
9.13.1	Detailed Description	76
9.13.2	Function Documentation	76
9.13.2.1	angleBetweenVectors	76
9.13.2.2	Arcball	76
9.13.2.3	clear	76
9.13.2.4	closestPointOnRay	76
9.13.2.5	combineRotations	76
9.13.2.6	findRayPlaneIntersection	77
9.13.2.7	findRaySphereIntersection	77
9.13.2.8	getRotation	77
9.13.2.9	initialize	77
9.13.2.10	move	77
9.13.2.11	norm	77
9.13.2.12	normalize	77
9.13.2.13	operator*	77
9.13.2.14	operator/	78
9.13.2.15	quaternionToRotation	78
9.13.2.16	rotationToQuaternion	78
9.13.2.17	setMode	78
9.13.2.18	start	78
9.14	Arithmetic Coders	79
9.14.1	Detailed Description	79
9.15	M-Coder	80
9.15.1	Detailed Description	80
9.15.2	Function Documentation	80
9.15.2.1	getBitCount	80
9.15.2.2	getBitCount	80
9.15.2.3	getInput	81
9.15.2.4	getNumContexts	81
9.15.2.5	getNumContexts	81
9.15.2.6	getOutput	81

9.15.2.7	getSymCount	81
9.15.2.8	getSymCount	81
9.15.2.9	setInput	81
9.15.2.10	setOutput	81
9.16	Binary and m-ary Arithmetic Coders	82
9.16.1	Detailed Description	83
9.16.2	Function Documentation	83
9.16.2.1	getBitCount	83
9.16.2.2	getBitCount	84
9.16.2.3	getBitCount	84
9.16.2.4	getBitCount	84
9.16.2.5	getDebugStream	84
9.16.2.6	getDebugStream	84
9.16.2.7	getDebugStream	84
9.16.2.8	getDebugStream	84
9.16.2.9	getInput	84
9.16.2.10	getInput	84
9.16.2.11	getMaxContexts	85
9.16.2.12	getMaxContexts	85
9.16.2.13	getNumContexts	85
9.16.2.14	getNumContexts	85
9.16.2.15	getOutput	85
9.16.2.16	getOutput	85
9.16.2.17	getSymCount	85
9.16.2.18	getSymCount	86
9.16.2.19	getSymCount	86
9.16.2.20	getSymCount	86
9.16.2.21	setDebugLevel	86
9.16.2.22	setDebugLevel	86
9.16.2.23	setDebugLevel	86
9.16.2.24	setDebugLevel	86
9.16.2.25	setDebugStream	86
9.16.2.26	setDebugStream	86
9.16.2.27	setDebugStream	87
9.16.2.28	setDebugStream	87
9.16.2.29	setInput	87
9.16.2.30	setInput	87

9.16.2.31	setOutput	87
9.16.2.32	setOutput	87
10	Class Documentation	89
10.1	SPL::Arcball< T > Class Template Reference	89
10.1.1	Detailed Description	90
10.1.2	Member Typedef Documentation	90
10.1.2.1	Kernel	90
10.1.2.2	Point	90
10.1.2.3	Rotation	90
10.1.2.4	Vector	90
10.1.3	Member Function Documentation	90
10.1.3.1	setDebugLevel	90
10.2	SPL::Array1< T > Class Template Reference	91
10.2.1	Detailed Description	93
10.2.2	Member Typedef Documentation	93
10.2.2.1	ConstIterator	93
10.2.2.2	ElemType	93
10.2.2.3	Iterator	93
10.2.3	Constructor & Destructor Documentation	93
10.2.3.1	Array1	93
10.2.3.2	Array1	93
10.3	SPL::Array2< T > Class Template Reference	93
10.3.1	Detailed Description	96
10.3.2	Member Typedef Documentation	96
10.3.2.1	ConstIterator	96
10.3.2.2	ConstXIterator	96
10.3.2.3	ConstYIterator	96
10.3.2.4	ElemType	97
10.3.2.5	Iterator	97
10.3.2.6	XIterator	97
10.3.2.7	YIterator	97
10.3.3	Constructor & Destructor Documentation	97
10.3.3.1	Array2	97
10.3.3.2	Array2	97
10.3.4	Member Function Documentation	97
10.3.4.1	operator=	97

10.4	SPL::BinArithCoderContextStat Struct Reference	97
10.4.1	Detailed Description	97
10.5	SPL::BinArithDecoder Class Reference	98
10.5.1	Detailed Description	99
10.5.2	Constructor & Destructor Documentation	99
10.5.2.1	BinArithDecoder	99
10.5.2.2	~BinArithDecoder	99
10.5.3	Member Function Documentation	99
10.5.3.1	decodeBypass	99
10.5.3.2	decodeRegular	99
10.5.3.3	dump	99
10.5.3.4	getContextState	100
10.5.3.5	setContextState	100
10.5.3.6	start	100
10.5.3.7	terminate	100
10.6	SPL::BinArithEncoder Class Reference	100
10.6.1	Detailed Description	101
10.6.2	Constructor & Destructor Documentation	102
10.6.2.1	BinArithEncoder	102
10.6.2.2	~BinArithEncoder	102
10.6.3	Member Function Documentation	102
10.6.3.1	dump	102
10.6.3.2	dumpModels	102
10.6.3.3	encodeBypass	102
10.6.3.4	encodeRegular	102
10.6.3.5	getContextState	103
10.6.3.6	setContextState	103
10.6.3.7	start	103
10.6.3.8	terminate	103
10.7	SPL::BitStream Class Reference	104
10.7.1	Detailed Description	105
10.7.2	Member Typedef Documentation	105
10.7.2.1	IoState	105
10.7.2.2	Offset	105
10.7.2.3	Size	105
10.7.3	Member Data Documentation	105
10.7.3.1	allIoBits	105

10.7.3.2	badBit	105
10.7.3.3	eofBit	105
10.7.3.4	limitBit	106
10.8	SPL::ConvolveMode Struct Reference	106
10.8.1	Detailed Description	106
10.8.2	Member Data Documentation	106
10.8.2.1	full	106
10.8.2.2	sameDomainConstExt	106
10.8.2.3	sameDomainPerExt	106
10.8.2.4	sameDomainSymExt0	106
10.8.2.5	sameDomainZeroExt	107
10.9	SPL::InputBitStream Class Reference	107
10.9.1	Detailed Description	108
10.9.2	Member Typedef Documentation	108
10.9.2.1	loState	108
10.9.2.2	Offset	109
10.9.2.3	Size	109
10.9.3	Constructor & Destructor Documentation	109
10.9.3.1	InputBitStream	109
10.9.3.2	InputBitStream	109
10.9.3.3	~InputBitStream	109
10.9.4	Member Function Documentation	109
10.9.4.1	align	109
10.9.4.2	dump	109
10.9.4.3	getBits	109
10.9.4.4	getInput	109
10.9.4.5	setInput	109
10.9.5	Member Data Documentation	110
10.9.5.1	alloBits	110
10.9.5.2	badBit	110
10.9.5.3	eofBit	110
10.9.5.4	limitBit	110
10.10	SPL::MDecoder Class Reference	110
10.10.1	Detailed Description	111
10.10.2	Constructor & Destructor Documentation	111
10.10.2.1	MDecoder	111
10.10.2.2	~MDecoder	111

10.10.3 Member Function Documentation	111
10.10.3.1 clearContexts	111
10.10.3.2 decodeBypass	111
10.10.3.3 decodeRegular	112
10.10.3.4 dump	112
10.10.3.5 getDebugStream	112
10.10.3.6 setDebugLevel	112
10.10.3.7 setDebugStream	112
10.10.3.8 setNumContexts	112
10.10.3.9 start	112
10.10.3.10 terminate	112
10.11 SPL::MEncoder Class Reference	112
10.11.1 Detailed Description	113
10.11.2 Constructor & Destructor Documentation	114
10.11.2.1 MEncoder	114
10.11.2.2 ~MEncoder	114
10.11.3 Member Function Documentation	114
10.11.3.1 clearContexts	114
10.11.3.2 dump	114
10.11.3.3 encodeBypass	114
10.11.3.4 encodeRegular	114
10.11.3.5 getDebugStream	114
10.11.3.6 setDebugLevel	114
10.11.3.7 setDebugStream	114
10.11.3.8 setNumContexts	114
10.11.3.9 start	115
10.11.3.10 terminate	115
10.12 SPL::MultiArithDecoder Class Reference	115
10.12.1 Detailed Description	116
10.12.2 Constructor & Destructor Documentation	116
10.12.2.1 MultiArithDecoder	116
10.12.2.2 ~MultiArithDecoder	116
10.12.3 Member Function Documentation	116
10.12.3.1 decodeBypass	116
10.12.3.2 decodeRegular	116
10.12.3.3 dump	116
10.12.3.4 setContext	116

10.12.3.5 setContext	116
10.12.3.6 start	117
10.12.3.7 terminate	117
10.13SPL::MultiArithEncoder Class Reference	117
10.13.1 Detailed Description	118
10.13.2 Constructor & Destructor Documentation	118
10.13.2.1 MultiArithEncoder	118
10.13.2.2 ~MultiArithEncoder	118
10.13.3 Member Function Documentation	118
10.13.3.1 dump	118
10.13.3.2 encodeBypass	118
10.13.3.3 encodeRegular	118
10.13.3.4 setContext	118
10.13.3.5 setContext	119
10.13.3.6 start	119
10.13.3.7 terminate	119
10.14SPL::OutputBitStream Class Reference	119
10.14.1 Detailed Description	121
10.14.2 Member Typedef Documentation	121
10.14.2.1 IoState	121
10.14.2.2 Offset	121
10.14.2.3 Size	121
10.14.3 Constructor & Destructor Documentation	121
10.14.3.1 OutputBitStream	121
10.14.3.2 OutputBitStream	121
10.14.3.3 ~OutputBitStream	121
10.14.4 Member Function Documentation	121
10.14.4.1 align	121
10.14.4.2 dump	121
10.14.4.3 flush	121
10.14.4.4 getOutput	122
10.14.4.5 putBits	122
10.14.4.6 setOutput	122
10.14.5 Member Data Documentation	122
10.14.5.1 alloBits	122
10.14.5.2 badBit	122
10.14.5.3 eofBit	122

10.14.5.4 limitBit	122
10.15SPL::PnmHeader Struct Reference	122
10.15.1 Detailed Description	123
10.15.2 Member Data Documentation	123
10.15.2.1 height	123
10.15.2.2 magic	123
10.15.2.3 maxVal	123
10.15.2.4 sgnd	123
10.15.2.5 width	123
10.16SPL::Quaternion< T > Struct Template Reference	123
10.16.1 Detailed Description	124
10.16.2 Member Typedef Documentation	124
10.16.2.1 Real	124
10.16.2.2 Vector_3	124
10.16.3 Constructor & Destructor Documentation	124
10.16.3.1 Quaternion	124
10.16.3.2 Quaternion	124
10.16.4 Member Data Documentation	125
10.16.4.1 scalar	125
10.16.4.2 vector	125
10.17SPL::Rotation_3< T > Struct Template Reference	125
10.17.1 Detailed Description	125
10.17.2 Member Typedef Documentation	126
10.17.2.1 Real	126
10.17.2.2 Vector_3	126
10.17.3 Constructor & Destructor Documentation	126
10.17.3.1 Rotation_3	126
10.17.4 Member Data Documentation	126
10.17.4.1 angle	126
10.17.4.2 axis	126
10.18SPL::Sequence1< T > Class Template Reference	126
10.18.1 Detailed Description	128
10.18.2 Member Typedef Documentation	128
10.18.2.1 ConstIterator	128
10.18.2.2 ElemType	128
10.18.2.3 Iterator	129
10.18.3 Constructor & Destructor Documentation	129

10.18.3.1 Sequence1	129
10.18.3.2 Sequence1	129
10.18.4 Member Function Documentation	129
10.18.4.1 operator=	129
10.19SPL::Sequence2< T > Class Template Reference	129
10.19.1 Detailed Description	132
10.19.2 Member Typedef Documentation	132
10.19.2.1 ConstIterator	132
10.19.2.2 ConstXIterator	132
10.19.2.3 ConstYIterator	132
10.19.2.4 ElemType	132
10.19.2.5 Iterator	132
10.19.2.6 XIterator	133
10.19.2.7 YIterator	133
10.19.3 Constructor & Destructor Documentation	133
10.19.3.1 Sequence2	133
10.19.3.2 Sequence2	133
10.19.4 Member Function Documentation	133
10.19.4.1 operator=	133
10.20SPL::Timer Class Reference	133
10.20.1 Detailed Description	134
10.20.2 Member Function Documentation	134
10.20.2.1 get	134
10.20.2.2 start	134
10.20.2.3 stop	134
11 File Documentation	135
11.1 Arcball.hpp File Reference	135
11.1.1 Detailed Description	135
11.2 Array1.hpp File Reference	136
11.2.1 Detailed Description	137
11.2.2 Macro Definition Documentation	137
11.2.2.1 SPL_ARRAY1_INLINE	137
11.3 Array2.hpp File Reference	137
11.3.1 Detailed Description	139
11.3.2 Macro Definition Documentation	139
11.3.2.1 SPL_ARRAY2_INLINE	139

11.4 audioFile.hpp File Reference	139
11.4.1 Detailed Description	139
11.5 bitStream.hpp File Reference	139
11.5.1 Detailed Description	140
11.6 cgalUtil.hpp File Reference	140
11.6.1 Detailed Description	141
11.7 filterDesign.hpp File Reference	141
11.7.1 Detailed Description	141
11.8 math.hpp File Reference	141
11.8.1 Detailed Description	142
11.9 mCoder.hpp File Reference	142
11.9.1 Detailed Description	143
11.10misc.hpp File Reference	143
11.10.1 Detailed Description	143
11.11pnmCodec.cpp File Reference	143
11.11.1 Detailed Description	144
11.12pnmCodec.hpp File Reference	144
11.12.1 Detailed Description	145
11.13Sequence.hpp File Reference	146
11.13.1 Detailed Description	146
11.14Sequence1.hpp File Reference	146
11.14.1 Detailed Description	148
11.14.2 Macro Definition Documentation	148
11.14.2.1 SPL_SEQUENCE1_DEBUG	148
11.14.2.2 SPL_SEQUENCE1_INLINE	148
11.14.2.3 SPL_SEQUENCE1_USE_NEW_CONV	148
11.15Sequence2.cpp File Reference	149
11.15.1 Detailed Description	149
11.16Sequence2.hpp File Reference	149
11.16.1 Detailed Description	151
11.16.2 Macro Definition Documentation	152
11.16.2.1 SPL_SEQUENCE2_INLINE	152
11.16.2.2 SPL_SEQUENCE2_USE_NEW_CONV	152
11.17Timer.cpp File Reference	152
11.17.1 Detailed Description	152
11.18Timer.hpp File Reference	152
11.18.1 Detailed Description	153

Chapter 1

Signal/Geometry Processing Library (SPL) Reference Manual (Version 2.0.6)

1.1 Introduction

The Signal Processing Library (SPL) provides several classes and associated code that are useful for various signal/geometry processing applications. The SPL was developed by [Michael Adams](#) from the Department of Electrical and Computer Engineering at the University of Victoria, Victoria, BC, Canada.

1.2 License

Copyright (c) 2015 Michael D. Adams
All rights reserved.

This file is part of the Signal Processing Library (SPL).

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; see the file LICENSE. If not, see <http://www.gnu.org/licenses/>.

1.3 Reporting Bugs

Please send any bug reports to mdadams@ece.uvic.ca.

Chapter 2

Getting Started

The following sections are useful for getting started with the SPL:

- [Installation](#). Describes how to install the SPL.
- [Organization of the Manual](#). Briefly explains the organization of the manual (e.g., where to find things).

2.1 Installation

Installing the SPL Software

=====

The SPL software should build on most Linux systems with a C++ compiler that supports C++14.

Installation Steps

In what follows, `$SOURCE_DIR` denotes the top-level directory of the SPL software distribution (i.e., the directory containing the `INSTALL` file that you are currently reading), `$BUILD_DIR` denotes a directory (which is either empty or to be newly created) to be used for building the software, and `$INSTALL_DIR` denotes the directory under which the software should be installed.

Note that in-source builds are not supported. So, `$BUILD_DIR` must be different from `$SOURCE_DIR`.

To build the software, the following steps are required (in order):

1. Install the prerequisite libraries.

Ensure that all of the libraries needed by the SPL software are installed on your system. This includes the following libraries:

1) Computational Geometry Algorithms Library (CGAL)
<http://www.cgal.org>

2) OpenGL Utility Toolkit (GLUT)
<http://www.opengl.org/resources/libraries/glut/>
<http://freeglut.sourceforge.net>

3) Sndfile Library
<http://www.mega-nerd.com/libsndfile>

Some platforms provide a package manager for installing software packages and their dependencies. For example, Fedora provides the DNF package manager, which is accessed via the `dnf` command, and Ubuntu provides the Apt package manager, which is accessed via the `apt-get` command. If a package manager is available on your system, it can likely be used to simplify the process of installing some of the above prerequisite libraries. For example, on a Fedora system with the DNF package manager, some of the above libraries could be installed by using a sequence of commands resembling the following:

```
dnf install CGAL CGAL-devel CGAL-demos-source mpfr-devel
dnf install freeglut freeglut-devel
dnf install libsndfile libsndfile-devel
```

Note that the specific package names required by the package manager will vary from one version of Fedora to another and may not exactly match those appearing above.

2. Generate the native build files.

If the build directory `$BUILD_DIR` does not exist, create it by using the command:

```
mkdir -p $BUILD_DIR
```

Generate the build files for the native build tool on your system using the command:

```
cd $BUILD_DIR
cmake -DCMAKE_INSTALL_PREFIX=$INSTALL_DIR $OPTIONS $SOURCE_DIR
```

where `$OPTIONS` corresponds to zero or more `-D` options as described in the later section titled "Cmake Options".

3. Build the software.

To build the software, use the command:

```
cmake --build $BUILD_DIR --clean-first
```

4. Test the software (prior to installation).

Run some basic sanity checks on the software, prior to installation. Although this step is not strictly required, it is strongly recommended that this step not be skipped. To test the software, use the command:

```
cd $BUILD_DIR
ctest --output-on-failure
```

Some tests may require considerable time to complete. So, be prepared to go for a coffee break. After all of the tests have run, a message should be printed that indicates how many tests passed of those that were run. If any of the tests failed, this is an indication that something is wrong and the SPL software is not working reliably. If such a situation arises, it is likely due to either an error made by the person installing the software or a bug in the software itself.

Some of the tests may require graphics capabilities. If the graphics display is not on the local machine, this can sometimes cause problems. For example, some tests may fail due to bugs in the graphics libraries that handle remote displays. In such cases, it may be desirable to disable tests that require graphics capabilities. To do this, set the environment variable `SPL_MAKE_CHECK_ENABLE_GRAPHICS` to 0 (before running `ctest`).

5. Install the software.

The actual installation of the software may require special administrator privileges depending on the target directory for installation (i.e., `$INSTALL_DIR`). To install the executables, libraries, include files, and

other auxiliary data, use the command:

```
cmake --build $BUILD_DIR --target install
```

Cmake Options

The option OPTION can be set to the value VALUE with a command-line option of the form -DOPTION=VALUE

The following options are supported:

CMAKE_INSTALL_PREFIX

Specify the installation directory.
Value: A directory name.

CMAKE_BUILD_TYPE

Specify the build type (i.e., release or debug).
Valid values: Debug or Release

SPL_ENABLE_ASAN

Enable the Address Sanitizer.
Valid values: true or false

SPL_ENABLE_USAN

Enable the Undefined-Behavior Sanitizer.
Valid values: true or false

SPL_ENABLE_LSAN

Enable the Leak Sanitizer.
Valid values: true or false

SPL_ENABLE_MSAN

Enable the Memory Sanitizer.
Valid values: true or false

2.2 Organization of the Manual

The library is partitioned into groups of related code called modules. The documentation is also partitioned in this way. The documentation for each of the various modules can be found in the [modules page](#).

Chapter 3

Frequently Asked Questions (FAQ)

The following is a list of common questions/problems encountered when using the library.

- I would like to avoid many stressful days and sleepless nights debugging my code that uses the SPL library. Do you have any suggestions on how I can avoid needless mistakes? Are there any common pitfalls that I should be particularly careful to avoid?

Sadly, one of the most common sources of problems is not reading the documentation and making incorrect assumptions about how the library works. Unfortunately, an incorrect assumption about how the library works can be quite costly, leading to hours of unnecessary debugging time.

- My code is triggering a failed assertion in the SPL library. What does this mean?

The SPL library makes frequent use of assertions in order to assist in the detection of bugs. If an assertion fails, the code has encountered a situation that should never occur in correct code. The most likely cause is that you are using the library incorrectly. Be sure that you have read the documentation for the parts of the library that you are using in order to ensure that you are using it correctly.

Chapter 4

Known Bugs

Currently, there are no known bugs in the library.

Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

Arrays and Sequences	19
One-Dimensional Arrays	20
Two-Dimensional Arrays	27
One-Dimensional Sequences	36
Two-Dimensional Sequences	48
Bit Stream I/O	61
Audio and Image Codecs	64
Audio Codecs	65
Image Codecs	66
Filter Design	69
CPU and Memory Utilization	71
Math Utilities	72
CGAL Utilities	75
Arithmetic Coders	79
M-Coder	80
Binary and m-ary Arithmetic Coders	82

Chapter 6

Hierarchical Index

6.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

SPL::Arcball< T >	89
SPL::Arcball< Kernel >	89
SPL::Array1< T >	91
SPL::Array2< T >	93
SPL::BinArithCoderContextStat	97
SPL::BinArithDecoder	98
SPL::BinArithEncoder	100
SPL::BitStream	104
SPL::InputBitStream	107
SPL::OutputBitStream	119
SPL::ConvolveMode	106
SPL::MDecoder	110
SPL::MEncoder	112
SPL::MultiArithDecoder	115
SPL::MultiArithEncoder	117
SPL::PnmHeader	122
SPL::Quaternion< T >	123
SPL::Rotation_3< T >	125
SPL::Rotation_3< Kernel >	125
SPL::Sequence1< T >	126
SPL::Sequence2< T >	129
SPL::Timer	133

Chapter 7

Class Index

7.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

SPL::Arcball< T >	
Arcball	89
SPL::Array1< T >	
A one-dimensional array class with lazy copying and reference counting	91
SPL::Array2< T >	
A two-dimensional array class with lazy copying and reference counting	93
SPL::BinArithCoderContextStat	
Binary Arithmetic Coder Context Statistics Class	97
SPL::BinArithDecoder	
Binary arithmetic decoder class	98
SPL::BinArithEncoder	
Binary arithmetic encoder class	100
SPL::BitStream	
A common base class for the input and output bit stream classes	104
SPL::ConvolveMode	
Constants identifying various convolution modes	106
SPL::InputBitStream	
Input bit stream class	107
SPL::MDecoder	
The M-Coder (binary) arithmetic decoder class	110
SPL::MEncoder	
The M-Coder (binary) arithmetic encoder class	112
SPL::MultiArithDecoder	
M-ary arithmetic decoder class	115
SPL::MultiArithEncoder	
M-ary arithmetic encoder class	117
SPL::OutputBitStream	
Output bit stream class	119
SPL::PnmHeader	
The header information for PNM data	122
SPL::Quaternion< T >	
A quaternion represented in terms of its scalar and vector parts	123
SPL::Rotation_3< T >	
A 3-D rotation	125

SPL::Sequence1< T >	
A one-dimensional sequence class with lazy copying and reference counting	126
SPL::Sequence2< T >	
A two-dimensional sequence class with lazy copying and reference counting	129
SPL::Timer	
A class for making timing measurements	133

Chapter 8

File Index

8.1 File List

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

Arcball.hpp	This file contains the Arcball class and related code	135
demo/arithCoder.hpp	??
include/SPL/arithCoder.hpp	??
Array1.hpp	This file contains the Array1 template class and supporting code	136
Array2.hpp	This file contains the Array2 template class and its supporting code	137
audioFile.hpp	This file contains code for performing reading and writing of audio files in WAV format	139
bitStream.hpp	Bit Stream Classes	139
cgalUtil.hpp	This file contains various CGAL utility code	140
filterDesign.hpp	This file contains code for performing filter design	141
math.hpp	This file contains various mathematical functions/code	141
mCoder.hpp	This file contains interface information for an implementation of the M-Coder arithmetic coder from: ISO/IEC 14496-10:2008 (a.k.a. H.264)	142
misc.hpp	This file contains miscellaneous code	143
pnmCodec.cpp	This file contains a PNM codec	143
pnmCodec.hpp	This file contains a PNM codec	144
Sequence.hpp	Common header for sequence classes	146
Sequence1.hpp	This file contains code for the Sequence1 template class	146
Sequence2.cpp	This file contains code for the Sequence2 template class	149

Sequence2.hpp	
This file contains code for the Sequence2 template class	149
Timer.cpp	
The file contains code for obtaining timing/memory usage information	152
Timer.hpp	
This file contains code for the Timer class	152

Chapter 9

Module Documentation

9.1 Arrays and Sequences

One- and two-dimensional arrays and sequences.

Modules

- [One-Dimensional Arrays](#)
One-dimensional arrays.
- [Two-Dimensional Arrays](#)
Two-dimensional arrays.
- [One-Dimensional Sequences](#)
One-dimensional sequences.
- [Two-Dimensional Sequences](#)
Two-dimensional sequences.

9.1.1 Detailed Description

One- and two-dimensional arrays and sequences.

9.2 One-Dimensional Arrays

One-dimensional arrays.

Classes

- class `SPL::Array1< T >`
A one-dimensional array class with lazy copying and reference counting.

Typedefs

- typedef `Array1< double > SPL::RealArray1`
A one-dimensional array with real elements.
- typedef `Array1< int > SPL::IntArray1`
A one-dimensional array with integer elements.

Functions

- template<class T >
`std::ostream & SPL::operator<< (std::ostream &out, const Array1< T > &a)`
Output an array to the specified stream.
- template<class T >
`std::istream & SPL::operator>> (std::istream &in, Array1< T > &a)`
Input an array from the specified stream.
- template<class T >
`bool SPL::operator== (const Array1< T > &a, const Array1< T > &b)`
Test two arrays for equality.
- template<class T >
`SPL_ARRAY1_INLINE bool SPL::operator!= (const Array1< T > &a, const Array1< T > &b)`
Test two arrays for inequality.
- `SPL::Array1< T >::Array1 ()`
Create an empty array.
- `SPL::Array1< T >::Array1 (int size)`
Create an array of the specified size.
- `SPL::Array1< T >::Array1 (const Array1 &a)`
Create a copy of an array.
- `SPL::Array1< T >::Array1 (int size, const T &value)`
Create an array of the given size with all elements initialized to the specified value.
- `SPL::Array1< T >::~~Array1 ()`
Destroy an array.
- `Array1 & SPL::Array1< T >::operator= (const Array1 &a)`
Assign one array to another.
- template<class OtherType >
`Array1< T > & SPL::Array1< T >::operator= (const Array1< OtherType > &a)`
Assign an array with elements of arbitrary type to another array.
- `Array1 & SPL::Array1< T >::operator+= (const Array1 &a)`
Add another array (elementwise) to this array.

- `Array1 & SPL::Array1< T >::operator-= (const Array1 &a)`
Subtract another array (elementwise) from this array.
- `Array1 & SPL::Array1< T >::operator*= (const Array1 &a)`
Multiply another array (elementwise) by this array.
- `Array1 & SPL::Array1< T >::operator/= (const Array1 &a)`
Divide this array (elementwise) by another array.
- `Array1 & SPL::Array1< T >::operator+= (const T &value)`
Add the specified value to each element in the array.
- `Array1 & SPL::Array1< T >::operator-= (const T &value)`
Subtract the specified value from each element in the array.
- `Array1 & SPL::Array1< T >::operator*= (const T &value)`
Multiply each element in the array by the specified value.
- `Array1 & SPL::Array1< T >::operator/= (const T &value)`
Divide each element in the array by the specified value.
- `int SPL::Array1< T >::getSize () const`
Get the number of elements in the array.
- `bool SPL::Array1< T >::isShared () const`
Is the data for this array shared with another array?
- `bool SPL::Array1< T >::isSharedWith (const Array1 &a) const`
Is the data for this array shared with the specified array?
- `T & SPL::Array1< T >::operator() (int i)`
Get a mutable reference to the specified element in the array.
- `const T & SPL::Array1< T >::operator() (int i) const`
Get a const reference to the specified element in the array.
- `Iterator SPL::Array1< T >::begin ()`
Get a mutable iterator referring to the first element in the array.
- `Iterator SPL::Array1< T >::end ()`
Get a mutable iterator referring to one past the last element in the array.
- `ConstIterator SPL::Array1< T >::begin () const`
Get a const iterator referring to the first element in the array.
- `ConstIterator SPL::Array1< T >::end () const`
Get a const iterator referring to one past the last element in the array.
- `void SPL::Array1< T >::resize (int size)`
Change the size of the array.
- `template<class InputIterator >`
`void SPL::Array1< T >::resize (int size, InputIterator data)`
Change the size of the array, initializing the resized array with the data obtained from the specified input iterator.
- `T SPL::Array1< T >::max () const`
Get the maximum of the elements in the array.
- `T SPL::Array1< T >::min () const`
Get the minimum of the elements in the array.
- `T SPL::Array1< T >::sum () const`
Get the sum of the elements in the array.
- `std::ostream & SPL::Array1< T >::output (std::ostream &out, int fieldWidth) const`
Output an array to a stream with a particular field width to be used for each element.
- `int SPL::Array1< T >::load (const char *fileName)`
Load an array from the file with the specified name.

- `int SPL::Array1< T >::save` (const char *fileName) const
Save an array to the file with the specified name.
- `void SPL::Array1< T >::swap` (Array1 &a)
Swap the contents of the array with the contents of another array.
- `void SPL::Array1< T >::fill` (const T &value=T(0))
Set all elements in the array to the specified value.
- `void SPL::Array1< T >::dump` (std::ostream &out) const
Output information about an array to a stream for debugging.

9.2.1 Detailed Description

One-dimensional arrays.

9.2.2 Typedef Documentation

9.2.2.1 `typedef Array1<int> SPL::IntArray1`

A one-dimensional array with integer elements.

9.2.2.2 `typedef Array1<double> SPL::RealArray1`

A one-dimensional array with real elements.

9.2.3 Function Documentation

9.2.3.1 `template<class T > SPL_ARRAY1_INLINE SPL::Array1< T >::Array1 ()`

Create an empty array.

9.2.3.2 `template<class T > SPL_ARRAY1_INLINE SPL::Array1< T >::Array1 (int size) [explicit]`

Create an array of the specified size.

9.2.3.3 `template<class T > SPL_ARRAY1_INLINE SPL::Array1< T >::Array1 (int size, const T & value)`

Create an array of the given size with all elements initialized to the specified value.

9.2.3.4 `template<class T > SPL_ARRAY1_INLINE SPL::Array1< T >::Array1 (const Array1< T > & a)`

Create a copy of an array.

9.2.3.5 `template<class T > SPL_ARRAY1_INLINE Array1< T >::ConstIterator SPL::Array1< T >::begin () const`

Get a const iterator referring to the first element in the array.

9.2.3.6 `template<class T> SPL_ARRAY1_INLINE Array1<T>::iterator SPL::Array1<T>::begin ()`

Get a mutable iterator referring to the first element in the array.

9.2.3.7 `template<class T> void SPL::Array1<T>::dump (std::ostream & out) const`

Output information about an array to a stream for debugging.

9.2.3.8 `template<class T> SPL_ARRAY1_INLINE Array1<T>::ConstIterator SPL::Array1<T>::end () const`

Get a const iterator referring to one past the last element in the array.

9.2.3.9 `template<class T> SPL_ARRAY1_INLINE Array1<T>::iterator SPL::Array1<T>::end ()`

Get a mutable iterator referring to one past the last element in the array.

9.2.3.10 `template<class T> SPL_ARRAY1_INLINE void SPL::Array1<T>::fill (const T & value = T (0))`

Set all elements in the array to the specified value.

9.2.3.11 `template<class T> SPL_ARRAY1_INLINE int SPL::Array1<T>::getSize () const`

Get the number of elements in the array.

9.2.3.12 `template<class T> SPL_ARRAY1_INLINE bool SPL::Array1<T>::isShared () const`

Is the data for this array shared with another array?

Under most normal circumstances, one should never need to call this function. In some instances, however, it might be necessary to know whether data is shared in order to write more optimal code.

9.2.3.13 `template<class T> SPL_ARRAY1_INLINE bool SPL::Array1<T>::isSharedWith (const Array1<T> & a) const`

Is the data for this array shared with the specified array?

9.2.3.14 `template<class T> int SPL::Array1<T>::load (const char * fileName)`

Load an array from the file with the specified name.

9.2.3.15 `template<class T> SPL_ARRAY1_INLINE T SPL::Array1<T>::max () const`

Get the maximum of the elements in the array.

The array must contain at least one element.

9.2.3.16 `template<class T> SPL_ARRAY1_INLINE T SPL::Array1< T>::min () const`

Get the minimum of the elements in the array.

The array must contain at least one element.

9.2.3.17 `template<class T> SPL_ARRAY1_INLINE bool SPL::operator!=(const Array1< T> & a, const Array1< T> & b)`

Test two arrays for inequality.

9.2.3.18 `template<class T> SPL_ARRAY1_INLINE T & SPL::Array1< T>::operator()(int i)`

Get a mutable reference to the specified element in the array.

9.2.3.19 `template<class T> SPL_ARRAY1_INLINE const T & SPL::Array1< T>::operator()(int i) const`

Get a const reference to the specified element in the array.

9.2.3.20 `template<class T> SPL_ARRAY1_INLINE Array1< T> & SPL::Array1< T>::operator*=(const Array1< T> & a)`

Multiply another array (elementwise) by this array.

9.2.3.21 `template<class T> SPL_ARRAY1_INLINE Array1< T> & SPL::Array1< T>::operator*=(const T & value)`

Multiply each element in the array by the specified value.

9.2.3.22 `template<class T> SPL_ARRAY1_INLINE Array1< T> & SPL::Array1< T>::operator+=(const Array1< T> & a)`

Add another array (elementwise) to this array.

9.2.3.23 `template<class T> SPL_ARRAY1_INLINE Array1< T> & SPL::Array1< T>::operator+=(const T & value)`

Add the specified value to each element in the array.

9.2.3.24 `template<class T> SPL_ARRAY1_INLINE Array1< T> & SPL::Array1< T>::operator-=(const Array1< T> & a)`

Subtract another array (elementwise) from this array.

9.2.3.25 `template<class T> SPL_ARRAY1_INLINE Array1< T> & SPL::Array1< T>::operator-=(const T & value)`

Subtract the specified value from each element in the array.

9.2.3.26 `template<class T> SPL_ARRAY1_INLINE Array1< T> & SPL::Array1< T>::operator/= (const Array1< T> & a)`

Divide this array (elementwise) by another array.

9.2.3.27 `template<class T> SPL_ARRAY1_INLINE Array1< T> & SPL::Array1< T>::operator/= (const T & value)`

Divide each element in the array by the specified value.

9.2.3.28 `template<class T> std::ostream& SPL::operator<< (std::ostream & out, const Array1< T> & a)`

Output an array to the specified stream.

9.2.3.29 `template<class T> SPL_ARRAY1_INLINE Array1< T> & SPL::Array1< T>::operator= (const Array1< T> & a)`

Assign one array to another.

9.2.3.30 `template<class T> template<class OtherType> Array1< T> & SPL::Array1< T>::operator= (const Array1< OtherType> & a)`

Assign an array with elements of arbitrary type to another array.

9.2.3.31 `template<class T> bool SPL::operator== (const Array1< T> & a, const Array1< T> & b)`

Test two arrays for equality.

9.2.3.32 `template<class T> std::istream& SPL::operator>> (std::istream & in, Array1< T> & a)`

Input an array from the specified stream.

9.2.3.33 `template<class T> std::ostream & SPL::Array1< T>::output (std::ostream & out, int fieldWidth) const`

Output an array to a stream with a particular field width to be used for each element.

9.2.3.34 `template<class T> void SPL::Array1< T>::resize (int size)`

Change the size of the array.

Effects: The array size is changed to the specified size. If the new size is the same as the old size, this function does nothing.

9.2.3.35 `template<class T> template<class InputIterator> void SPL::Array1< T>::resize (int size, InputIterator data)`

Change the size of the array, initializing the resized array with the data obtained from the specified input iterator.

9.2.3.36 `template<class T> int SPL::Array1< T>::save (const char * fileName) const`

Save an array to the file with the specified name.

9.2.3.37 `template<class T> SPL_ARRAY1_INLINE T SPL::Array1< T>::sum () const`

Get the sum of the elements in the array.

9.2.3.38 `template<class T> SPL_ARRAY1_INLINE void SPL::Array1< T>::swap (Array1< T> & a)`

Swap the contents of the array with the contents of another array.

9.2.3.39 `template<class T> SPL_ARRAY1_INLINE SPL::Array1< T>::~~Array1 ()`

Destroy an array.

9.3 Two-Dimensional Arrays

Two-dimensional arrays.

Classes

- class `SPL::Array2< T >`
A two-dimensional array class with lazy copying and reference counting.

Typedefs

- typedef `Array2< double > SPL::RealArray2`
A two-dimensional array with real elements.
- typedef `Array2< int > SPL::IntArray2`
A two-dimensional array with integer elements.

Functions

- template<class T >
`std::ostream & SPL::operator<< (std::ostream &out, const Array2< T > &a)`
Output an array to the specified stream.
- template<class T >
`std::istream & SPL::operator>> (std::istream &in, Array2< T > &a)`
Input an array from the specified stream.
- template<class T >
`Array2< T > SPL::transpose (const Array2< T > &a)`
Get the transpose of the array.
- template<class T >
`bool SPL::operator== (const Array2< T > &a, const Array2< T > &b)`
Test two arrays for equality.
- template<class T >
`bool SPL::operator!= (const Array2< T > &a, const Array2< T > &b)`
Test two arrays for inequality.
- `SPL::Array2< T >::Array2 ()`
Create an empty array.
- `SPL::Array2< T >::Array2 (int width, int height)`
Create an array of the specified width and height.
- `SPL::Array2< T >::Array2 (const Array2 &a)`
The copy constructor.
- `SPL::Array2< T >::Array2 (int width, int height, const T &value)`
Create an array of the specified width and height with the elements of the array initialized to the specified value.
- `SPL::Array2< T >::~~Array2 ()`
The destructor.
- `Array2 & SPL::Array2< T >::operator= (const Array2 &a)`
The assignment operator.
- `Array2 & SPL::Array2< T >::operator+= (const Array2 &a)`
Add another array (elementwise) to this array.

- `Array2 & SPL::Array2< T >::operator-= (const Array2 &a)`
Subtract another array (elementwise) from this array.
- `Array2 & SPL::Array2< T >::operator*= (const Array2 &a)`
Multiply another array (elementwise) by this array.
- `Array2 & SPL::Array2< T >::operator/= (const Array2 &a)`
Divide this array (elementwise) by another array.
- `Array2 & SPL::Array2< T >::operator+= (const T &a)`
Add the specified value to each element in the array.
- `Array2 & SPL::Array2< T >::operator-= (const T &a)`
Subtract the specified value from each element in the array.
- `Array2 & SPL::Array2< T >::operator*= (const T &a)`
Multiply each element in the array by the specified value.
- `Array2 & SPL::Array2< T >::operator/= (const T &a)`
Divide each element in the array by the specified value.
- `int SPL::Array2< T >::getWidth () const`
Get the width of the array.
- `int SPL::Array2< T >::getHeight () const`
Get the height of the array.
- `int SPL::Array2< T >::getSize () const`
Get the number of elements in the array.
- `bool SPL::Array2< T >::isShared () const`
Is the data for this array shared with another array?
- `bool SPL::Array2< T >::isSharedWith (const Array2 &a) const`
Is the data for this array shared with the specified array?
- `T & SPL::Array2< T >::operator() (int x, int y)`
Get a mutable reference to the (x,y)-th element in the array.
- `const T & SPL::Array2< T >::operator() (int x, int y) const`
Get a const reference to the (x,y)-th element in the array.
- `T & SPL::Array2< T >::operator() (int i)`
Get a mutable reference to the i-th element in the array.
- `const T & SPL::Array2< T >::operator() (int i) const`
Get a const reference to the i-th element in the array.
- `ConstIterator SPL::Array2< T >::begin () const`
Get a const iterator for the first element in the array.
- `Iterator SPL::Array2< T >::begin ()`
Get a mutable iterator for the first element in the array.
- `ConstIterator SPL::Array2< T >::end () const`
Get a const iterator for one past the last element in the array.
- `Iterator SPL::Array2< T >::end ()`
Get a mutable iterator for one past the last element in the array.
- `ConstXIterator SPL::Array2< T >::rowBegin (int y) const`
Get a const iterator for the first element in the specified row of the array.
- `XIterator SPL::Array2< T >::rowBegin (int y)`
Get a mutable iterator for the first element in the specified row of the array.
- `ConstXIterator SPL::Array2< T >::rowEnd (int y) const`
Get a const iterator for one past the end in the specified row of the array.
- `XIterator SPL::Array2< T >::rowEnd (int y)`

- Get a mutable iterator for one past the end in the specified row of the array.*
- ConstYIterator `SPL::Array2< T >::colBegin` (int x) const
- Get a const iterator for the first element in the specified column of the array.*
- YIterator `SPL::Array2< T >::colBegin` (int x)
- Get a mutable iterator for the first element in the specified column of the array.*
- ConstYIterator `SPL::Array2< T >::colEnd` (int x) const
- Get a const iterator for one past the end in the specified column of the array.*
- YIterator `SPL::Array2< T >::colEnd` (int x)
- Get a mutable iterator for one past the end in the specified column of the array.*
- void `SPL::Array2< T >::resize` (int width, int height)
- Change the size of the array.*
- template<class InputIterator >
void `SPL::Array2< T >::resize` (int width, int height, InputIterator data)
- Change the size of the array, initializing the resized array with the data obtained from the specified input iterator.*
- T `SPL::Array2< T >::max` () const
- Get the maximum of the elements in the array.*
- T `SPL::Array2< T >::min` () const
- Get the minimum of the elements in the array.*
- T `SPL::Array2< T >::sum` () const
- Get the sum of the elements in the array.*
- std::ostream & `SPL::Array2< T >::output` (std::ostream &out, int fieldWidth) const
- Output an array to a stream using the specified field width for each array element.*
- int `SPL::Array2< T >::load` (const char *fileName)
- Load an array from the file with the specified name.*
- int `SPL::Array2< T >::save` (const char *fileName) const
- Save an array to the file with the specified name.*
- void `SPL::Array2< T >::swap` (Array2 &a)
- Swap the array data with the data of the specified array.*
- void `SPL::Array2< T >::fill` (const T &value=T(0))
- Set all elements in the array to the specified value.*
- Array2 & `SPL::Array2< T >::flipud` ()
- Flip the array upside down.*
- Array2 & `SPL::Array2< T >::fliplr` ()
- Flip the array left to right.*
- void `SPL::Array2< T >::dump` (std::ostream &out) const
- Output information about an array to a stream for debugging.*
- void `SPL::Array2< T >::unshare` () const
- Force the underlying data to be copied if the data is shared.*

9.3.1 Detailed Description

Two-dimensional arrays.

9.3.2 Typedef Documentation

9.3.2.1 typedef Array2<int> SPL::IntArray2

A two-dimensional array with integer elements.

9.3.2.2 `typedef Array2<double> SPL::RealArray2`

A two-dimensional array with real elements.

9.3.3 Function Documentation

9.3.3.1 `template<class T> SPL_ARRAY2_INLINE SPL::Array2<T>::Array2 ()`

Create an empty array.

9.3.3.2 `template<class T> SPL_ARRAY2_INLINE SPL::Array2<T>::Array2 (int width, int height)`

Create an array of the specified width and height.

9.3.3.3 `template<class T> SPL_ARRAY2_INLINE SPL::Array2<T>::Array2 (int width, int height, const T & value)`

Create an array of the specified width and height with the elements of the array initialized to the specified value.

9.3.3.4 `template<class T> SPL_ARRAY2_INLINE SPL::Array2<T>::Array2 (const Array2<T> & a)`

The copy constructor.

9.3.3.5 `template<class T> SPL_ARRAY2_INLINE Array2<T>::ConstIterator SPL::Array2<T>::begin () const`

Get a const iterator for the first element in the array.

9.3.3.6 `template<class T> SPL_ARRAY2_INLINE Array2<T>::Iterator SPL::Array2<T>::begin ()`

Get a mutable iterator for the first element in the array.

9.3.3.7 `template<class T> SPL_ARRAY2_INLINE Array2<T>::ConstYIterator SPL::Array2<T>::colBegin (int x) const`

Get a const iterator for the first element in the specified column of the array.

9.3.3.8 `template<class T> SPL_ARRAY2_INLINE Array2<T>::YIterator SPL::Array2<T>::colBegin (int x)`

Get a mutable iterator for the first element in the specified column of the array.

9.3.3.9 `template<class T> SPL_ARRAY2_INLINE Array2<T>::ConstYIterator SPL::Array2<T>::colEnd (int x) const`

Get a const iterator for one past the end in the specified column of the array.

9.3.3.10 `template<class T> SPL_ARRAY2_INLINE Array2<T>::YIterator SPL::Array2<T>::colEnd (int x)`

Get a mutable iterator for one past the end in the specified column of the array.

9.3.3.11 `template<class T> void SPL::Array2<T>::dump (std::ostream & out) const`

Output information about an array to a stream for debugging.

Output information about an array to the specified stream for debugging purposes.

9.3.3.12 `template<class T> SPL_ARRAY2_INLINE Array2<T>::ConstIterator SPL::Array2<T>::end () const`

Get a const iterator for one past the last element in the array.

9.3.3.13 `template<class T> SPL_ARRAY2_INLINE Array2<T>::Iterator SPL::Array2<T>::end ()`

Get a mutable iterator for one past the last element in the array.

9.3.3.14 `template<class T> SPL_ARRAY2_INLINE void SPL::Array2<T>::fill (const T & value = T (0))`

Set all elements in the array to the specified value.

9.3.3.15 `template<class T> Array2<T> & SPL::Array2<T>::fliplr ()`

Flip the array left to right.

9.3.3.16 `template<class T> Array2<T> & SPL::Array2<T>::flipud ()`

Flip the array upside down.

9.3.3.17 `template<class T> SPL_ARRAY2_INLINE int SPL::Array2<T>::getHeight () const`

Get the height of the array.

9.3.3.18 `template<class T> SPL_ARRAY2_INLINE int SPL::Array2<T>::getSize () const`

Get the number of elements in the array.

9.3.3.19 `template<class T> SPL_ARRAY2_INLINE int SPL::Array2<T>::getWidth () const`

Get the width of the array.

9.3.3.20 `template<class T> SPL_ARRAY2_INLINE bool SPL::Array2<T>::isShared () const`

Is the data for this array shared with another array?

Under most normal circumstances, one should never need to call this function. In some instances, however, it might be necessary to know whether data is shared in order to write more optimal code.

9.3.3.21 `template<class T> SPL_ARRAY2_INLINE bool SPL::Array2< T>::isSharedWith (const Array2< T> & a) const`

Is the data for this array shared with the specified array?

9.3.3.22 `template<class T> int SPL::Array2< T>::load (const char * fileName)`

Load an array from the file with the specified name.

9.3.3.23 `template<class T> SPL_ARRAY2_INLINE T SPL::Array2< T>::max () const`

Get the maximum of the elements in the array.

The array must contain at least one element.

9.3.3.24 `template<class T> SPL_ARRAY2_INLINE T SPL::Array2< T>::min () const`

Get the minimum of the elements in the array.

The array must contain at least one element.

9.3.3.25 `template<class T> bool SPL::operator!=(const Array2< T> & a, const Array2< T> & b)`

Test two arrays for inequality.

9.3.3.26 `template<class T> SPL_ARRAY2_INLINE T & SPL::Array2< T>::operator() (int x, int y)`

Get a mutable reference to the (x,y)-th element in the array.

9.3.3.27 `template<class T> SPL_ARRAY2_INLINE const T & SPL::Array2< T>::operator() (int x, int y) const`

Get a const reference to the (x,y)-th element in the array.

9.3.3.28 `template<class T> SPL_ARRAY2_INLINE T & SPL::Array2< T>::operator() (int i)`

Get a mutable reference to the i-th element in the array.

The array must have either a width or height of one.

9.3.3.29 `template<class T> SPL_ARRAY2_INLINE const T & SPL::Array2< T>::operator() (int i) const`

Get a const reference to the i-th element in the array.

The array must have either a width or height of one.

9.3.3.30 `template<class T> SPL_ARRAY2_INLINE Array2< T> & SPL::Array2< T>::operator*= (const Array2< T> & a)`

Multiply another array (elementwise) by this array.

9.3.3.31 `template<class T> SPL_ARRAY2_INLINE Array2< T> & SPL::Array2< T>::operator*=(const T & a)`

Multiply each element in the array by the specified value.

9.3.3.32 `template<class T> SPL_ARRAY2_INLINE Array2< T> & SPL::Array2< T>::operator+=(const Array2< T> & a)`

Add another array (elementwise) to this array.

9.3.3.33 `template<class T> SPL_ARRAY2_INLINE Array2< T> & SPL::Array2< T>::operator+=(const T & a)`

Add the specified value to each element in the array.

9.3.3.34 `template<class T> SPL_ARRAY2_INLINE Array2< T> & SPL::Array2< T>::operator-=(const Array2< T> & a)`

Subtract another array (elementwise) from this array.

9.3.3.35 `template<class T> SPL_ARRAY2_INLINE Array2< T> & SPL::Array2< T>::operator-=(const T & a)`

Subtract the specified value from each element in the array.

9.3.3.36 `template<class T> SPL_ARRAY2_INLINE Array2< T> & SPL::Array2< T>::operator/=(const Array2< T> & a)`

Divide this array (elementwise) by another array.

9.3.3.37 `template<class T> SPL_ARRAY2_INLINE Array2< T> & SPL::Array2< T>::operator/=(const T & a)`

Divide each element in the array by the specified value.

9.3.3.38 `template<class T> std::ostream& SPL::operator<< (std::ostream & out, const Array2< T> & a)`

Output an array to the specified stream.

9.3.3.39 `template<class T> SPL_ARRAY2_INLINE Array2< T> & SPL::Array2< T>::operator=(const Array2< T> & a)`

The assignment operator.

9.3.3.40 `template<class T> bool SPL::operator==(const Array2< T> & a, const Array2< T> & b)`

Test two arrays for equality.

9.3.3.41 `template<class T> std::istream& SPL::operator>> (std::istream & in, Array2< T> & a)`

Input an array from the specified stream.

9.3.3.42 `template<class T> std::ostream & SPL::Array2< T>::output (std::ostream & out, int fieldWidth) const`

Output an array to a stream using the specified field width for each array element.

9.3.3.43 `template<class T> void SPL::Array2< T>::resize (int width, int height)`

Change the size of the array.

Effects: The array size is changed to the specified size. If the new size is the same as the old size, this function does nothing.

9.3.3.44 `template<class T> template<class InputIterator> void SPL::Array2< T>::resize (int width, int height, InputIterator data)`

Change the size of the array, initializing the resized array with the data obtained from the specified input iterator.

9.3.3.45 `template<class T> SPL_ARRAY2_INLINE Array2< T>::ConstXIterator SPL::Array2< T>::rowBegin (int y) const`

Get a const iterator for the first element in the specified row of the array.

9.3.3.46 `template<class T> SPL_ARRAY2_INLINE Array2< T>::XIterator SPL::Array2< T>::rowBegin (int y)`

Get a mutable iterator for the first element in the specified row of the array.

9.3.3.47 `template<class T> SPL_ARRAY2_INLINE Array2< T>::ConstXIterator SPL::Array2< T>::rowEnd (int y) const`

Get a const iterator for one past the end in the specified row of the array.

9.3.3.48 `template<class T> SPL_ARRAY2_INLINE Array2< T>::XIterator SPL::Array2< T>::rowEnd (int y)`

Get a mutable iterator for one past the end in the specified row of the array.

9.3.3.49 `template<class T> int SPL::Array2< T>::save (const char * fileName) const`

Save an array to the file with the specified name.

9.3.3.50 `template<class T> SPL_ARRAY2_INLINE T SPL::Array2< T>::sum () const`

Get the sum of the elements in the array.

9.3.3.51 `template<class T> void SPL::Array2< T>::swap (Array2< T> & a)`

Swap the array data with the data of the specified array.

9.3.3.52 `template<class T> Array2<T> SPL::transpose (const Array2< T> & a)`

Get the transpose of the array.

9.3.3.53 `template<class T> SPL_ARRAY2_INLINE void SPL::Array2< T>::unshare () const`

Force the underlying data to be copied if the data is shared.

9.3.3.54 `template<class T> SPL_ARRAY2_INLINE SPL::Array2< T>::~~Array2 ()`

The destructor.

9.4 One-Dimensional Sequences

One-dimensional sequences.

Classes

- class `SPL::Sequence1< T >`

A one-dimensional sequence class with lazy copying and reference counting.

Typedefs

- typedef `Sequence1< double >` `SPL::RealSequence1`

Real sequence.

- typedef `Sequence1< int >` `SPL::IntSequence1`

Integer sequence.

Functions

- template<class T >
`std::ostream & SPL::operator<< (std::ostream &out, const Sequence1< T > &f)`
Output a sequence to a stream.
- template<class T >
`std::istream & SPL::operator>> (std::istream &in, Sequence1< T > &f)`
Input a sequence from a stream.
- template<class T >
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator+ (const Sequence1< T > &f, const Sequence1< T > &g)`
Compute the sum of two sequences.
- template<class T >
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator- (const Sequence1< T > &f, const Sequence1< T > &g)`
Compute the difference of two sequences.
- template<class T >
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator* (const Sequence1< T > &f, const Sequence1< T > &g)`
Compute the (element-wise) product of two sequences.
- template<class T >
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator/ (const Sequence1< T > &f, const Sequence1< T > &g)`
Compute the (element-wise) quotient of two sequences.
- template<class T >
`Sequence1< T > SPL::add (const Sequence1< T > &f, const Sequence1< T > &g)`
Compute the sum of two sequences with potentially differing domains.
- template<class T >
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator+ (const T &a, const Sequence1< T > &f)`
Add a value to a sequence.
- template<class T >
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator+ (const Sequence1< T > &f, const T &a)`

- Add a value to a sequence.*

 - `template<class T >`
`SPL_SEQUENCE1_INLINE` `Sequence1< T > SPL::operator+ (const Sequence1< T > &f, const T &a)`

Subtract a value from a sequence.

 - `template<class T >`
`SPL_SEQUENCE1_INLINE` `Sequence1< T > SPL::operator- (const T &a, const Sequence1< T > &f)`

Compute a scalar multiple of a sequence.

 - `template<class T >`
`SPL_SEQUENCE1_INLINE` `Sequence1< T > SPL::operator* (const Sequence1< T > &f, const T &a)`

Compute a scalar multiple of a sequence.

 - `template<class T >`
`SPL_SEQUENCE1_INLINE` `Sequence1< T > SPL::operator/ (const Sequence1< T > &f, const T &a)`

Divide a sequence by a scalar.

 - `template<class T >`
`SPL_SEQUENCE1_INLINE` `bool SPL::operator== (const Sequence1< T > &f, const Sequence1< T > &g)`

Test two sequences for equality.

 - `template<class T >`
`SPL_SEQUENCE1_INLINE` `bool SPL::operator!= (const Sequence1< T > &f, const Sequence1< T > &g)`

Test two sequences for inequality.

 - `template<class T >`
`SPL_SEQUENCE1_INLINE` `bool SPL::approxEqual (const Sequence1< T > &f, const Sequence1< T > &g, T threshold=1e-9)`

Test two sequences for approximate equality.

 - `template<class T >`
`Sequence1< T > SPL::subsequence (const Sequence1< T > &f, int startInd, int size)`

Extract a subsequence from a sequence.

 - `template<class T >`
`SPL_SEQUENCE1_INLINE` `Sequence1< T > SPL::translate (const Sequence1< T > &f, int delta)`

Translate a sequence by the specified amount.

 - `template<class T >`
`Sequence1< T > SPL::convolve (const Sequence1< T > &f, const Sequence1< T > &g, int mode=Convolve↵ Mode::full)`

Compute the convolution of two sequences.

 - `template<class T >`
`Sequence1< T > SPL::downsample (const Sequence1< T > &f, int factor)`

Downsample a sequence by the specified factor.

 - `template<class T >`
`Sequence1< T > SPL::upsample (const Sequence1< T > &f, int factor, int pad=0)`

Upsample a sequence by the specified factor.

 - `template<class T >`
`Array1< Sequence1< T > > SPL::polyphaseSplit (const Sequence1< T > &seq, int type, int numPhases)`

Split a sequence into its polyphase components.

 - `template<class T >`
`Sequence1< T > SPL::polyphaseJoin (const Array1< Sequence1< T > > &comps, int type)`

Reassemble a sequence from its polyphase components.

 - `SPL::Sequence1< T >::Sequence1 ()`

The default constructor.

 - `SPL::Sequence1< T >::Sequence1 (int startInd, int size)`

Construct a sequence with the specified start index and size.

- `SPL::Sequence1< T >::Sequence1` (int startInd, int size, const T &value)
Construct a sequence with the specified start index and size, with all elements set to the given value.
- `SPL::Sequence1< T >::Sequence1` (const Array1< T > &data)
Create a sequence from an array.
- `SPL::Sequence1< T >::Sequence1` (int startInd, const Array1< T > &data)
Create a sequence from an array using the given starting index.
- `SPL::Sequence1< T >::Sequence1` (const Sequence1 &f)
The copy constructor.
- `SPL::Sequence1< T >::~~Sequence1` ()
The destructor.
- `Sequence1 & SPL::Sequence1< T >::operator=` (const Sequence1 &f)
The assignment operator.
- `Sequence1 & SPL::Sequence1< T >::operator+=` (const Sequence1 &f)
Add another sequence to this one.
- `Sequence1 & SPL::Sequence1< T >::operator-=` (const Sequence1 &f)
Subtract another sequence from this one.
- `Sequence1 & SPL::Sequence1< T >::operator*=` (const Sequence1 &f)
Multiply elementwise this sequence by another one.
- `Sequence1 & SPL::Sequence1< T >::operator/=` (const Sequence1 &f)
Divide elementwise this sequence by another one.
- `Sequence1 & SPL::Sequence1< T >::operator+=` (const T &value)
Add a value to each element of this sequence.
- `Sequence1 & SPL::Sequence1< T >::operator-=` (const T &value)
Subtract a value from each element of this sequence.
- `Sequence1 & SPL::Sequence1< T >::operator*=` (const T &value)
Multiply each element of this sequence by the specified value.
- `Sequence1 & SPL::Sequence1< T >::operator/=` (const T &value)
Divide each element of the sequence by the given value.
- `int SPL::Sequence1< T >::getStartInd` () const
Get the start index for the sequence.
- `int SPL::Sequence1< T >::getEndInd` () const
Get the end index for the sequence.
- `int SPL::Sequence1< T >::getSize` () const
Get the length of the sequence.
- `bool SPL::Sequence1< T >::isShared` () const
Is the array for this sequence shared with another array?
- `const T & SPL::Sequence1< T >::operator()` (int i) const
Get the specified element in the sequence.
- `T & SPL::Sequence1< T >::operator()` (int i)
Get the specified element in the sequence.
- `ConstIterator SPL::Sequence1< T >::begin` () const
Get an iterator referencing the first element in the sequence.
- `ConstIterator SPL::Sequence1< T >::end` () const
Get an iterator referencing just after the last element in the sequence.
- `Iterator SPL::Sequence1< T >::begin` ()
Get an iterator referencing the first element in the sequence.
- `Iterator SPL::Sequence1< T >::end` ()

Get an iterator referencing just after the last element in the sequence.

- `T SPL::Sequence1< T >::min () const`

Get the minimum element in the sequence.

- `T SPL::Sequence1< T >::max () const`

Get the maximum element in the sequence.

- `T SPL::Sequence1< T >::sum () const`

Get the sum of the elements in the sequence.

- `void SPL::Sequence1< T >::swapArray (Array1< T > &data)`

Swap the data for the underlying array and the specified array.

- `void SPL::Sequence1< T >::fill (const T &value)`

Set all of the elements in the sequence to the specified value.

- `Array1< T > SPL::Sequence1< T >::getArray () const`

Get a copy of the underlying array.

- `Sequence1 & SPL::Sequence1< T >::translate (int delta)`

Translate (i.e., shift) a sequence by the specified displacement.

9.4.1 Detailed Description

One-dimensional sequences.

9.4.2 Typedef Documentation

9.4.2.1 `typedef Sequence1<int> SPL::IntSequence1`

Integer sequence.

9.4.2.2 `typedef Sequence1<double> SPL::RealSequence1`

Real sequence.

9.4.3 Function Documentation

9.4.3.1 `template<class T> Sequence1<T> SPL::add (const Sequence1< T > &f, const Sequence1< T > &g)`

Compute the sum of two sequences with potentially differing domains.

Effects: The sum of the sequences `f` and `g` is computed. The domain of the sum is taken to be the smallest domain that contains the domains of both of the sequences being summed.

Returns: The sum is returned.

9.4.3.2 `template<class T> SPL_SEQUENCE1_INLINE bool SPL::approxEqual (const Sequence1< T > &f, const Sequence1< T > &g, T threshold = 1e-9)`

Test two sequences for approximate equality.

9.4.3.3 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T>::ConstIterator SPL::Sequence1<T>::begin ()`
`const`

Get an iterator referencing the first element in the sequence.

Returns: An iterator referencing the first element in the sequence (i.e., the element with index `getStartInd()`) is returned.

9.4.3.4 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T>::Iterator SPL::Sequence1<T>::begin ()`

Get an iterator referencing the first element in the sequence.

Returns: An iterator referencing the first element in the sequence (i.e., the element with index `getStartInd()`) is returned.

9.4.3.5 `template<class T> Sequence1<T> SPL::convolve (const Sequence1<T> & f, const Sequence1<T> & g, int mode = ConvolveMode::full)`

Compute the convolution of two sequences.

Effects: The convolution of the sequences `f` and `g` is computed. The domain of the resulting sequence (as well as how boundaries are handled) depends on the convolution mode `mode`. The "full" mode is the same as the "full" mode in MATLAB. The "sameDomainZeroExt" mode is the same as the "same" mode in MATLAB.

Returns: A sequence containing the convolution result is returned.

9.4.3.6 `template<class T> Sequence1<T> SPL::downsample (const Sequence1<T> & f, int factor)`

Downsample a sequence by the specified factor.

Effects: The sequence `f` is downsampled by the factor `factor`.

Returns: The downsampled sequence is returned.

9.4.3.7 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T>::ConstIterator SPL::Sequence1<T>::end ()`
`const`

Get an iterator referencing just after the last element in the sequence.

Returns: An iterator for the end of the sequence (i.e., one past the last element) is returned.

9.4.3.8 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T>::Iterator SPL::Sequence1<T>::end ()`

Get an iterator referencing just after the last element in the sequence.

Returns: An iterator for the end of the sequence (i.e., one past the last element) is returned.

9.4.3.9 `template<class T> SPL_SEQUENCE1_INLINE void SPL::Sequence1<T>::fill (const T & value)`

Set all of the elements in the sequence to the specified value.

Effects: Each elements in the sequence is set to the value `value`.

9.4.3.10 `template<class T> SPL_SEQUENCE1_INLINE Array1< T> SPL::Sequence1< T>::getArray () const`

Get a copy of the underlying array.

9.4.3.11 `template<class T> SPL_SEQUENCE1_INLINE int SPL::Sequence1< T>::getEndInd () const`

Get the end index for the sequence.

Returns: The ending index (i.e., one past the last valid index) is returned.

9.4.3.12 `template<class T> SPL_SEQUENCE1_INLINE int SPL::Sequence1< T>::getSize () const`

Get the length of the sequence.

Returns: The number of elements in the sequence is returned. This value is equivalent to [getEndInd\(\)](#) - [getStartInd\(\)](#).

9.4.3.13 `template<class T> SPL_SEQUENCE1_INLINE int SPL::Sequence1< T>::getStartInd () const`

Get the start index for the sequence.

Returns: The starting index for the sequence is returned.

9.4.3.14 `template<class T> SPL_SEQUENCE1_INLINE bool SPL::Sequence1< T>::isShared () const`

Is the array for this sequence shared with another array?

9.4.3.15 `template<class T> SPL_SEQUENCE1_INLINE T SPL::Sequence1< T>::max () const`

Get the maximum element in the sequence.

The sequence must contain at least one element.

9.4.3.16 `template<class T> SPL_SEQUENCE1_INLINE T SPL::Sequence1< T>::min () const`

Get the minimum element in the sequence.

The sequence must contain at least one element.

9.4.3.17 `template<class T> SPL_SEQUENCE1_INLINE bool SPL::operator!= (const Sequence1< T> & f, const Sequence1< T> & g)`

Test two sequences for inequality.

9.4.3.18 `template<class T> SPL_SEQUENCE1_INLINE const T & SPL::Sequence1< T>::operator() (int i) const`

Get the specified element in the sequence.

Returns: A reference to the i-th element in the sequence is returned.

9.4.3.19 `template<class T> SPL_SEQUENCE1_INLINE T & SPL::Sequence1<T>::operator()(int i)`

Get the specified element in the sequence.

Returns: A reference to the i-th element in the sequence is returned.

9.4.3.20 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::operator* (const Sequence1<T> & f, const Sequence1<T> & g)`

Compute the (element-wise) product of two sequences.

Returns: The element-wise product of the sequences f and g is returned. Both sequences must have the same domain.

9.4.3.21 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::operator* (const T & a, const Sequence1<T> & f)`

Compute a scalar multiple of a sequence.

Returns: The sequence f multiplied by the value a is returned.

9.4.3.22 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::operator* (const Sequence1<T> & f, const T & a)`

Compute a scalar multiple of a sequence.

Returns: The sequence f multiplied by the value a is returned.

9.4.3.23 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> & SPL::Sequence1<T>::operator*= (const Sequence1<T> & f)`

Multiply elementwise this sequence by another one.

Effects: This sequence is multiplied (element-wise) by the sequence f. Both sequences must have the same domain.

9.4.3.24 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> & SPL::Sequence1<T>::operator*= (const T & value)`

Multiply each element of this sequence by the specified value.

Effects: This sequence is multiplied by the element value value.

9.4.3.25 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::operator+ (const Sequence1<T> & f, const Sequence1<T> & g)`

Compute the sum of two sequences.

Returns: The sum of the sequences f and g is returned. Both sequences must have the same domain.

9.4.3.26 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::operator+ (const T & a, const Sequence1<T> & f)`

Add a value to a sequence.

Returns: The sequence f with a added to each of its elements is returned.

9.4.3.27 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::operator+ (const Sequence1< T > & f,
const T & a)`

Add a value to a sequence.

Returns: The sequence f with a added to each of its elements is returned.

9.4.3.28 `template<class T> SPL_SEQUENCE1_INLINE Sequence1< T > & SPL::Sequence1< T >::operator+= (const
Sequence1< T > & f)`

Add another sequence to this one.

Effects: The sequence f is added to this sequence. Both sequences must have the same domain.

9.4.3.29 `template<class T> SPL_SEQUENCE1_INLINE Sequence1< T > & SPL::Sequence1< T >::operator+= (const T
& value)`

Add a value to each element of this sequence.

Effects: The value $value$ is added to each element of the sequence.

9.4.3.30 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::operator- (const Sequence1< T > & f,
const Sequence1< T > & g)`

Compute the difference of two sequences.

Returns: The difference between the sequence f and sequence g (i.e., $f - g$) is returned. Both sequences must have the same domain.

9.4.3.31 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::operator- (const Sequence1< T > & f,
const T & a)`

Subtract a value from a sequence.

Returns: The sequence f with a subtracted from each of its elements is returned.

9.4.3.32 `template<class T> SPL_SEQUENCE1_INLINE Sequence1< T > & SPL::Sequence1< T >::operator-= (const
Sequence1< T > & f)`

Subtract another sequence from this one.

Effects: The sequence f is subtracted from this sequence. Both sequences must have the same domain.

9.4.3.33 `template<class T> SPL_SEQUENCE1_INLINE Sequence1< T > & SPL::Sequence1< T >::operator-= (const T
& value)`

Subtract a value from each element of this sequence.

Effects: The value $value$ is subtracted from each element of the sequence.

9.4.3.34 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::operator/ (const Sequence1< T> & f, const Sequence1< T> & g)`

Compute the (element-wise) quotient of two sequences.

Returns: The element-wise quotient of the sequences f and g is returned. Both sequences must have the same domain.

9.4.3.35 `template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::operator/ (const Sequence1< T> & f, const T & a)`

Divide a sequence by a scalar.

Returns: The sequence f divided by the value a is returned.

9.4.3.36 `template<class T> SPL_SEQUENCE1_INLINE Sequence1< T> & SPL::Sequence1< T>::operator/= (const Sequence1< T> & f)`

Divide elementwise this sequence by another one.

Effects: This sequence is divided (element-wise) by the sequence f. Both sequences must have the same domain.

9.4.3.37 `template<class T> SPL_SEQUENCE1_INLINE Sequence1< T> & SPL::Sequence1< T>::operator/= (const T & value)`

Divide each element of the sequence by the given value.

Effects: Each element of the sequence is divided by the value value.

9.4.3.38 `template<class T> std::ostream& SPL::operator<< (std::ostream & out, const Sequence1< T> & f)`

Output a sequence to a stream.

Effects: The sequence f is written to the output stream out. The output consists of the following information in order: 1) the starting index of the sequence 2) the size of the sequence 3) the elements of the sequence in increasing order of index

Returns: A reference to the stream out is returned.

9.4.3.39 `template<class T> SPL_SEQUENCE1_INLINE Sequence1< T> & SPL::Sequence1< T>::operator= (const Sequence1< T> & f)`

The assignment operator.

9.4.3.40 `template<class T> SPL_SEQUENCE1_INLINE bool SPL::operator== (const Sequence1< T> & f, const Sequence1< T> & g)`

Test two sequences for equality.

In order for two sequences to be deemed equal, they must be defined on the same domain and have their element values match everywhere in this domain.

9.4.3.41 `template<class T> std::istream& SPL::operator>> (std::istream & in, Sequence1< T > & f)`

Input a sequence from a stream.

Effects: The sequence *f* is read from the input stream *in*. The data is read in a format consistent with that used by `operator<<`.

Returns: A reference to the stream *in* is returned.

9.4.3.42 `template<class T> Sequence1<T> SPL::polyphaseJoin (const Array1< Sequence1< T > > & comps, int type)`

Reassemble a sequence from its polyphase components.

Effects: A sequence is recomposed from its polyphase components *comps*. A polyphase decomposition of type *type* is assumed.

Returns: The recomposed sequence is returned.

9.4.3.43 `template<class T> Array1<Sequence1<T> > SPL::polyphaseSplit (const Sequence1< T > & seq, int type, int numPhases)`

Split a sequence into its polyphase components.

Effects: The polyphase decomposition of the sequence *seq* is computed. In particular, the polyphase decomposition with *numPhases* phases and type *type* is computed.

Returns: An array containing the polyphase components is returned.

9.4.3.44 `template<class T> SPL_SEQUENCE1_INLINE SPL::Sequence1< T >::Sequence1 ()`

The default constructor.

9.4.3.45 `template<class T> SPL_SEQUENCE1_INLINE SPL::Sequence1< T >::Sequence1 (int startInd, int size)`

Construct a sequence with the specified start index and size.

Effects: A sequence with a starting index of *startInd* and size *size* is created. The elements in the sequence are default constructed!

9.4.3.46 `template<class T> SPL_SEQUENCE1_INLINE SPL::Sequence1< T >::Sequence1 (int startInd, int size, const T & value)`

Construct a sequence with the specified start index and size, with all elements set to the given value.

Effects: A sequence with a starting index of *startInd* and size *size* is created, with all elements initialized to the value *value*.

9.4.3.47 `template<class T> SPL_SEQUENCE1_INLINE SPL::Sequence1< T >::Sequence1 (const Sequence1< T > & f)`

The copy constructor.

```
9.4.3.48 template<class T> SPL_SEQUENCE1_INLINE SPL::Sequence1< T>::Sequence1 ( const Array1< T> & data )
```

Create a sequence from an array.

```
9.4.3.49 template<class T> SPL_SEQUENCE1_INLINE SPL::Sequence1< T>::Sequence1 ( int startInd, const Array1< T> & data )
```

Create a sequence from an array using the given starting index.

```
9.4.3.50 template<class T> Sequence1<T> SPL::subsequence ( const Sequence1< T> & f, int startInd, int size )
```

Extract a subsequence from a sequence.

Effects: The subsequence with start index startInd and size size is extracted from the sequence f.

Returns: The extracted subsequence is returned.

```
9.4.3.51 template<class T> SPL_SEQUENCE1_INLINE T SPL::Sequence1< T>::sum ( ) const
```

Get the sum of the elements in the sequence.

```
9.4.3.52 template<class T> SPL_SEQUENCE1_INLINE void SPL::Sequence1< T>::swapArray ( Array1< T> & data )
```

Swap the data for the underlying array and the specified array.

```
9.4.3.53 template<class T> SPL_SEQUENCE1_INLINE Sequence1< T> & SPL::Sequence1< T>::translate ( int delta )
```

Translate (i.e., shift) a sequence by the specified displacement.

```
9.4.3.54 template<class T> SPL_SEQUENCE1_INLINE Sequence1<T> SPL::translate ( const Sequence1< T> & f, int delta )
```

Translate a sequence by the specified amount.

Effects: The sequence f is translated (i.e., time shifted) by i. For example, if the sequence f is defined on the domain a, a+1, ..., b, then the translated sequence will be defined on the domain a+i, a+i+1, ..., b+i.

Returns: The translated sequence is returned.

```
9.4.3.55 template<class T> Sequence1<T> SPL::upsample ( const Sequence1< T> & f, int factor, int pad = 0 )
```

Upsample a sequence by the specified factor.

Effects: The sequence f is upsampled by the factor factor. If pad is zero, new samples will only be added between the first and last sample. Up to (factor - 1) extra new samples can be added at the end of the new sequence, by specifying a nonzero value for pad. The default is no padding.

Returns: The upsampled sequence is returned.

9.4.3.56 `template<class T> SPL_SEQUENCE1_INLINE SPL::Sequence1<T>::~~Sequence1 ()`

The destructor.

9.5 Two-Dimensional Sequences

Two-dimensional sequences.

Classes

- class [SPL::Sequence2< T >](#)
A two-dimensional sequence class with lazy copying and reference counting.

Typedefs

- typedef Sequence2< double > [SPL::RealSequence2](#)
Real sequence.
- typedef Sequence2< int > [SPL::IntSequence2](#)
Integer sequence.

Functions

- template<class T >
std::ostream & [SPL::operator<<](#) (std::ostream &out, const Sequence2< T > &f)
Output a sequence to a stream.
- template<class T >
std::istream & [SPL::operator>>](#) (std::istream &in, Sequence2< T > &f)
Input a sequence from a stream.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator+](#) (const Sequence2< T > &f, const Sequence2< T > &g)
Compute the sum of two sequences.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator-](#) (const Sequence2< T > &f, const Sequence2< T > &g)
Compute the difference of two sequences.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator*](#) (const Sequence2< T > &f, const Sequence2< T > &g)
Compute the (element-wise) product of two sequences.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator/](#) (const Sequence2< T > &f, const Sequence2< T > &g)
Compute the (element-wise) quotient of two sequences.
- template<class T >
Sequence2< T > [SPL::add](#) (const Sequence2< T > &f, const Sequence2< T > &g)
Compute the sum of two sequences with potentially differing domains.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator+](#) (const T &value, const Sequence2< T > &f)
Add a value to a sequence.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator+](#) (const Sequence2< T > &f, const T &value)

Add a value to a sequence.

- `template<class T >`
`SPL_SEQUENCE2_INLINE` `Sequence2< T > SPL::operator-` (`const Sequence2< T > &f`, `const T &value`)

Subtract a value from a sequence.

- `template<class T >`
`SPL_SEQUENCE2_INLINE` `Sequence2< T > SPL::operator*` (`const T &value`, `const Sequence2< T > &f`)

Compute a scalar multiple of a sequence.

- `template<class T >`
`SPL_SEQUENCE2_INLINE` `Sequence2< T > SPL::operator*` (`const Sequence2< T > &f`, `const T &value`)

Compute a scalar multiple of a sequence.

- `template<class T >`
`SPL_SEQUENCE2_INLINE` `Sequence2< T > SPL::operator/` (`const Sequence2< T > &f`, `const T &value`)

Divide a sequence by a scalar.

- `template<class T >`
`bool SPL::operator==` (`const Sequence2< T > &f`, `const Sequence2< T > &g`)

Test two sequences for equality.

- `template<class T >`
`SPL_SEQUENCE2_INLINE` `bool SPL::operator!=` (`const Sequence2< T > &f`, `const Sequence2< T > &g`)

Test two sequences for inequality.

- `template<class T >`
`SPL_SEQUENCE2_INLINE` `bool SPL::approxEqual` (`const Sequence2< T > &f`, `const Sequence2< T > &g`, `T threshold=1e-9`)

Test two sequences for approximate equality.

- `template<class T >`
`Sequence2< T > SPL::subsequence` (`const Sequence2< T > &f`, `int startX`, `int startY`, `int width`, `int height`)

Extract a subsequence from a sequence.

- `template<class T >`
`SPL_SEQUENCE2_INLINE` `Sequence2< T > SPL::translate` (`const Sequence2< T > &f`, `int deltaX`, `int deltaY`)

Translate a sequence by the specified amount.

- `template<class T >`
`Sequence2< T > SPL::convolve` (`const Sequence2< T > &f`, `const Sequence2< T > &g`, `int mode`)

Compute the convolution of two sequences.

- `template<class T >`
`Sequence2< T > SPL::convolveSeparable` (`const Sequence2< T > &f`, `const Sequence1< T > &horzFilt`, `const Sequence1< T > &vertFilt`, `int mode=ConvolveMode::full`)

Compute the convolution of a sequence with two 1-D filters (i.e., convolution with a separable filter).

- `template<class T >`
`Sequence2< T > SPL::downsample` (`const Sequence2< T > &f`, `int factorX`, `int factorY`)

Downsample a sequence in each of the horizontal and vertical directions by the specified factors.

- `template<class T >`
`Sequence2< T > SPL::upsample` (`const Sequence2< T > &f`, `int factorX`, `int factorY`)

Upsample a sequence in each of the horizontal and vertical directions by the specified factors.

- `template<class T >`
`Sequence2< T > SPL::upsample` (`const Sequence2< T > &f`, `int factorX`, `int factorY`, `int padX`, `int padY`)

Upsample a sequence in each of the horizontal and vertical directions by the specified factors.

- `template<class T >`
`Array2< Sequence2< T > > SPL::polyphaseSplit` (`const Sequence2< T > &seq`, `int typeX`, `int numPhasesX`, `int typeY`, `int numPhasesY`)

Split a sequence into its polyphase components.

- `template<class T >`
`Sequence2< T > SPL::polyphaseJoin` (const Array2< Sequence2< T > > &comps, int typeX, int typeY)
Reassemble a sequence from its polyphase components.
- `SPL::Sequence2< T >::Sequence2` ()
The default constructor.
- `SPL::Sequence2< T >::Sequence2` (int startX, int startY, int width, int height)
Construct a sequence with the specified start index and size.
- `SPL::Sequence2< T >::Sequence2` (int startX, int startY, int width, int height, const T &data)
Construct a sequence with the specified start index and size, with all elements set to the given value.
- `SPL::Sequence2< T >::Sequence2` (const Sequence2 &f)
The copy constructor.
- `SPL::Sequence2< T >::Sequence2` (const Array2< T > &data)
Create a sequence from an array.
- `SPL::Sequence2< T >::Sequence2` (int startX, int startY, const Array2< T > &data)
Create a sequence from an array using the given starting index.
- `SPL::Sequence2< T >::~~Sequence2` ()
The destructor.
- `Sequence2 & SPL::Sequence2< T >::operator=` (const Sequence2 &f)
The assignment operator.
- `Sequence2 & SPL::Sequence2< T >::operator+=` (const Sequence2 &f)
Add another sequence to this one.
- `Sequence2 & SPL::Sequence2< T >::operator-=` (const Sequence2 &f)
Subtract another sequence from this one.
- `Sequence2 & SPL::Sequence2< T >::operator*=` (const Sequence2 &f)
Multiply elementwise this sequence by another one.
- `Sequence2 & SPL::Sequence2< T >::operator/=` (const Sequence2 &f)
Divide elementwise this sequence by another one.
- `Sequence2 & SPL::Sequence2< T >::operator+=` (const T &value)
Add a value to each element of this sequence.
- `Sequence2 & SPL::Sequence2< T >::operator-=` (const T &value)
Subtract a value from each element of this sequence.
- `Sequence2 & SPL::Sequence2< T >::operator*=` (const T &value)
Multiply each element of this sequence by the specified value.
- `Sequence2 & SPL::Sequence2< T >::operator/=` (const T &value)
Divide each element of the sequence by the given value.
- `int SPL::Sequence2< T >::getStartX` () const
Get the x-coordinate of the start index for the sequence.
- `int SPL::Sequence2< T >::getStartY` () const
Get the y-coordinate of the start index for the sequence.
- `int SPL::Sequence2< T >::getEndX` () const
Get the x-coordinate of the end index for the sequence.
- `int SPL::Sequence2< T >::getEndY` () const
Get the y-coordinate of the end index for the sequence.
- `int SPL::Sequence2< T >::getWidth` () const
Get the width of the sequence.
- `int SPL::Sequence2< T >::getHeight` () const
Get the height of the sequence.

- `int SPL::Sequence2< T >::getSize () const`
Get the number of elements in the sequence.
- `bool SPL::Sequence2< T >::isShared () const`
Is the array for this sequence shared with another array?
- `const T & SPL::Sequence2< T >::operator() (int x, int y) const`
Get a const reference to the specified element in the sequence.
- `T & SPL::Sequence2< T >::operator() (int x, int y)`
Get a mutable reference to the specified element in the sequence.
- `ConstIterator SPL::Sequence2< T >::begin () const`
Get a const iterator for the first element in the sequence.
- `Iterator SPL::Sequence2< T >::begin ()`
Get a mutable iterator for the first element in the sequence.
- `ConstIterator SPL::Sequence2< T >::end () const`
Get a const iterator for one past the last element in the sequence.
- `Iterator SPL::Sequence2< T >::end ()`
Get a mutable iterator for one past the last element in the sequence.
- `ConstXIterator SPL::Sequence2< T >::rowBegin (int y) const`
Get a const iterator for the first element in the specified row of the sequence.
- `XIterator SPL::Sequence2< T >::rowBegin (int y)`
Get a mutable iterator for the first element in the specified row of the sequence.
- `ConstXIterator SPL::Sequence2< T >::rowEnd (int y) const`
Get a const iterator for one past the end in the specified row of the sequence.
- `XIterator SPL::Sequence2< T >::rowEnd (int y)`
Get a mutable iterator for one past the end in the specified row of the sequence.
- `ConstYIterator SPL::Sequence2< T >::colBegin (int x) const`
Get a const iterator for the first element in the specified column of the sequence.
- `YIterator SPL::Sequence2< T >::colBegin (int x)`
Get a mutable iterator for the first element in the specified column of the sequence.
- `ConstYIterator SPL::Sequence2< T >::colEnd (int x) const`
Get a const iterator for one past the end in the specified column of the sequence.
- `YIterator SPL::Sequence2< T >::colEnd (int x)`
Get a mutable iterator for one past the end in the specified column of the sequence.
- `T SPL::Sequence2< T >::min () const`
Get the minimum element in the sequence.
- `T SPL::Sequence2< T >::max () const`
Get the maximum element in the sequence.
- `T SPL::Sequence2< T >::sum () const`
Get the sum of the elements in the sequence.
- `std::ostream & SPL::Sequence2< T >::output (std::ostream &out, int fieldWidth) const`
Output a sequence to the specified stream using the given field width for each sequence element.
- `void SPL::Sequence2< T >::fill (const T &value)`
Get a copy of the underlying array.
- `void SPL::Sequence2< T >::swapArray (Array2< T > &data)`
Swap the data for the underlying array and the specified array.
- `Array2< T > SPL::Sequence2< T >::getArray () const`
Get a copy of the underlying array.
- `Sequence2 & SPL::Sequence2< T >::translate (int x, int y)`
Translate (i.e., shift) a sequence by the specified displacement.

9.5.1 Detailed Description

Two-dimensional sequences.

9.5.2 Typedef Documentation

9.5.2.1 `typedef Sequence2<int> SPL::IntSequence2`

Integer sequence.

9.5.2.2 `typedef Sequence2<double> SPL::RealSequence2`

Real sequence.

9.5.3 Function Documentation

9.5.3.1 `template<class T> Sequence2<T> SPL::add (const Sequence2< T > & f, const Sequence2< T > & g)`

Compute the sum of two sequences with potentially differing domains.

Effects: The sum of the sequences f and g is computed. The domain of the sum is taken to be the smallest domain that contains the domains of both of the sequences being summed.

Returns: The sum is returned.

9.5.3.2 `template<class T> SPL_SEQUENCE2_INLINE bool SPL::approxEqual (const Sequence2< T > & f, const Sequence2< T > & g, T threshold = 1e-9)`

Test two sequences for approximate equality.

9.5.3.3 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T >::ConstIterator SPL::Sequence2< T >::begin () const`

Get a const iterator for the first element in the sequence.

9.5.3.4 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T >::Iterator SPL::Sequence2< T >::begin ()`

Get a mutable iterator for the first element in the sequence.

9.5.3.5 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T >::ConstYIterator SPL::Sequence2< T >::colBegin (int x) const`

Get a const iterator for the first element in the specified column of the sequence.

9.5.3.6 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T >::YIterator SPL::Sequence2< T >::colBegin (int x)`

Get a mutable iterator for the first element in the specified column of the sequence.

9.5.3.7 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T >::ConstYIterator SPL::Sequence2< T >::colEnd (int x) const`

Get a const iterator for one past the end in the specified column of the sequence.

9.5.3.8 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T >::YIterator SPL::Sequence2< T >::colEnd (int x)`

Get a mutable iterator for one past the end in the specified column of the sequence.

9.5.3.9 `template<class T> Sequence2<T> SPL::convolve (const Sequence2< T > & f, const Sequence2< T > & g, int mode)`

Compute the convolution of two sequences.

9.5.3.10 `template<class T> Sequence2<T> SPL::convolveSeparable (const Sequence2< T > & f, const Sequence1< T > & horzFilt, const Sequence1< T > & vertFilt, int mode = ConvolveMode::full)`

Compute the convolution of a sequence with two 1-D filters (i.e., convolution with a separable filter).

9.5.3.11 `template<class T> Sequence2<T> SPL::downsample (const Sequence2< T > & f, int factorX, int factorY)`

Downsample a sequence in each of the horizontal and vertical directions by the specified factors.

9.5.3.12 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T >::ConstIterator SPL::Sequence2< T >::end () const`

Get a const iterator for one past the last element in the sequence.

9.5.3.13 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T >::Iterator SPL::Sequence2< T >::end ()`

Get a mutable iterator for one past the last element in the sequence.

9.5.3.14 `template<class T> SPL_SEQUENCE2_INLINE void SPL::Sequence2< T >::fill (const T & value)`

Get a copy of the underlying array.

9.5.3.15 `template<class T> SPL_SEQUENCE2_INLINE Array2< T > SPL::Sequence2< T >::getArray () const`

Get a copy of the underlying array.

9.5.3.16 `template<class T> SPL_SEQUENCE2_INLINE int SPL::Sequence2< T >::getEndX () const`

Get the x-coordinate of the end index for the sequence.

9.5.3.17 `template<class T> SPL_SEQUENCE2_INLINE int SPL::Sequence2< T>::getEndY () const`

Get the y-coordinate of the end index for the sequence.

9.5.3.18 `template<class T> SPL_SEQUENCE2_INLINE int SPL::Sequence2< T>::getHeight () const`

Get the height of the sequence.

9.5.3.19 `template<class T> SPL_SEQUENCE2_INLINE int SPL::Sequence2< T>::getSize () const`

Get the number of elements in the sequence.

9.5.3.20 `template<class T> SPL_SEQUENCE2_INLINE int SPL::Sequence2< T>::getStartX () const`

Get the x-coordinate of the start index for the sequence.

9.5.3.21 `template<class T> SPL_SEQUENCE2_INLINE int SPL::Sequence2< T>::getStartY () const`

Get the y-coordinate of the start index for the sequence.

9.5.3.22 `template<class T> SPL_SEQUENCE2_INLINE int SPL::Sequence2< T>::getWidth () const`

Get the width of the sequence.

9.5.3.23 `template<class T> SPL_SEQUENCE2_INLINE bool SPL::Sequence2< T>::isShared () const`

Is the array for this sequence shared with another array?

9.5.3.24 `template<class T> SPL_SEQUENCE2_INLINE T SPL::Sequence2< T>::max () const`

Get the maximum element in the sequence.

The sequence must contain at least one element.

9.5.3.25 `template<class T> SPL_SEQUENCE2_INLINE T SPL::Sequence2< T>::min () const`

Get the minimum element in the sequence.

The sequence must contain at least one element.

9.5.3.26 `template<class T> SPL_SEQUENCE2_INLINE bool SPL::operator!= (const Sequence2< T> & f, const Sequence2< T> & g)`

Test two sequences for inequality.

9.5.3.27 `template<class T> SPL_SEQUENCE2_INLINE T & SPL::Sequence2< T>::operator()(int x, int y)`

Get a mutable reference to the specified element in the sequence.

9.5.3.28 `template<class T> SPL_SEQUENCE2_INLINE const T & SPL::Sequence2< T >::operator()(int x, int y) const`

Get a const reference to the specified element in the sequence.

9.5.3.29 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::operator* (const Sequence2< T > & f, const Sequence2< T > & g)`

Compute the (element-wise) product of two sequences.

Returns: The element-wise product of the sequences f and g is returned. Both sequences must have the same domain.

9.5.3.30 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::operator* (const T & value, const Sequence2< T > & f)`

Compute a scalar multiple of a sequence.

Returns: The sequence f multiplied by the value a is returned.

9.5.3.31 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::operator* (const Sequence2< T > & f, const T & value)`

Compute a scalar multiple of a sequence.

Returns: The sequence f multiplied by the value a is returned.

9.5.3.32 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T > & SPL::Sequence2< T >::operator*= (const Sequence2< T > & f)`

Multiply elementwise this sequence by another one.

9.5.3.33 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T > & SPL::Sequence2< T >::operator*= (const T & value)`

Multiply each element of this sequence by the specified value.

9.5.3.34 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::operator+ (const Sequence2< T > & f, const Sequence2< T > & g)`

Compute the sum of two sequences.

Returns: The sum of the sequences f and g is returned. Both sequences must have the same domain.

9.5.3.35 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::operator+ (const T & value, const Sequence2< T > & f)`

Add a value to a sequence.

Returns: The sequence f with a added to each of its elements is returned.

9.5.3.36 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::operator+ (const Sequence2< T> & f, const T & value)`

Add a value to a sequence.

Returns: The sequence f with a added to each of its elements is returned.

9.5.3.37 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T> & SPL::Sequence2< T>::operator+= (const Sequence2< T> & f)`

Add another sequence to this one.

9.5.3.38 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T> & SPL::Sequence2< T>::operator+= (const T & value)`

Add a value to each element of this sequence.

9.5.3.39 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::operator- (const Sequence2< T> & f, const Sequence2< T> & g)`

Compute the difference of two sequences.

Returns: The difference between the sequence f and sequence g (i.e., $f - g$) is returned. Both sequences must have the same domain.

9.5.3.40 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::operator- (const Sequence2< T> & f, const T & value)`

Subtract a value from a sequence.

Returns: The sequence f with a subtracted from each of its elements is returned.

9.5.3.41 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T> & SPL::Sequence2< T>::operator-= (const Sequence2< T> & f)`

Subtract another sequence from this one.

9.5.3.42 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T> & SPL::Sequence2< T>::operator-= (const T & value)`

Subtract a value from each element of this sequence.

9.5.3.43 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::operator/ (const Sequence2< T> & f, const Sequence2< T> & g)`

Compute the (element-wise) quotient of two sequences.

Returns: The element-wise quotient of the sequences f and g is returned. Both sequences must have the same domain.

9.5.3.44 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::operator/ (const Sequence2< T> & f, const T & value)`

Divide a sequence by a scalar.

Returns: The sequence f divided by the value a is returned.

9.5.3.45 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T> & SPL::Sequence2< T>::operator/= (const Sequence2< T> & f)`

Divide elementwise this sequence by another one.

9.5.3.46 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T> & SPL::Sequence2< T>::operator/= (const T & value)`

Divide each element of the sequence by the given value.

9.5.3.47 `template<class T> std::ostream& SPL::operator<< (std::ostream & out, const Sequence2< T> & f)`

Output a sequence to a stream.

Effects: The sequence f is written to the output stream out. The output consists of the following information in order: 1) the x-coordinate of the start index of the sequence 2) the y-coordinate of the start index of the sequence 3) the width of the sequence 4) the height of the sequence 5) the elements of the sequence in row-major order

Returns: A reference to the stream out is returned.

9.5.3.48 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T> & SPL::Sequence2< T>::operator= (const Sequence2< T> & f)`

The assignment operator.

9.5.3.49 `template<class T> bool SPL::operator==(const Sequence2< T> & f, const Sequence2< T> & g)`

Test two sequences for equality.

In order for two sequences to be deemed equal, they must be defined on the same domain and have their element values match everywhere in this domain.

9.5.3.50 `template<class T> std::istream& SPL::operator>> (std::istream & in, Sequence2< T> & f)`

Input a sequence from a stream.

Effects: The sequence f is read from the input stream in. The data is read in a format consistent with that used by operator<<.

Returns: A reference to the stream in is returned.

9.5.3.51 `template<class T> std::ostream & SPL::Sequence2< T>::output (std::ostream & out, int fieldWidth) const`

Output a sequence to the specified stream using the given field width for each sequence element.

9.5.3.52 `template<class T> Sequence2<T> SPL::polyphaseJoin (const Array2< Sequence2< T> > & comps, int typeX, int typeY)`

Reassemble a sequence from its polyphase components.

Effects: A sequence is recomposed from its polyphase components comps. A polyphase decomposition of type type is assumed.

Returns: The recomposed sequence is returned.

9.5.3.53 `template<class T> Array2<Sequence2<T> > SPL::polyphaseSplit (const Sequence2< T> & seq, int typeX, int numPhasesX, int typeY, int numPhasesY)`

Split a sequence into its polyphase components.

Effects: The polyphase decomposition of the sequence seq is computed. In particular, the polyphase decomposition with numPhase phases and type type is computed.

Returns: An array containing the polyphase components is returned.

9.5.3.54 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T>::ConstXIterator SPL::Sequence2< T>::rowBegin (int y) const`

Get a const iterator for the first element in the specified row of the sequence.

9.5.3.55 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T>::XIterator SPL::Sequence2< T>::rowBegin (int y)`

Get a mutable iterator for the first element in the specified row of the sequence.

9.5.3.56 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T>::ConstXIterator SPL::Sequence2< T>::rowEnd (int y) const`

Get a const iterator for one past the end in the specified row of the sequence.

9.5.3.57 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T>::XIterator SPL::Sequence2< T>::rowEnd (int y)`

Get a mutable iterator for one past the end in the specified row of the sequence.

9.5.3.58 `template<class T> SPL_SEQUENCE2_INLINE SPL::Sequence2< T>::Sequence2 ()`

The default constructor.

9.5.3.59 `template<class T> SPL_SEQUENCE2_INLINE SPL::Sequence2< T>::Sequence2 (int startX, int startY, int width, int height)`

Construct a sequence with the specified start index and size.

9.5.3.60 `template<class T> SPL_SEQUENCE2_INLINE SPL::Sequence2< T >::Sequence2 (int startX, int startY, int width, int height, const T & data)`

Construct a sequence with the specified start index and size, with all elements set to the given value.

9.5.3.61 `template<class T> SPL_SEQUENCE2_INLINE SPL::Sequence2< T >::Sequence2 (const Sequence2< T > & f)`

The copy constructor.

9.5.3.62 `template<class T> SPL_SEQUENCE2_INLINE SPL::Sequence2< T >::Sequence2 (const Array2< T > & data)`

Create a sequence from an array.

9.5.3.63 `template<class T> SPL_SEQUENCE2_INLINE SPL::Sequence2< T >::Sequence2 (int startX, int startY, const Array2< T > & data)`

Create a sequence from an array using the given starting index.

9.5.3.64 `template<class T> Sequence2<T> SPL::subsequence (const Sequence2< T > & f, int startX, int startY, int width, int height)`

Extract a subsequence from a sequence.

Effects: The subsequence with start index *startInd* and size *size* is extracted from the sequence *f*.

Returns: The extracted subsequence is returned.

9.5.3.65 `template<class T> SPL_SEQUENCE2_INLINE T SPL::Sequence2< T >::sum () const`

Get the sum of the elements in the sequence.

9.5.3.66 `template<class T> SPL_SEQUENCE2_INLINE void SPL::Sequence2< T >::swapArray (Array2< T > & data)`

Swap the data for the underlying array and the specified array.

9.5.3.67 `template<class T> SPL_SEQUENCE2_INLINE Sequence2< T > & SPL::Sequence2< T >::translate (int x, int y)`

Translate (i.e., shift) a sequence by the specified displacement.

9.5.3.68 `template<class T> SPL_SEQUENCE2_INLINE Sequence2<T> SPL::translate (const Sequence2< T > & f, int deltaX, int deltaY)`

Translate a sequence by the specified amount.

Effects: The sequence *f* is translated by (*deltaX*, *deltaY*).

Returns: The translated sequence is returned.

9.5.3.69 `template<class T> Sequence2<T> SPL::upsample (const Sequence2< T > & f, int factorX, int factorY)`

Upsample a sequence in each of the horizontal and vertical directions by the specified factors.

The following condition should always be true: `downsample(upsample(f, factorX, factorY), factorX, factorY) == f`.

9.5.3.70 `template<class T> Sequence2<T> SPL::upsample (const Sequence2< T > & f, int factorX, int factorY, int padX, int padY)`

Upsample a sequence in each of the horizontal and vertical directions by the specified factors.

The following condition should always be true: `downsample(upsample(f, factorX, factorY), factorX, factorY) == f`.

9.5.3.71 `template<class T> SPL_SEQUENCE2_INLINE SPL::Sequence2< T >::~~Sequence2 ()`

The destructor.

9.6 Bit Stream I/O

Bit streams.

Classes

- class `SPL::BitStream`
A common base class for the input and output bit stream classes.
- class `SPL::InputBitStream`
Input bit stream class.
- class `SPL::OutputBitStream`
Output bit stream class.

Functions

- bool `SPL::BitStream::isOk ()` const
Test if the bitstream is in an okay (i.e., non-error) state.
- bool `SPL::BitStream::isEof ()` const
Test if the bitstream has encountered end-of-file (EOF).
- bool `SPL::BitStream::isLimit ()` const
Test if the bitstream has encountered a read/write limit.
- void `SPL::BitStream::setIoState (IoState state)`
Set the I/O state of a bit stream.
- IoState `SPL::BitStream::getIoState ()` const
Get the I/O state of a bit stream.
- void `SPL::BitStream::clearIoStateBits (IoState state=allIoBits)`
Clear the specified bits in the I/O state of a bit stream.
- void `SPL::BitStream::setIoStateBits (IoState state)`
Set the specified bits in the I/O state of a bit stream.
- void `SPL::InputBitStream::clearReadCount ()`
Set the read count to zero.
- Size `SPL::InputBitStream::getReadCount ()` const
Get the number of bits read from the bit stream so far.
- void `SPL::InputBitStream::setReadLimit (Offset readLimit)`
Specify the maximum allowable number of bits that may be read from the bit stream.
- Offset `SPL::InputBitStream::getReadLimit ()` const
Get the number of bits that still may be read from the bit stream before the read limit is reached.
- Size `SPL::OutputBitStream::getWriteCount ()` const
Get the number of bits written to the bit stream.
- void `SPL::OutputBitStream::clearWriteCount ()`
Clear the count of the number of bits written to the bit stream.
- Offset `SPL::OutputBitStream::getWriteLimit ()` const
Get the number of bits that may still be written to the underlying (character) stream.
- void `SPL::OutputBitStream::setWriteLimit (Offset writeLimit)`
Set the number of bits that may still be written to the bit stream.

9.6.1 Detailed Description

Bit streams.

9.6.2 Function Documentation

9.6.2.1 `void SPL::BitStream::clearIoStateBits (BitStream::IoState mask = allIoBits) [inline]`

Clear the specified bits in the I/O state of a bit stream.

If no parameter is provided, all bits are cleared.

9.6.2.2 `void SPL::InputBitStream::clearReadCount () [inline]`

Set the read count to zero.

9.6.2.3 `void SPL::OutputBitStream::clearWriteCount () [inline]`

Clear the count of the number of bits written to the bit stream.

9.6.2.4 `BitStream::IoState SPL::BitStream::getIoState () const [inline]`

Get the I/O state of a bit stream.

(This is similar in spirit to `basic_ios::rdstate`.)

9.6.2.5 `InputBitStream::Size SPL::InputBitStream::getReadCount () const [inline]`

Get the number of bits read from the bit stream so far.

9.6.2.6 `InputBitStream::Offset SPL::InputBitStream::getReadLimit () const [inline]`

Get the number of bits that still may be read from the bit stream before the read limit is reached.

If read-limit checking is enabled, the function returns the number of bits that can still be read before the read limit is reached. If read-limit checking is disabled, a negative value is returned.

9.6.2.7 `OutputBitStream::Size SPL::OutputBitStream::getWriteCount () const [inline]`

Get the number of bits written to the bit stream.

9.6.2.8 `OutputBitStream::Offset SPL::OutputBitStream::getWriteLimit () const [inline]`

Get the number of bits that may still be written to the underlying (character) stream.

If write-limit checking is enabled, the function returns the number of bits that can still be written before the write limit is reached. If write-limit checking is disabled, a negative value is returned.

9.6.2.9 `bool SPL::BitStream::isEof () const [inline]`

Test if the bitstream has encountered end-of-file (EOF).

(This is similar in spirit to `basic_ios::eof`.)

9.6.2.10 `bool SPL::BitStream::isLimit () const [inline]`

Test if the bitstream has encountered a read/write limit.

9.6.2.11 `bool SPL::BitStream::isOk () const [inline]`

Test if the bitstream is in an okay (i.e., non-error) state.

(This is similar in spirit to `!basic_ios::fail`.)

9.6.2.12 `void SPL::BitStream::setIoState (BitStream::IoState state) [inline]`

Set the I/O state of a bit stream.

(This is similar in spirit to `basic_ios::clear`.)

9.6.2.13 `void SPL::BitStream::setIoStateBits (BitStream::IoState mask) [inline]`

Set the specified bits in the I/O state of a bit stream.

(This is similar in spirit to `basic_ios::setstate`.)

9.6.2.14 `void SPL::InputBitStream::setReadLimit (InputBitStream::Offset readLimit) [inline]`

Specify the maximum allowable number of bits that may be read from the bit stream.

If `readLimit` is nonnegative, read-limit checking is enabled and the current read limit is set to `readLimit`. If `readLimit` is negative, read-limit checking is disabled.

9.6.2.15 `void SPL::OutputBitStream::setWriteLimit (OutputBitStream::Offset writeLimit) [inline]`

Set the number of bits that may still be written to the bit stream.

If `writeLimit` is nonnegative, write-limit checking is enabled and the current write limit is set to `writeLimit`. If `writeLimit` is negative, write-limit checking is disabled.

9.7 Audio and Image Codecs

Audio and image codecs.

Modules

- [Audio Codecs](#)
Audio file I/O support.
- [Image Codecs](#)
Image file I/O support.

9.7.1 Detailed Description

Audio and image codecs.

9.8 Audio Codecs

Audio file I/O support.

Functions

- int [SPL::loadAudioFile](#) (const std::string &fileName, int &samplingRate, [RealArray1](#) &samples)
Read audio data from a file in WAV format.
- int [SPL::saveAudioFile](#) (const std::string &fileName, int samplingRate, const [RealArray1](#) &samples)
Write a sequence to a file in WAV format.

9.8.1 Detailed Description

Audio file I/O support.

9.8.2 Function Documentation

9.8.2.1 int [SPL::loadAudioFile](#) (const std::string & *fileName*, int & *samplingRate*, [RealArray1](#) & *samples*)

Read audio data from a file in WAV format.

Effects: The audio signal from the file named *fileName* in WAV format is read. The sample data is placed in the array *samples* and the sampling rate is placed in *samplingRate*. If the file has more than one channel, only the first channel is read. The sample data will always lie in the range [-1.0, 1.0].

Returns: On success, zero is returned. On failure, a nonzero value is returned.

9.8.2.2 int [SPL::saveAudioFile](#) (const std::string & *fileName*, int *samplingRate*, const [RealArray1](#) & *samples*)

Write a sequence to a file in WAV format.

Effects: The sequence *seq* with sampling rate *samplingRate* is written to the file named *fileName* in WAV format. The sample data must lie in the range [-1.0, 1.0].

Returns: On success, zero is returned. On failure, a nonzero value is returned.

9.9 Image Codecs

Image file I/O support.

Functions

- `template<class T >`
`int SPL::encodePnm (std::ostream &outStream, const std::vector< Array2< T > > &comps, int maxVal, bool sgnd, bool binaryFormat=true)`
Output the array as an image in the PNM format.
- `template<class T >`
`int SPL::encodePbm (std::ostream &outStream, const Array2< T > &bits, bool binaryFormat=true)`
Output the array as an image in the PNM format (PBM type).
- `template<class T >`
`int SPL::encodePgm (std::ostream &outStream, const Array2< T > &gray, int maxVal, bool sgnd, bool binaryFormat=true)`
Output the array as an image in the PNM format (PGM type).
- `template<class T >`
`int SPL::encodePpm (std::ostream &outStream, const Array2< T > &red, const Array2< T > &green, const Array2< T > &blue, int maxVal, bool sgnd, bool binaryFormat=true)`
Output the array as an image in the PNM format (PPM type).
- `template<class T >`
`int SPL::decodePnm (std::istream &inStream, std::vector< Array2< T > > &comps, int &maxVal, bool &sgnd)`
Input an array as an image in the PNM format.
- `template<class T >`
`int SPL::decodePbm (std::istream &inStream, Array2< T > &bits)`
Input an array as an image in the PNM format.
- `template<class T >`
`int SPL::decodePgm (std::istream &inStream, Array2< T > &gray, int &maxVal, bool &sgnd)`
Input an array as an image in the PNM format.
- `template<class T >`
`int SPL::decodePpm (std::istream &inStream, Array2< T > &red, Array2< T > &green, Array2< T > &blue, int &maxVal, bool &sgnd)`
Input an array as an image in the PNM format.

9.9.1 Detailed Description

Image file I/O support.

9.9.2 Function Documentation

9.9.2.1 `template<class T > int SPL::decodePbm (std::istream & inStream, Array2< T > & bits)`

Input an array as an image in the PNM format.

Effects: A binary image in the PBM format is read from the stream inStream. The parameter bits is set to the image read. The image to be read must be of the PBM type (i.e., binary).

Returns: Upon success, zero is returned; otherwise, a nonzero value is returned.

9.9.2.2 `template<class T> int SPL::decodePgm (std::istream & inStream, Array2< T> & gray, int & maxVal, bool & sgnd)`

Input an array as an image in the PNM format.

Effects: A grayscale image in the PGM format is read from the stream *inStream*. The parameter *gray* is updated to hold the image read. The parameter *maxVal* is set to the maximum value for the image sample data. The parameter *sgnd* is set to indicate whether the image data is signed. The image to be read must be of the PGM type (i.e., grayscale).

Returns: Upon success, zero is returned; otherwise, a nonzero value is returned.

9.9.2.3 `template<class T> int SPL::decodePnm (std::istream & inStream, std::vector< Array2< T> > & comps, int & maxVal, bool & sgnd)`

Input an array as an image in the PNM format.

Effects: An image in the PNM format is read from the stream *inStream*. The parameter *comps* is updated to hold the components of the image read, where the red, green, and blue color components are placed in *comps*[0], *comps*[1], and *comps*[2], respectively. The parameter *maxVal* is set to the maximum value for the image sample data. The parameter *sgnd* is set to indicate whether the image data is signed.

Returns: Upon success, zero is returned; otherwise, a nonzero value is returned.

9.9.2.4 `template<class T> int SPL::decodePpm (std::istream & inStream, Array2< T> & red, Array2< T> & green, Array2< T> & blue, int & maxVal, bool & sgnd)`

Input an array as an image in the PNM format.

Effects: A color image in the PPM format is read from the stream *inStream*. The parameters *red*, *green*, and *blue* are set to RGB components of the image read. The parameter *maxVal* is set to the maximum value for the image sample data. The parameter *sgnd* is set to indicate whether the image data is signed. The image to be read must be of the PPM type (i.e., RGB color).

Returns: Upon success, zero is returned; otherwise, a nonzero value is returned.

9.9.2.5 `template<class T> int SPL::encodePbm (std::ostream & outStream, const Array2< T> & bits, bool binaryFormat = true)`

Output the array as an image in the PNM format (PBM type).

Effects: The binary image in the array *bits* is written to the stream *outStream*. If the parameter *binaryFormat* is true, the binary variant of the PBM format is used; otherwise, the text variant is employed.

Returns: Upon success, zero is returned; otherwise, a nonzero value is returned.

9.9.2.6 `template<class T> int SPL::encodePgm (std::ostream & outStream, const Array2< T> & gray, int maxVal, bool sgnd, bool binaryFormat = true)`

Output the array as an image in the PNM format (PGM type).

Effects: The grayscale image in the array *gray* is written to the stream *outStream*. If the parameter *binaryFormat* is true, the binary variant of the PBM format is used; otherwise, the text variant is employed. The parameter *maxVal* specifies the maximum value for sample data. The parameter *sgnd* specified if the sample data is signed.

Returns: Upon success, zero is returned; otherwise, a nonzero value is returned.

9.9.2.7 `template<class T> int SPL::encodePnm (std::ostream & outStream, const std::vector< Array2< T > > & comps, int maxVal, bool sgnd, bool binaryFormat = true)`

Output the array as an image in the PNM format.

Effects: The image components *comps* are written to the stream *outStream* in the PNM format, where the red, green, and blue color components are given by *comps*[0], *comps*[1], and *comps*[2], respectively. If the parameter *binaryFormat* is true, the binary variant of the PNM format is used; otherwise, the text variant is used. The parameter *maxVal* specifies the maximum value for sample data. The parameter *sgnd* specified if the sample data is signed.

Returns: Upon success, zero is returned; otherwise, a nonzero value is returned.

9.9.2.8 `template<class T> int SPL::encodePpm (std::ostream & outStream, const Array2< T > & red, const Array2< T > & green, const Array2< T > & blue, int maxVal, bool sgnd, bool binaryFormat = true)`

Output the array as an image in the PNM format (PPM type).

Effects: The color image with RGB color planes in the arrays *red*, *green*, and *blue*, respectively, is written to the stream *outStream*. If the parameter *binaryFormat* is true, the binary variant of the PBM format is used; otherwise, the text variant is employed. The parameter *maxVal* specifies the maximum value for sample data. The parameter *sgnd* specified if the sample data is signed.

Returns: Upon success, zero is returned; otherwise, a nonzero value is returned.

9.10 Filter Design

Filter design.

Functions

- [RealSequence1 SPL::lowpassFilter](#) (double cutoffFreq, double transWidth, double maxPassbandRipple=0.1, double minStopbandAtten=20.0)
Design a zero-phase FIR lowpass filter.
- [RealSequence1 SPL::highpassFilter](#) (double cutoffFreq, double transWidth, double maxPassbandRipple=0.1, double minStopbandAtten=20.0)
Design a zero-phase FIR highpass filter.
- [RealSequence1 SPL::bandpassFilter](#) (double cutoffFreq0, double cutoffFreq1, double transWidth0, double transWidth1, double maxPassbandRipple=0.1, double minStopbandAtten=20.0)
Design a zero-phase FIR bandpass filter.

9.10.1 Detailed Description

Filter design.

9.10.2 Function Documentation

9.10.2.1 RealSequence1 SPL::bandpassFilter (double *cutoffFreq0*, double *cutoffFreq1*, double *transWidth0*, double *transWidth1*, double *maxPassbandRipple* = 0.1, double *minStopbandAtten* = 20.0)

Design a zero-phase FIR bandpass filter.

Effects: A linear-phase FIR filter with zero group delay is designed with the (normalized) lower cutoff frequency *cutoffFreq0*, (normalized) upper cutoff frequency *cutoffFreq1*, (normalized) lower transition width *transWidth0*, (normalized) upper transition width *transWidth1*, maximum peak-to-peak passband ripple *maxPassbandRipple* (in dB), and minimum stopband attenuation *minStopbandAtten* (in dB). Note: Frequencies are normalized such that a value of one corresponds to the Nyquist frequency.

The cutoff frequencies cannot be 0 or 1. In other words, this function cannot be used to design a lowpass or highpass filter.

Returns: The impulse response of the designed filter is returned.

9.10.2.2 RealSequence1 SPL::highpassFilter (double *cutoffFreq*, double *transWidth*, double *maxPassbandRipple* = 0.1, double *minStopbandAtten* = 20.0)

Design a zero-phase FIR highpass filter.

Effects: A linear-phase FIR filter with zero group delay is designed with the (normalized) cutoff frequency *cutoffFreq*, (normalized) transition width *transWidth*, maximum peak-to-peak passband ripple *maxPassbandRipple* (in dB), and minimum stopband attenuation *minStopbandAtten* (in dB). Note: Frequencies are normalized such that a value of one corresponds to the Nyquist frequency.

Returns: The impulse response of the designed filter is returned.

9.10.2.3 **RealSequence1** SPL::lowpassFilter (double *cutoffFreq*, double *transWidth*, double *maxPassbandRipple* = 0 . 1, double *minStopbandAtten* = 20 . 0)

Design a zero-phase FIR lowpass filter.

Effects: A linear-phase FIR filter with zero group delay is designed with the (normalized) cutoff frequency *cutoffFreq*, (normalized) transition width *transWidth*, maximum peak-to-peak passband ripple *maxPassbandRipple* (in dB), and minimum stopband attenuation *minStopbandAtten* (in dB). Note: Frequencies are normalized such that a value of one corresponds to the Nyquist frequency.

Returns: The impulse response of the designed filter is returned.

9.11 CPU and Memory Utilization

Support for measuring CPU and memory utilization.

Classes

- class `SPL::Timer`
A class for making timing measurements.

Functions

- double `SPL::getCurrentMemUsage()`
Get the amount of memory currently being used by the process.
- double `SPL::getPeakMemUsage()`
Get the peak memory usage for the process.

9.11.1 Detailed Description

Support for measuring CPU and memory utilization.

9.11.2 Function Documentation

9.11.2.1 double `SPL::getCurrentMemUsage()`

Get the amount of memory currently being used by the process.

Effects: Query the total amount of memory currently being used by the process.

Returns: The amount of memory (in bytes) currently being used by the process is returned.

9.11.2.2 double `SPL::getPeakMemUsage()`

Get the peak memory usage for the process.

Effects: Query the peak memory usage for the process.

Returns: The peak memory usage for the process (in bytes) is returned.

9.12 Math Utilities

Math utilities.

Functions

- `template<class T >`
`T SPL::absVal (T x)`
The absolute value function.
- `template<class T >`
`T SPL::signum (T x)`
The signum function.
- `template<class T >`
`T SPL::sqr (const T &x)`
The square function.
- `template<class T >`
`T SPL::clip (T x, T min, T max)`
The clip function.
- `double SPL::sinc (double x)`
The cardinal sine function.
- `long SPL::roundTowardZeroDiv (long x, long y)`
Compute a quotient with the result rounded towards zero.
- `long SPL::floorDiv (long x, long y)`
Compute the floor of a quotient.
- `template<class T >`
`T SPL::mod (T x, T y)`
Compute the remainder after division.
- `long SPL::ceilDiv (long x, long y)`
Compute the ceiling of a quotient.
- `double SPL::radToDeg (double x)`
Convert from radians to degrees.
- `double SPL::degToRad (double x)`
Convert from degrees to radians.

9.12.1 Detailed Description

Math utilities.

9.12.2 Function Documentation

9.12.2.1 `template<class T > T SPL::absVal (T x) [inline]`

The absolute value function.

Returns: The absolute value of the quantity x is returned.

9.12.2.2 `long SPL::ceilDiv (long x, long y) [inline]`

Compute the ceiling of a quotient.

Returns: The ceiling of x divided by y is returned.

9.12.2.3 `template<class T> T SPL::clip (T x, T min, T max) [inline]`

The clip function.

9.12.2.4 `double SPL::degToRad (double x) [inline]`

Convert from degrees to radians.

Returns: The quantity x converted (from degrees) to radians is returned.

9.12.2.5 `long SPL::floorDiv (long x, long y) [inline]`

Compute the floor of a quotient.

Returns: The floor of x divided by y is returned.

9.12.2.6 `template<class T> T SPL::mod (T x, T y) [inline]`

Compute the remainder after division.

9.12.2.7 `double SPL::radToDeg (double x) [inline]`

Convert from radians to degrees.

Returns: The quantity x converted (from radians) to degrees is returned.

9.12.2.8 `long SPL::roundTowardZeroDiv (long x, long y) [inline]`

Compute a quotient with the result rounded towards zero.

Returns: The floor of x divided by y is returned.

9.12.2.9 `template<class T> T SPL::signum (T x) [inline]`

The signum function.

Returns: The signum of the quantity x is returned.

9.12.2.10 `double SPL::sinc (double x) [inline]`

The cardinal sine function.

Returns: The sinc of x is returned.

9.12.2.11 `template<class T> T SPL::sqr (const T & x) [inline]`

The square function.

Returns: The square of the quantity x is returned.

9.13 CGAL Utilities

CGAL utilities.

Classes

- class [SPL::Arcball< T >](#)
Arcball.
- struct [SPL::Rotation_3< T >](#)
A 3-D rotation.
- struct [SPL::Quaternion< T >](#)
A quaternion represented in terms of its scalar and vector parts.

Functions

- template<class T >
T::Point_3 [SPL::closestPointOnRay](#) (const typename CGAL::Point_3< T > &rayOrigin, const typename CGAL::Vector_3< T > &rayDir, const typename CGAL::Point_3< T > &point)
Compute the closest point on a ray to the specified point.
- template<class T >
std::pair< bool, typename T::Point_3 > [SPL::findRaySphereIntersection](#) (const typename CGAL::Point_3< T > &sphereCenter, typename T::FT sphereRadius, const typename CGAL::Point_3< T > &rayOrigin, const typename CGAL::Vector_3< T > &rayDir)
Compute the intersection of a ray and a sphere.
- template<class T >
std::pair< bool, typename T::Point_3 > [SPL::findRayPlaneIntersection](#) (const typename CGAL::Point_3< T > &planePoint, const typename CGAL::Vector_3< T > &planeNormal, const typename CGAL::Point_3< T > &rayOrigin, const typename CGAL::Vector_3< T > &rayDir)
Compute the intersection of a ray and a plane.
- template<class T >
T::FT [SPL::norm](#) (const typename CGAL::Vector_3< T > &v)
Compute the norm of a vector.
- template<class T >
T::Vector_3 [SPL::normalize](#) (const typename CGAL::Vector_3< T > &v)
Compute a unit vector.
- template<class T >
T::FT [SPL::angleBetweenVectors](#) (const typename CGAL::Vector_3< T > &u, const CGAL::Vector_3< T > &v)
Compute the angle between two vectors.
- template<class T >
Quaternion< T > [SPL::operator*](#) (const Quaternion< T > &q, const Quaternion< T > &r)
Compute the product of two quaternions.
- template<class T >
Quaternion< T > [SPL::operator/](#) (const Quaternion< T > &q, const Quaternion< T > &r)
Compute the quotient of two quaternions.
- template<class T >
Quaternion< T > [SPL::rotationToQuaternion](#) (const Rotation_3< T > &rot)
Convert a rotation into its corresponding quaternion.
- template<class T >
Rotation_3< T > [SPL::quaternionToRotation](#) (const Quaternion< T > &q)

Convert a unit-norm quaternion into its corresponding rotation.

- `SPL::Arcball< T >::Arcball ()`
Create an arcball.
- `void SPL::Arcball< T >::initialize (double arcBallRadius, const Point &eyePos, const Vector &eyeDir, const Vector &eyeUp, const Point &sceneCenter)`
Initialize the state of an arcball.
- `void SPL::Arcball< T >::setMode (int mode)`
Set the arcball rotation mode.
- `void SPL::Arcball< T >::start (const Point &pos)`
Set the starting position for arcball movement.
- `void SPL::Arcball< T >::move (const Point &pos)`
Set the current position for arcball movement.
- `void SPL::Arcball< T >::clear ()`
Clear the starting and current positions for the arcball.
- `Rotation SPL::Arcball< T >::getRotation () const`
Get the rotation required to turn the arcball from the starting position to the current position.
- `static Rotation SPL::Arcball< T >::combineRotations (const Rotation &, const Rotation &)`
Combine two rotations.

9.13.1 Detailed Description

CGAL utilities.

9.13.2 Function Documentation

9.13.2.1 `template<class T> T::FT SPL::angleBetweenVectors (const typename CGAL::Vector_3< T > & u, const CGAL::Vector_3< T > & v) [inline]`

Compute the angle between two vectors.

9.13.2.2 `template<class T> SPL::Arcball< T >::Arcball ()`

Create an arcball.

9.13.2.3 `template<class T> void SPL::Arcball< T >::clear ()`

Clear the starting and current positions for the arcball.

9.13.2.4 `template<class T> T::Point_3 SPL::closestPointOnRay (const typename CGAL::Point_3< T > & rayOrigin, const typename CGAL::Vector_3< T > & rayDir, const typename CGAL::Point_3< T > & point)`

Compute the closest point on a ray to the specified point.

9.13.2.5 `template<class T> Arcball< T >::Rotation SPL::Arcball< T >::combineRotations (const Rotation &, const Rotation &) [static]`

Combine two rotations.

```
9.13.2.6  template<class T> std::pair<bool, typename T::Point_3> SPL::findRayPlaneIntersection ( const typename
CGAL::Point_3< T> & planePoint, const typename CGAL::Vector_3< T> & planeNormal, const typename
CGAL::Point_3< T> & rayOrigin, const typename CGAL::Vector_3< T> & rayDir )
```

Compute the intersection of a ray and a plane.

Compute the intersection of a ray and a plane. The return value is a pair. The first element of the pair is a boolean. This value is true if an intersection point was found, and is false otherwise. The second element in the pair is the intersection point closest to the ray's origin. If no intersection point was found, the ray's origin is returned.

```
9.13.2.7  template<class T> std::pair<bool, typename T::Point_3> SPL::findRaySphereIntersection ( const typename
CGAL::Point_3< T> & sphereCenter, typename T::FT sphereRadius, const typename CGAL::Point_3< T> & rayOrigin,
const typename CGAL::Vector_3< T> & rayDir )
```

Compute the intersection of a ray and a sphere.

Compute the intersection of a ray and a sphere. The return value is a pair. The first element in the pair is a boolean value. This value is true if an intersection point was found, and is false otherwise. The second element in the pair is the intersection point closest to the ray's origin. If no intersection was found, the point on the sphere that is closest to the ray is returned.

```
9.13.2.8  template<class T> Arcball< T>::Rotation SPL::Arcball< T>::getRotation ( ) const
```

Get the rotation required to turn the arcball from the starting position to the current position.

```
9.13.2.9  template<class T> void SPL::Arcball< T>::initialize ( double arcBallRadius, const Point & eyePos, const Vector &
eyeDir, const Vector & eyeUp, const Point & sceneCenter )
```

Initialize the state of an arcball.

```
9.13.2.10 template<class T> void SPL::Arcball< T>::move ( const Point & pos )
```

Set the current position for arcball movement.

```
9.13.2.11 template<class T> T::FT SPL::norm ( const typename CGAL::Vector_3< T> & v ) [inline]
```

Compute the norm of a vector.

```
9.13.2.12 template<class T> T::Vector_3 SPL::normalize ( const typename CGAL::Vector_3< T> & v ) [inline]
```

Compute a unit vector.

Compute a unit vector in the direction of the given vector. If the zero vector is given, the zero vector is returned.

```
9.13.2.13 template<class T> Quaternion<T> SPL::operator* ( const Quaternion< T> & q, const Quaternion< T> & r )
```

Compute the product of two quaternions.

9.13.2.14 `template<class T> Quaternion<T> SPL::operator/ (const Quaternion< T> & q, const Quaternion< T> & r)`

Compute the quotient of two quaternions.

9.13.2.15 `template<class T> Rotation_3<T> SPL::quaternionToRotation (const Quaternion< T> & q)`

Convert a unit-norm quaternion into its corresponding rotation.

9.13.2.16 `template<class T> Quaternion<T> SPL::rotationToQuaternion (const Rotation_3< T> & rot)`

Convert a rotation into its corresponding quaternion.

9.13.2.17 `template<class T> void SPL::Arcball< T>::setMode (int mode)`

Set the arcball rotation mode.

9.13.2.18 `template<class T> void SPL::Arcball< T>::start (const Point & pos)`

Set the starting position for arcball movement.

9.14 Arithmetic Coders

Arithmetic coders.

Modules

- [M-Coder](#)
M-Coder (binary arithmetic coder).
- [Binary and m-ary Arithmetic Coders](#)
Binary and m-ary arithmetic coders.

9.14.1 Detailed Description

Arithmetic coders.

9.15 M-Coder

M-Coder (binary arithmetic coder).

Classes

- class [SPL::MEncoder](#)
The M-Coder (binary) arithmetic encoder class.
- class [SPL::MDecoder](#)
The M-Coder (binary) arithmetic decoder class.

Functions

- long [SPL::MEncoder::getSymCount](#) () const
Get the number of symbols that have been encoded so far.
- long [SPL::MEncoder::getBitCount](#) () const
Get the number of bits (of encoded data) that have been output to the underlying bit stream so far.
- int [SPL::MEncoder::getNumContexts](#) () const
Get the number of contexts.
- OutputBitStream * [SPL::MEncoder::getOutput](#) () const
Get the bit stream being used for output.
- void [SPL::MEncoder::setOutput](#) (OutputBitStream *out)
Set the bit stream to use for output.
- long [SPL::MDecoder::getBitCount](#) () const
Get the number of bits read so far.
- long [SPL::MDecoder::getSymCount](#) () const
Get the number of symbols decoded so far.
- int [SPL::MDecoder::getNumContexts](#) () const
Get the number of contexts.
- InputBitStream * [SPL::MDecoder::getInput](#) () const
Get the input bit stream (i.e., the bit stream from which encoded data is to be read).
- void [SPL::MDecoder::setInput](#) (InputBitStream *in)
Set the input bit stream (i.e., the bit stream from which encoded data is to be read).

9.15.1 Detailed Description

M-Coder (binary arithmetic coder).

9.15.2 Function Documentation

9.15.2.1 long SPL::MEncoder::getBitCount () const [inline]

Get the number of bits (of encoded data) that have been output to the underlying bit stream so far.

9.15.2.2 long SPL::MDecoder::getBitCount () const [inline]

Get the number of bits read so far.

9.15.2.3 `InputBitStream * SPL::MDecoder::getInput () const [inline]`

Get the input bit stream (i.e., the bit stream from which encoded data is to be read).

9.15.2.4 `int SPL::MEncoder::getNumContexts () const [inline]`

Get the number of contexts.

9.15.2.5 `int SPL::MDecoder::getNumContexts () const [inline]`

Get the number of contexts.

9.15.2.6 `OutputBitStream * SPL::MEncoder::getOutput () const [inline]`

Get the bit stream being used for output.

9.15.2.7 `long SPL::MEncoder::getSymCount () const [inline]`

Get the number of symbols that have been encoded so far.

9.15.2.8 `long SPL::MDecoder::getSymCount () const [inline]`

Get the number of symbols decoded so far.

9.15.2.9 `void SPL::MDecoder::setInput (InputBitStream * in) [inline]`

Set the input bit stream (i.e., the bit stream from which encoded data is to be read).

9.15.2.10 `void SPL::MEncoder::setOutput (OutputBitStream * out) [inline]`

Set the bit stream to use for output.

9.16 Binary and m-ary Arithmetic Coders

Binary and m-ary arithmetic coders.

Classes

- class [SPL::MultiArithEncoder](#)
M-ary arithmetic encoder class.
- class [SPL::MultiArithDecoder](#)
M-ary arithmetic decoder class.
- struct [SPL::BinArithCoderContextStat](#)
Binary Arithmetic Coder Context Statistics Class.
- class [SPL::BinArithEncoder](#)
Binary arithmetic encoder class.
- class [SPL::BinArithDecoder](#)
Binary arithmetic decoder class.

Functions

- static std::ostream & [SPL::MultiArithEncoder::getDebugStream](#) ()
Get the stream for debugging output.
- OutputBitStream * [SPL::MultiArithEncoder::getOutput](#) ()
Get the bit stream used for output.
- void [SPL::MultiArithEncoder::setOutput](#) (OutputBitStream *out)
Set the bit stream used for output.
- SPL_ArithCoder_ulong [SPL::MultiArithEncoder::getSymCount](#) () const
Get the number of symbols encoded so far.
- SPL_ArithCoder_ulong [SPL::MultiArithEncoder::getBitCount](#) () const
Get the number of bits of output generated so far including bits awaiting output.
- static void [SPL::MultiArithEncoder::setDebugLevel](#) (int debugLevel)
Set the debug level.
- static void [SPL::MultiArithEncoder::setDebugStream](#) (std::ostream &out)
Set the stream for debugging output.
- int [SPL::MultiArithEncoder::getMaxContexts](#) () const
Get the maximum number of contexts.
- static std::ostream & [SPL::MultiArithDecoder::getDebugStream](#) ()
Get the stream used for debugging output.
- InputBitStream * [SPL::MultiArithDecoder::getInput](#) () const
Get the bit stream from which to read encoded data.
- void [SPL::MultiArithDecoder::setInput](#) (InputBitStream *in)
Set the bit stream from which to read encoded data.
- SPL_ArithCoder_ulong [SPL::MultiArithDecoder::getBitCount](#) () const
Get the number of bits read so far.
- SPL_ArithCoder_ulong [SPL::MultiArithDecoder::getSymCount](#) () const
Get the number of symbols decoded so far.
- static void [SPL::MultiArithDecoder::setDebugLevel](#) (int debugLevel)

- Set the debug level.*

 - static void `SPL::MultiArithDecoder::setDebugStream` (std::ostream &out)

Set the stream to use for debugging output.
- int `SPL::MultiArithDecoder::getMaxContexts` () const

Get the maximum number of contexts.
- SPL_ArithCoder_ulong `SPL::BinArithEncoder::getSymCount` () const

Get the number of symbols output so far.
- SPL_ArithCoder_ulong `SPL::BinArithEncoder::getBitCount` () const

Get the number of bits output so far.
- OutputBitStream * `SPL::BinArithEncoder::getOutput` () const

Get the bit stream to which encoded data should be written.
- int `SPL::BinArithEncoder::getNumContexts` () const

Get the number of contexts.
- static void `SPL::BinArithEncoder::setDebugStream` (std::ostream &out)

Set the stream to use for debugging output.
- static std::ostream & `SPL::BinArithEncoder::getDebugStream` ()

Get the stream used for debugging output.
- static void `SPL::BinArithEncoder::setDebugLevel` (int debugLevel)

Set the debug level.
- void `SPL::BinArithEncoder::setOutput` (OutputBitStream *out)

Set the bit stream to which encoded data should be written.
- SPL_ArithCoder_ulong `SPL::BinArithDecoder::getSymCount` () const

Get the number of symbols decoded so far.
- SPL_ArithCoder_ulong `SPL::BinArithDecoder::getBitCount` () const

Get the number of bits read so far.
- InputBitStream * `SPL::BinArithDecoder::getInput` () const

Get the bit stream from which to read encoded data.
- int `SPL::BinArithDecoder::getNumContexts` () const

Get the number of contexts.
- static void `SPL::BinArithDecoder::setDebugStream` (std::ostream &out)

Set the stream to be used for debugging output.
- static std::ostream & `SPL::BinArithDecoder::getDebugStream` ()

Get the stream used for debugging output.
- static void `SPL::BinArithDecoder::setDebugLevel` (int debugLevel)

Set the debug level.
- void `SPL::BinArithDecoder::setInput` (InputBitStream *in)

Set the bit stream from which to read encoded data.

9.16.1 Detailed Description

Binary and m-ary arithmetic coders.

9.16.2 Function Documentation

9.16.2.1 SPL_ArithCoder_ulong SPL::MultiArithEncoder::getBitCount () const [inline]

Get the number of bits of output generated so far including bits awaiting output.

9.16.2.2 `SPL_ArithCoder_ulong SPL::MultiArithDecoder::getBitCount () const [inline]`

Get the number of bits read so far.

9.16.2.3 `SPL_ArithCoder_ulong SPL::BinArithEncoder::getBitCount () const [inline]`

Get the number of bits output so far.

This function gets the number of bits output so far by the arithmetic encoder.

Returns

The number of bits output so far is returned.

9.16.2.4 `SPL_ArithCoder_ulong SPL::BinArithDecoder::getBitCount () const [inline]`

Get the number of bits read so far.

This function gets the number of bits read so far by the arithmetic decoder.

Returns

The number of bits read so far is returned.

9.16.2.5 `std::ostream & SPL::MultiArithEncoder::getDebugStream () [inline],[static]`

Get the stream for debugging output.

9.16.2.6 `std::ostream & SPL::MultiArithDecoder::getDebugStream () [inline],[static]`

Get the stream used for debugging output.

9.16.2.7 `std::ostream & SPL::BinArithEncoder::getDebugStream () [inline],[static]`

Get the stream used for debugging output.

9.16.2.8 `std::ostream & SPL::BinArithDecoder::getDebugStream () [inline],[static]`

Get the stream used for debugging output.

9.16.2.9 `InputBitStream * SPL::MultiArithDecoder::getInput () const [inline]`

Get the bit stream from which to read encoded data.

9.16.2.10 `InputBitStream * SPL::BinArithDecoder::getInput () const [inline]`

Get the bit stream from which to read encoded data.

This function gets the bit stream from which to read encoded data.

Returns

A pointer to the bit stream is returned.

9.16.2.11 `int SPL::MultiArithEncoder::getMaxContexts () const [inline]`

Get the maximum number of contexts.

9.16.2.12 `int SPL::MultiArithDecoder::getMaxContexts () const [inline]`

Get the maximum number of contexts.

9.16.2.13 `int SPL::BinArithEncoder::getNumContexts () const [inline]`

Get the number of contexts.

This function gets the number of contexts employed by the arithmetic encoder.

Returns

The number of contexts is returned.

9.16.2.14 `int SPL::BinArithDecoder::getNumContexts () const [inline]`

Get the number of contexts.

This function gets the number of contexts employed by the arithmetic decoder.

Returns

The number of contexts is returned.

9.16.2.15 `OutputBitStream * SPL::MultiArithEncoder::getOutput () [inline]`

Get the bit stream used for output.

9.16.2.16 `OutputBitStream * SPL::BinArithEncoder::getOutput () const [inline]`

Get the bit stream to which encoded data should be written.

This function gets the bit stream to which encoded data should be written.

Returns

A pointer to the output bit stream is returned.

9.16.2.17 `SPL_ArithCoder_ulong SPL::MultiArithEncoder::getSymCount () const [inline]`

Get the number of symbols encoded so far.

9.16.2.18 `SPL_ArithCoder_ulong SPL::MultiArithDecoder::getSymCount () const [inline]`

Get the number of symbols decoded so far.

9.16.2.19 `SPL_ArithCoder_ulong SPL::BinArithEncoder::getSymCount () const [inline]`

Get the number of symbols output so far.

This function gets the number of symbols output so far by the arithmetic encoder.

Returns

The number of symbols output so far is returned.

9.16.2.20 `SPL_ArithCoder_ulong SPL::BinArithDecoder::getSymCount () const [inline]`

Get the number of symbols decoded so far.

This function gets the number of symbols decoded so far by the arithmetic decoder.

Returns

The number of symbols decoded so far is returned.

9.16.2.21 `void SPL::MultiArithEncoder::setDebugLevel (int debugLevel) [inline],[static]`

Set the debug level.

9.16.2.22 `void SPL::MultiArithDecoder::setDebugLevel (int debugLevel) [inline],[static]`

Set the debug level.

9.16.2.23 `void SPL::BinArithEncoder::setDebugLevel (int debugLevel) [inline],[static]`

Set the debug level.

9.16.2.24 `void SPL::BinArithDecoder::setDebugLevel (int debugLevel) [inline],[static]`

Set the debug level.

9.16.2.25 `void SPL::MultiArithEncoder::setDebugStream (std::ostream & out) [inline],[static]`

Set the stream for debugging output.

9.16.2.26 `void SPL::MultiArithDecoder::setDebugStream (std::ostream & out) [inline],[static]`

Set the stream to use for debugging output.

9.16.2.27 `void SPL::BinArithEncoder::setDebugStream (std::ostream & out) [inline],[static]`

Set the stream to use for debugging output.

9.16.2.28 `void SPL::BinArithDecoder::setDebugStream (std::ostream & out) [inline],[static]`

Set the stream to be used for debugging output.

9.16.2.29 `void SPL::MultiArithDecoder::setInput (InputBitStream * in) [inline]`

Set the bit stream from which to read encoded data.

9.16.2.30 `void SPL::BinArithDecoder::setInput (InputBitStream * in) [inline]`

Set the bit stream from which to read encoded data.

Parameters

<i>in</i>	The input bit stream.
-----------	-----------------------

This function sets the bit stream from which to read encoded data to *in*.

9.16.2.31 `void SPL::MultiArithEncoder::setOutput (OutputBitStream * out) [inline]`

Set the bit stream used for output.

9.16.2.32 `void SPL::BinArithEncoder::setOutput (OutputBitStream * out) [inline]`

Set the bit stream to which encoded data should be written.

Parameters

<i>out</i>	The output bit stream.
------------	------------------------

This function sets the bit stream to which encoded data should be written.

Chapter 10

Class Documentation

10.1 SPL::Arcball< T > Class Template Reference

[Arcball](#).

```
#include <Arcball.hpp>
```

Public Types

- typedef T [Kernel](#)
The CGAL kernel.
- typedef Kernel::Point_3 [Point](#)
The point type.
- typedef Kernel::Vector_3 [Vector](#)
The vector type.
- typedef [Rotation_3](#)< [Kernel](#) > [Rotation](#)
The representation of a rotation.

Public Member Functions

- [Arcball](#) ()
Create an arcball.
- void [initialize](#) (double arcBallRadius, const [Point](#) &eyePos, const [Vector](#) &eyeDir, const [Vector](#) &eyeUp, const [Point](#) &sceneCenter)
Initialize the state of an arcball.
- void [setMode](#) (int mode)
Set the arcball rotation mode.
- void [start](#) (const [Point](#) &pos)
Set the starting position for arcball movement.
- void [move](#) (const [Point](#) &pos)
Set the current position for arcball movement.
- void [clear](#) ()
Clear the starting and current positions for the arcball.
- [Rotation](#) [getRotation](#) () const

Get the rotation required to turn the arcball from the starting position to the current position.

- void [setDebugLevel](#) (int debugLevel) const

For debugging...

Static Public Member Functions

- static [Rotation combineRotations](#) (const [Rotation](#) &, const [Rotation](#) &)

Combine two rotations.

10.1.1 Detailed Description

template<class T>class [SPL::Arcball](#)< T >

[Arcball](#).

References: K. Shoemake, [Arcball](#) Rotation Control, Graphics Gems IV, 1994, pp. 175-192.

10.1.2 Member Typedef Documentation

10.1.2.1 template<class T> typedef T [SPL::Arcball](#)< T >::Kernel

The CGAL kernel.

10.1.2.2 template<class T> typedef Kernel::Point_3 [SPL::Arcball](#)< T >::Point

The point type.

10.1.2.3 template<class T> typedef Rotation_3<Kernel> [SPL::Arcball](#)< T >::Rotation

The representation of a rotation.

10.1.2.4 template<class T> typedef Kernel::Vector_3 [SPL::Arcball](#)< T >::Vector

The vector type.

10.1.3 Member Function Documentation

10.1.3.1 template<class T> void [SPL::Arcball](#)< T >::setDebugLevel (int *debugLevel*) const

For debugging...

The documentation for this class was generated from the following file:

- [Arcball.hpp](#)

10.2 SPL::Array1< T > Class Template Reference

A one-dimensional array class with lazy copying and reference counting.

```
#include <Array1.hpp>
```

Public Types

- typedef T [ElemType](#)
The type of the elements in the array.
- typedef std::vector< T >::iterator [Iterator](#)
A mutable iterator for the array elements.
- typedef std::vector< T >::const_iterator [ConstIterator](#)
A constant iterator for the array elements.

Public Member Functions

- [Array1](#) ()
Create an empty array.
- [Array1](#) (int size)
Create an array of the specified size.
- [Array1](#) (int size, const T &value)
Create an array of the given size with all elements initialized to the specified value.
- template<class InputIterator >
[Array1](#) (int size, InputIterator data)
Create an array of the specified size with the elements initialized to the data obtained from the given input iterator.
- [Array1](#) (const [Array1](#) &a)
Create a copy of an array.
- template<class OtherType >
[Array1](#) (const [Array1](#)< OtherType > &a)
Create a copy of an array with elements of arbitrary type.
- ~[Array1](#) ()
Destroy an array.
- [Array1](#) & operator= (const [Array1](#) &a)
Assign one array to another.
- template<class OtherType >
[Array1](#)< T > & operator= (const [Array1](#)< OtherType > &a)
Assign an array with elements of arbitrary type to another array.
- [Array1](#) & operator+= (const [Array1](#) &a)
Add another array (elementwise) to this array.
- [Array1](#) & operator-= (const [Array1](#) &a)
Subtract another array (elementwise) from this array.
- [Array1](#) & operator*= (const [Array1](#) &a)
Multiply another array (elementwise) by this array.
- [Array1](#) & operator/= (const [Array1](#) &a)
Divide this array (elementwise) by another array.
- [Array1](#) & operator+= (const T &value)
Add the specified value to each element in the array.

- `Array1 & operator-= (const T &value)`
Subtract the specified value from each element in the array.
- `Array1 & operator*-= (const T &value)`
Multiply each element in the array by the specified value.
- `Array1 & operator/= (const T &value)`
Divide each element in the array by the specified value.
- `int getSize () const`
Get the number of elements in the array.
- `bool isShared () const`
Is the data for this array shared with another array?
- `bool isSharedWith (const Array1 &a) const`
Is the data for this array shared with the specified array?
- `T & operator() (int i)`
Get a mutable reference to the specified element in the array.
- `const T & operator() (int i) const`
Get a const reference to the specified element in the array.
- `ConstIterator begin () const`
Get a const iterator referring to the first element in the array.
- `Iterator begin ()`
Get a mutable iterator referring to the first element in the array.
- `ConstIterator end () const`
Get a const iterator referring to one past the last element in the array.
- `Iterator end ()`
Get a mutable iterator referring to one past the last element in the array.
- `void resize (int size)`
Change the size of the array.
- `template<class InputIterator >`
`void resize (int size, InputIterator data)`
Change the size of the array, initializing the resized array with the data obtained from the specified input iterator.
- `T max () const`
Get the maximum of the elements in the array.
- `T min () const`
Get the minimum of the elements in the array.
- `T sum () const`
Get the sum of the elements in the array.
- `std::ostream & output (std::ostream &out, int fieldWidth) const`
Output an array to a stream with a particular field width to be used for each element.
- `int load (const char *fileName)`
Load an array from the file with the specified name.
- `int save (const char *fileName) const`
Save an array to the file with the specified name.
- `void fill (const T &value=T(0))`
Set all elements in the array to the specified value.
- `void swap (Array1 &a)`
Swap the contents of the array with the contents of another array.
- `void dump (std::ostream &out) const`
Output information about an array to a stream for debugging.

10.2.1 Detailed Description

`template<class T>class SPL::Array1< T >`

A one-dimensional array class with lazy copying and reference counting.

10.2.2 Member Typedef Documentation

10.2.2.1 `template<class T> typedef std::vector<T>::const_iterator SPL::Array1< T >::ConstIterator`

A constant iterator for the array elements.

10.2.2.2 `template<class T> typedef T SPL::Array1< T >::ElemType`

The type of the elements in the array.

10.2.2.3 `template<class T> typedef std::vector<T>::iterator SPL::Array1< T >::Iterator`

A mutable iterator for the array elements.

10.2.3 Constructor & Destructor Documentation

10.2.3.1 `template<class T> template<class InputIterator > SPL::Array1< T >::Array1 (int size, InputIterator data)`

Create an array of the specified size with the elements initialized to the data obtained from the given input iterator.

10.2.3.2 `template<class T> template<class OtherType > SPL::Array1< T >::Array1 (const Array1< OtherType > & a)`

Create a copy of an array with elements of arbitrary type.

Note: The type OtherType must be assignable to the type T.

The documentation for this class was generated from the following file:

- [Array1.hpp](#)

10.3 SPL::Array2< T > Class Template Reference

A two-dimensional array class with lazy copying and reference counting.

```
#include <Array2.hpp>
```

Public Types

- typedef T [ElemType](#)
The type of the elements in the array.
- typedef std::vector< T >::iterator [Iterator](#)
A mutable iterator for all elements in the array.

- `typedef std::vector< T >::const_iterator` [ConstIterator](#)
A constant iterator for all elements in the array.
- `typedef` [Iterator](#) [XIterator](#)
A mutable iterator for elements of a row in the array.
- `typedef std::vector< T >::const_iterator` [ConstXIterator](#)
A constant iterator for elements of a row in the array.
- `typedef YIter< T >` [YIterator](#)
A mutable iterator for elements of a column in the array.
- `typedef YIter< const T >` [ConstYIterator](#)
A constant iterator for elements of a column in the array.

Public Member Functions

- [Array2](#) ()
Create an empty array.
- [Array2](#) (int width, int height)
Create an array of the specified width and height.
- [Array2](#) (int width, int height, const T &value)
Create an array of the specified width and height with the elements of the array initialized to the specified value.
- `template<class InputIter >`
[Array2](#) (int width, int height, InputIter data)
Create an array of the specified width and height with the elements of the array initialized to the specified data.
- [~Array2](#) ()
The destructor.
- [Array2](#) (const [Array2](#) &a)
The copy constructor.
- `template<class OtherType >`
[Array2](#) (const [Array2](#)< OtherType > &a)
Create an array from an array having elements of a different type.
- [Array2](#) & `operator=` (const [Array2](#) &a)
The assignment operator.
- `template<class OtherType >`
[Array2](#) & `operator=` (const [Array2](#)< OtherType > &a)
Assign another array with elements of a different type to this array.
- [Array2](#) & `operator+=` (const [Array2](#) &a)
Add another array (elementwise) to this array.
- [Array2](#) & `operator-=` (const [Array2](#) &a)
Subtract another array (elementwise) from this array.
- [Array2](#) & `operator*=` (const [Array2](#) &a)
Multiply another array (elementwise) by this array.
- [Array2](#) & `operator/=` (const [Array2](#) &a)
Divide this array (elementwise) by another array.
- [Array2](#) & `operator+=` (const T &a)
Add the specified value to each element in the array.
- [Array2](#) & `operator-=` (const T &a)
Subtract the specified value from each element in the array.
- [Array2](#) & `operator*=` (const T &a)

- Multiply each element in the array by the specified value.*

 - `Array2 & operator/=(const T &a)`

Divide each element in the array by the specified value.
- `int getWidth()` const

Get the width of the array.
- `int getHeight()` const

Get the height of the array.
- `int getSize()` const

Get the number of elements in the array.
- `bool isShared()` const

Is the data for this array shared with another array?
- `bool isSharedWith(const Array2 &a)` const

Is the data for this array shared with the specified array?
- `T & operator()(int x, int y)`

Get a mutable reference to the (x,y)-th element in the array.
- `const T & operator()(int x, int y)` const

Get a const reference to the (x,y)-th element in the array.
- `T & operator()(int i)`

Get a mutable reference to the i-th element in the array.
- `const T & operator()(int i)` const

Get a const reference to the i-th element in the array.
- `ConstIterator begin()` const

Get a const iterator for the first element in the array.
- `Iterator begin()`

Get a mutable iterator for the first element in the array.
- `ConstIterator end()` const

Get a const iterator for one past the last element in the array.
- `Iterator end()`

Get a mutable iterator for one past the last element in the array.
- `ConstXIterator rowBegin(int y)` const

Get a const iterator for the first element in the specified row of the array.
- `XIterator rowBegin(int y)`

Get a mutable iterator for the first element in the specified row of the array.
- `ConstXIterator rowEnd(int y)` const

Get a const iterator for one past the end in the specified row of the array.
- `XIterator rowEnd(int y)`

Get a mutable iterator for one past the end in the specified row of the array.
- `ConstYIterator colBegin(int x)` const

Get a const iterator for the first element in the specified column of the array.
- `YIterator colBegin(int x)`

Get a mutable iterator for the first element in the specified column of the array.
- `ConstYIterator colEnd(int x)` const

Get a const iterator for one past the end in the specified column of the array.
- `YIterator colEnd(int x)`

Get a mutable iterator for one past the end in the specified column of the array.
- `void resize(int width, int height)`

Change the size of the array.

- `template<class InputIterator >`
`void resize (int width, int height, InputIterator data)`
Change the size of the array, initializing the resized array with the data obtained from the specified input iterator.
- `T max () const`
Get the maximum of the elements in the array.
- `T min () const`
Get the minimum of the elements in the array.
- `T sum () const`
Get the sum of the elements in the array.
- `std::ostream & output (std::ostream &out, int fieldWidth) const`
Output an array to a stream using the specified field width for each array element.
- `int load (const char *fileName)`
Load an array from the file with the specified name.
- `int save (const char *fileName) const`
Save an array to the file with the specified name.
- `void fill (const T &value=T(0))`
Set all elements in the array to the specified value.
- `Array2 & flipud ()`
Flip the array upside down.
- `Array2 & fliplr ()`
Flip the array left to right.
- `void swap (Array2 &a)`
Swap the array data with the data of the specified array.
- `void dump (std::ostream &out) const`
Output information about an array to a stream for debugging.
- `void unshare () const`
Force the underlying data to be copied if the data is shared.

10.3.1 Detailed Description

`template<class T>class SPL::Array2< T >`

A two-dimensional array class with lazy copying and reference counting.

10.3.2 Member Typedef Documentation

10.3.2.1 `template<class T> typedef std::vector<T>::const_iterator SPL::Array2< T >::ConstIterator`

A constant iterator for all elements in the array.

10.3.2.2 `template<class T> typedef std::vector<T>::const_iterator SPL::Array2< T >::ConstXIterator`

A constant iterator for elements of a row in the array.

10.3.2.3 `template<class T> typedef YIter<const T> SPL::Array2< T >::ConstYIterator`

A constant iterator for elements of a column in the array.

10.3.2.4 `template<class T> typedef T SPL::Array2< T >::ElemType`

The type of the elements in the array.

10.3.2.5 `template<class T> typedef std::vector<T>::iterator SPL::Array2< T >::Iterator`

A mutable iterator for all elements in the array.

10.3.2.6 `template<class T> typedef Iterator SPL::Array2< T >::XIterator`

A mutable iterator for elements of a row in the array.

10.3.2.7 `template<class T> typedef YIter<T> SPL::Array2< T >::YIterator`

A mutable iterator for elements of a column in the array.

10.3.3 Constructor & Destructor Documentation

10.3.3.1 `template<class T> template<class InputIter > SPL::Array2< T >::Array2 (int width, int height, InputIter data)`

Create an array of the specified width and height with the elements of the array initialized to the specified data.

10.3.3.2 `template<class T> template<class OtherType > SPL::Array2< T >::Array2 (const Array2< OtherType > & a)`

Create an array from an array having elements of a different type.

10.3.4 Member Function Documentation

10.3.4.1 `template<class T> template<class OtherType > Array2& SPL::Array2< T >::operator= (const Array2< OtherType > & a)`

Assign another array with elements of a different type to this array.

The documentation for this class was generated from the following file:

- [Array2.hpp](#)

10.4 SPL::BinArithCoderContextStat Struct Reference

Binary Arithmetic Coder Context Statistics Class.

```
#include <arithCoder.hpp>
```

10.4.1 Detailed Description

Binary Arithmetic Coder Context Statistics Class.

The documentation for this struct was generated from the following file:

- `include/SPL/arithCoder.hpp`

10.5 SPL::BinArithDecoder Class Reference

Binary arithmetic decoder class.

```
#include <arithCoder.hpp>
```

Public Member Functions

- [BinArithDecoder](#) (int numContexts, [InputBitStream](#) *in=nullptr)
Create a decoder with the specified number of contexts that receives input from the given bit stream.
- [~BinArithDecoder](#) ()
Destroy a decoder.
- [SPL_ArithCoder_ulong](#) [getSymCount](#) () const
Get the number of symbols decoded so far.
- [SPL_ArithCoder_ulong](#) [getBitCount](#) () const
Get the number of bits read so far.
- void [setInput](#) ([InputBitStream](#) *in)
Set the bit stream from which to read encoded data.
- [InputBitStream](#) * [getInput](#) () const
Get the bit stream from which to read encoded data.
- int [getNumContexts](#) () const
Get the number of contexts.
- void [setContextState](#) (int contextId, [ArithCoder::Freq](#) oneFreq, [ArithCoder::Freq](#) totalFreq, [ArithCoder::Freq](#) maxFreq, bool adaptive)
Set the symbol probabilities and adaptivity for the specified context.
- void [getContextState](#) (int contextId, [ArithCoder::Freq](#) &oneFreq, [ArithCoder::Freq](#) &totalFreq, [ArithCoder::Freq](#) &maxFreq, bool &adaptive)
Get the symbol probabilities and adaptivity for the specified context.
- int [start](#) ()
Start a code word.
- int [terminate](#) ()
Terminate the code word (for synchronization with the encoder).
- int [decodeRegular](#) (int contextId)
Decode a symbol in the specified context.
- int [decodeBypass](#) ()
Decode a symbol in bypass mode (i.e., using a fixed probability distribution with all symbols being equiprobable).
- void [dump](#) (std::ostream &out) const
Dump the internal decoder state to the specified stream for debugging purposes.

Static Public Member Functions

- static void [setDebugLevel](#) (int debugLevel)
Set the debug level.
- static void [setDebugStream](#) (std::ostream &out)
Set the stream to be used for debugging output.
- static std::ostream & [getDebugStream](#) ()
Get the stream used for debugging output.

10.5.1 Detailed Description

Binary arithmetic decoder class.

10.5.2 Constructor & Destructor Documentation

10.5.2.1 SPL::BinArithDecoder::BinArithDecoder (int *numContexts*, InputBitStream * *in* = nullptr)

Create a decoder with the specified number of contexts that receives input from the given bit stream.

Parameters

<i>numContexts</i>	The number of contexts.
<i>in</i>	The input bit stream.

This function creates an arithmetic decoder with the number of contexts being `numContexts` that receives input from the bit stream `in`.

10.5.2.2 SPL::BinArithDecoder::~~BinArithDecoder ()

Destroy a decoder.

This function destroys an arithmetic decoder.

10.5.3 Member Function Documentation

10.5.3.1 int SPL::BinArithDecoder::decodeBypass ()

Decode a symbol in bypass mode (i.e., using a fixed probability distribution with all symbols being equiprobable).

This function decodes a symbol in bypass mode (i.e., using a fixed probability distribution with all symbols being equiprobable).

Returns

Upon success, the decoded symbol is returned (which is either 0 or 1); otherwise, a negative value is returned.

10.5.3.2 int SPL::BinArithDecoder::decodeRegular (int *contextId*)

Decode a symbol in the specified context.

Parameters

<i>contextId</i>	The ID of the context to be used for decoding.
------------------	--

The function decodes a symbol using the context specified by `contextId`. The value of `contextId` must be from 0 to `n - 1` (inclusive), where `n` is the number of contexts employed by the arithmetic decoder.

Returns

Upon success, the decoded symbol is returned (which is either 0 or 1); otherwise, a negative value is returned.

10.5.3.3 void SPL::BinArithDecoder::dump (std::ostream & *out*) const

Dump the internal decoder state to the specified stream for debugging purposes.

10.5.3.4 `void SPL::BinArithDecoder::getContextState (int contextId, ArithCoder::Freq & oneFreq, ArithCoder::Freq & totalFreq, ArithCoder::Freq & maxFreq, bool & adaptive)`

Get the symbol probabilities and adaptivity for the specified context.

Parameters

	<i>contextId</i>	The ID of the context to query.
out	<i>oneFreq</i>	The frequency count for a one symbol.
out	<i>totalFreq</i>	The normalizing frequency count for all symbols.
out	<i>maxFreq</i>	The maximum normalizing frequency count.
out	<i>adaptive</i>	The adaptivity flag.

This function queries the state of the context with the context ID `contextId`, and sets the parameters `oneFreq`, `totalFreq`, `maxFreq`, and `adaptive` appropriately. The probability of a one symbol is given by `oneFreq / totalFreq`, while the probability of a zero symbol is given by `1 - oneFreq / totalFreq`.

10.5.3.5 `void SPL::BinArithDecoder::setContextState (int contextId, ArithCoder::Freq oneFreq, ArithCoder::Freq totalFreq, ArithCoder::Freq maxFreq, bool adaptive)`

Set the symbol probabilities and adaptivity for the specified context.

10.5.3.6 `int SPL::BinArithDecoder::start ()`

Start a code word.

This function starts the decoding of a new arithmetic code word. This function must be called before attempting to decode any symbols.

Returns

Upon success, zero is returned; otherwise, a nonzero value is returned.

10.5.3.7 `int SPL::BinArithDecoder::terminate ()`

Terminate the code word (for synchronization with the encoder).

This function terminates the decoding of the current arithmetic code word.

Returns

Upon success, zero is returned; otherwise, a nonzero value is returned.

The documentation for this class was generated from the following files:

- `include/SPL/arithCoder.hpp`
- `arithCoder.cpp`

10.6 SPL::BinArithEncoder Class Reference

Binary arithmetic encoder class.

```
#include <arithCoder.hpp>
```

Public Member Functions

- [BinArithEncoder](#) (int numContexts, [OutputBitStream](#) *out=nullptr)
Create an arithmetic encoder with the specified number of contexts that sends output to the given bit stream.
- [~BinArithEncoder](#) ()
Destroy an arithmetic encoder.
- int [getNumContexts](#) () const
Get the number of contexts.
- SPL_ArithCoder_ulong [getSymCount](#) () const
Get the number of symbols output so far.
- SPL_ArithCoder_ulong [getBitCount](#) () const
Get the number of bits output so far.
- void [setOutput](#) ([OutputBitStream](#) *out)
Set the bit stream to which encoded data should be written.
- [OutputBitStream](#) * [getOutput](#) () const
Get the bit stream to which encoded data should be written.
- void [setContextState](#) (int contextId, ArithCoder::Freq oneFreq, ArithCoder::Freq totalFreq, ArithCoder::Freq maxFreq, bool adaptive)
Set the symbol probabilities and adaptivity for the specified context.
- void [getContextState](#) (int contextId, ArithCoder::Freq &oneFreq, ArithCoder::Freq &totalFreq, ArithCoder::Freq &maxFreq, bool &adaptive)
Get the symbol probabilities and adaptivity for the specified context.
- int [start](#) ()
Start a code word.
- int [encodeRegular](#) (int contextId, int binVal)
Encode the specified symbol in the given context.
- int [encodeBypass](#) (int binVal)
Encode the specified symbol in bypass mode (i.e., using a fixed probability distribution with all symbols being equiprobable).
- int [terminate](#) ()
Terminate the code word.
- void [dump](#) (std::ostream &out) const
Dump the internal encoder state to the specified output stream for debugging purposes.
- void [dumpModels](#) (std::ostream &out) const
Dump the internal encoder context state to the specified output stream for debugging purposes.

Static Public Member Functions

- static void [setDebugLevel](#) (int debugLevel)
Set the debug level.
- static void [setDebugStream](#) (std::ostream &out)
Set the stream to use for debugging output.
- static std::ostream & [getDebugStream](#) ()
Get the stream used for debugging output.

10.6.1 Detailed Description

Binary arithmetic encoder class.

10.6.2 Constructor & Destructor Documentation

10.6.2.1 `SPL::BinArithEncoder::BinArithEncoder (int numContexts, OutputBitStream * out = nullptr)`

Create an arithmetic encoder with the specified number of contexts that sends output to the given bit stream.

Parameters

<i>numContexts</i>	The number of contexts.
<i>out</i>	The output bit stream.

This constructor creates an arithmetic encoder with the number of contexts being `numContexts` that sends output to the bit stream `out`.

10.6.2.2 `SPL::BinArithEncoder::~~BinArithEncoder ()`

Destroy an arithmetic encoder.

This destructor destroys an arithmetic encoder.

10.6.3 Member Function Documentation

10.6.3.1 `void SPL::BinArithEncoder::dump (std::ostream & out) const`

Dump the internal encoder state to the specified output stream for debugging purposes.

10.6.3.2 `void SPL::BinArithEncoder::dumpModels (std::ostream & out) const`

Dump the internal encoder context state to the specified output stream for debugging purposes.

10.6.3.3 `int SPL::BinArithEncoder::encodeBypass (int binVal)`

Encode the specified symbol in bypass mode (i.e., using a fixed probability distribution with all symbols being equiprobable).

Parameters

<i>binVal</i>	The symbol to be encoded.
---------------	---------------------------

This function encodes the symbol `binVal` in bypass mode (i.e., using a fixed probability distribution with all symbols being equiprobable). The symbol to be encoded must be either 0 or 1.

Returns

Upon success, zero is returned; otherwise, a nonzero value is returned.

10.6.3.4 `int SPL::BinArithEncoder::encodeRegular (int contextId, int binVal)`

Encode the specified symbol in the given context.

Parameters

<i>contextId</i>	The ID of the context to be used for encoding.
<i>binVal</i>	The symbol to be encoded.

This function encodes the symbol `binVal` using the context specified by `contextId`. The value of `contextId` must be from 0 to `n - 1` (inclusive), where `n` is the number of contexts employed by the arithmetic encoder. The symbol to be encoded must be either 0 or 1.

Returns

Upon success, zero is returned; otherwise, a nonzero value is returned.

10.6.3.5 `void SPL::BinArithEncoder::getContextState (int contextId, ArithCoder::Freq & oneFreq, ArithCoder::Freq & totalFreq, ArithCoder::Freq & maxFreq, bool & adaptive)`

Get the symbol probabilities and adaptivity for the specified context.

Parameters

	<i>contextId</i>	The ID of the context to query.
out	<i>oneFreq</i>	The frequency count for a one symbol.
out	<i>totalFreq</i>	The normalizing frequency count for all symbols.
out	<i>maxFreq</i>	The maximum normalizing frequency count.
out	<i>adaptive</i>	The adaptivity flag.

This function queries the state of the context with the context ID `contextId`, and sets the parameters `oneFreq`, `totalFreq`, `maxFreq`, and `adaptive` appropriately. The probability of a one symbol is given by `oneFreq / totalFreq`, while the probability of a zero symbol is given by `1 - oneFreq / totalFreq`.

10.6.3.6 `void SPL::BinArithEncoder::setContextState (int contextId, ArithCoder::Freq oneFreq, ArithCoder::Freq totalFreq, ArithCoder::Freq maxFreq, bool adaptive)`

Set the symbol probabilities and adaptivity for the specified context.

10.6.3.7 `int SPL::BinArithEncoder::start ()`

Start a code word.

This function starts the encoding of a new code word. This function must be called before attempting to encode any symbols.

Returns

Upon success, zero is returned; otherwise, a nonzero value is returned.

10.6.3.8 `int SPL::BinArithEncoder::terminate ()`

Terminate the code word.

This function terminates the encoding of the current arithmetic code word.

Returns

Upon success, zero is returned; otherwise, a nonzero value is returned.

The documentation for this class was generated from the following files:

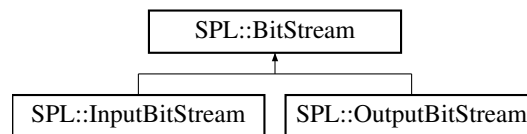
- include/SPL/arithCoder.hpp
- arithCoder.cpp

10.7 SPL::BitStream Class Reference

A common base class for the input and output bit stream classes.

```
#include <bitStream.hpp>
```

Inheritance diagram for SPL::BitStream:

**Public Types**

- typedef unsigned [loState](#)
The type used for the error state for a stream.
- typedef unsigned long long [Size](#)
An unsigned integral type (used for sizes/counts).
- typedef long long [Offset](#)
A signed integral type (used for differences).

Public Member Functions

- bool [isOkay](#) () const
Test if the bitstream is in an okay (i.e., non-error) state.
- bool [isEof](#) () const
Test if the bitstream has encountered end-of-file (EOF).
- bool [isLimit](#) () const
Test if the bitstream has encountered a read/write limit.
- [loState](#) [getloState](#) () const
Get the I/O state of a bit stream.
- void [setloState](#) ([loState](#) state)
Set the I/O state of a bit stream.
- void [setloStateBits](#) ([loState](#) state)
Set the specified bits in the I/O state of a bit stream.
- void [clearloStateBits](#) ([loState](#) state=[allloBits](#))
Clear the specified bits in the I/O state of a bit stream.

Static Public Attributes

- static const `loState eofBit` = 1
end of file (EOF) reached on input
- static const `loState limitBit` = 2
read/write limit exceeded
- static const `loState badBit` = 4
I/O error.
- static const `loState allIoBits` = `eofBit` | `limitBit` | `badBit`
all error bits

10.7.1 Detailed Description

A common base class for the input and output bit stream classes.

This class provides some error handling functionality common to the input and output bit stream classes.

10.7.2 Member Typedef Documentation

10.7.2.1 `typedef unsigned SPL::BitStream::loState`

The type used for the error state for a stream.

10.7.2.2 `typedef long long SPL::BitStream::Offset`

A signed integral type (used for differences).

10.7.2.3 `typedef unsigned long long SPL::BitStream::Size`

An unsigned integral type (used for sizes/counts).

10.7.3 Member Data Documentation

10.7.3.1 `const loState SPL::BitStream::allIoBits = eofBit | limitBit | badBit` [static]

all error bits

10.7.3.2 `const loState SPL::BitStream::badBit = 4` [static]

I/O error.

10.7.3.3 `const loState SPL::BitStream::eofBit = 1` [static]

end of file (EOF) reached on input

10.7.3.4 `const IoState SPL::BitStream::limitBit = 2` `[static]`

read/write limit exceeded

The documentation for this class was generated from the following file:

- [bitStream.hpp](#)

10.8 SPL::ConvolveMode Struct Reference

Constants identifying various convolution modes.

```
#include <Sequence.hpp>
```

Static Public Attributes

- static const int `full` = 0
The full convolution result (i.e., the same as "full" in MATLAB)
- static const int `sameDomainZeroExt` = 1
The same as "same" in MATLAB.
- static const int `sameDomainConstExt` = 3
Constant extension.
- static const int `sameDomainPerExt` = 2
Periodic extension.
- static const int `sameDomainSymExt0` = 4
Symmetric periodic extension.

10.8.1 Detailed Description

Constants identifying various convolution modes.

10.8.2 Member Data Documentation

10.8.2.1 `const int SPL::ConvolveMode::full = 0` `[static]`

The full convolution result (i.e., the same as "full" in MATLAB)

10.8.2.2 `const int SPL::ConvolveMode::sameDomainConstExt = 3` `[static]`

Constant extension.

10.8.2.3 `const int SPL::ConvolveMode::sameDomainPerExt = 2` `[static]`

Periodic extension.

10.8.2.4 `const int SPL::ConvolveMode::sameDomainSymExt0 = 4` `[static]`

Symmetric periodic extension.

10.8.2.5 `const int SPL::ConvolveMode::sameDomainZeroExt = 1` `[static]`

The same as "same" in MATLAB.

The documentation for this struct was generated from the following file:

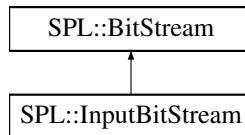
- [Sequence.hpp](#)

10.9 SPL::InputBitStream Class Reference

Input bit stream class.

```
#include <bitStream.hpp>
```

Inheritance diagram for SPL::InputBitStream:



Public Types

- typedef unsigned [loState](#)
The type used for the error state for a stream.
- typedef unsigned long long [Size](#)
An unsigned integral type (used for sizes/counts).
- typedef long long [Offset](#)
A signed integral type (used for differences).

Public Member Functions

- [InputBitStream](#) ()
Create a bit stream that is not initially bound to any (character) stream.
- [InputBitStream](#) (std::istream &in)
Create a bit stream that reads data from the specified (character) stream.
- [~InputBitStream](#) ()
Destroy a bit stream.
- std::istream * [getInput](#) () const
Get the (character) stream from which data is read.
- void [setInput](#) (std::istream *in)
Set the (character) stream from which data is read.
- [Offset](#) [getReadLimit](#) () const
Get the number of bits that still may be read from the bit stream before the read limit is reached.
- void [setReadLimit](#) ([Offset](#) readLimit)
Specify the maximum allowable number of bits that may be read from the bit stream.
- [Size](#) [getReadCount](#) () const

- Get the number of bits read from the bit stream so far.*

 - void `clearReadCount` ()

Set the read count to zero.
- long `getBits` (int numBits)

Read the specified number of bits from the bit stream.
- void `align` ()

Force byte-alignment of the bit stream.
- void `dump` (std::ostream &out) const

Dump the internal state of the bit stream to a (character) stream for debugging purposes.
- bool `isOk` () const

Test if the bitstream is in an okay (i.e., non-error) state.
- bool `isEof` () const

Test if the bitstream has encountered end-of-file (EOF).
- bool `isLimit` () const

Test if the bitstream has encountered a read/write limit.
- `loState` `getloState` () const

Get the I/O state of a bit stream.
- void `setloState` (`loState` state)

Set the I/O state of a bit stream.
- void `setloStateBits` (`loState` state)

Set the specified bits in the I/O state of a bit stream.
- void `clearloStateBits` (`loState` state=`allloBits`)

Clear the specified bits in the I/O state of a bit stream.

Static Public Attributes

- static const `loState` `eofBit` = 1

end of file (EOF) reached on input
- static const `loState` `limitBit` = 2

read/write limit exceeded
- static const `loState` `badBit` = 4

I/O error.
- static const `loState` `allloBits` = `eofBit` | `limitBit` | `badBit`

all error bits

10.9.1 Detailed Description

Input bit stream class.

10.9.2 Member Typedef Documentation

10.9.2.1 typedef unsigned SPL::BitStream::loState [inherited]

The type used for the error state for a stream.

10.9.2.2 `typedef long long SPL::BitStream::Offset` [inherited]

A signed integral type (used for differences).

10.9.2.3 `typedef unsigned long long SPL::BitStream::Size` [inherited]

An unsigned integral type (used for sizes/counts).

10.9.3 Constructor & Destructor Documentation

10.9.3.1 `SPL::InputBitStream::InputBitStream ()`

Create a bit stream that is not initially bound to any (character) stream.

10.9.3.2 `SPL::InputBitStream::InputBitStream (std::istream & in)`

Create a bit stream that reads data from the specified (character) stream.

10.9.3.3 `SPL::InputBitStream::~~InputBitStream ()`

Destroy a bit stream.

10.9.4 Member Function Documentation

10.9.4.1 `void SPL::InputBitStream::align ()`

Force byte-alignment of the bit stream.

The bit stream position is moved forward to the nearest byte (i.e., multiple of 8 bits) boundary.

10.9.4.2 `void SPL::InputBitStream::dump (std::ostream & out) const`

Dump the internal state of the bit stream to a (character) stream for debugging purposes.

10.9.4.3 `long SPL::InputBitStream::getBits (int numBits)`

Read the specified number of bits from the bit stream.

The bits that are read from the bit stream are assigned to the returned integer value in most-significant to least-significant order.

10.9.4.4 `std::istream* SPL::InputBitStream::getInput () const`

Get the (character) stream from which data is read.

10.9.4.5 `void SPL::InputBitStream::setInput (std::istream * in)`

Set the (character) stream from which data is read.

10.9.5 Member Data Documentation

10.9.5.1 `const IoState SPL::BitStream::alloBits = eofBit | limitBit | badBit` `[static], [inherited]`

all error bits

10.9.5.2 `const IoState SPL::BitStream::badBit = 4` `[static], [inherited]`

I/O error.

10.9.5.3 `const IoState SPL::BitStream::eofBit = 1` `[static], [inherited]`

end of file (EOF) reached on input

10.9.5.4 `const IoState SPL::BitStream::limitBit = 2` `[static], [inherited]`

read/write limit exceeded

The documentation for this class was generated from the following files:

- [bitStream.hpp](#)
- [bitStream.cpp](#)

10.10 SPL::MDecoder Class Reference

The M-Coder (binary) arithmetic decoder class.

```
#include <mCoder.hpp>
```

Inherits `SPL::MCoder`.

Public Member Functions

- [MDecoder](#) (int numContexts=0, [InputBitStream](#) *in=0)
Create a decoder with the specified number of contexts that reads input from the given bit stream.
- [~MDecoder](#) ()
Destroy a decoder.
- void [setNumContexts](#) (int numContexts)
Set the number of contexts.
- int [getNumContexts](#) () const
Get the number of contexts.
- void [setInput](#) ([InputBitStream](#) *in)
Set the input bit stream (i.e., the bit stream from which encoded data is to be read).
- [InputBitStream](#) * [getInput](#) () const
Get the input bit stream (i.e., the bit stream from which encoded data is to be read).
- void [clearContexts](#) ()
Clear the state of all of the contexts.
- long [getBitCount](#) () const

- *Get the number of bits read so far.*
• long `getSymCount` () const
- *Get the number of symbols decoded so far.*
• int `start` ()
- *Prepare to decode an arithmetic code word.*
• int `terminate` ()
- *Terminate the arithmetic code word.*
• int `decodeRegular` (int contextId)
- *Decode a symbol in the specified context.*
• int `decodeBypass` ()
- *Decode a symbol in bypass mode (i.e., assuming both symbols are equiprobable).*
• void `dump` (std::ostream &out) const
- *Dump the internal state information for the decoder to a stream (for debugging).*

Static Public Member Functions

- static void `setDebugLevel` (int debugLevel)
Set the debug level.
- static void `setDebugStream` (std::ostream &debugStream)
Set the stream to use for debugging output.
- static std::ostream & `getDebugStream` ()
Get the stream used for debugging output.

10.10.1 Detailed Description

The M-Coder (binary) arithmetic decoder class.

10.10.2 Constructor & Destructor Documentation

10.10.2.1 `SPL::MDecoder::MDecoder (int numContexts = 0, InputBitStream * in = 0)`

Create a decoder with the specified number of contexts that reads input from the given bit stream.

10.10.2.2 `SPL::MDecoder::~~MDecoder ()`

Destroy a decoder.

10.10.3 Member Function Documentation

10.10.3.1 `void SPL::MDecoder::clearContexts ()`

Clear the state of all of the contexts.

10.10.3.2 `int SPL::MDecoder::decodeBypass ()`

Decode a symbol in bypass mode (i.e., assuming both symbols are equiprobable).

10.10.3.3 `int SPL::MDecoder::decodeRegular (int contextId)`

Decode a symbol in the specified context.

10.10.3.4 `void SPL::MDecoder::dump (std::ostream & out) const`

Dump the internal state information for the decoder to a stream (for debugging).

10.10.3.5 `std::ostream & SPL::MDecoder::getDebugStream () [static]`

Get the stream used for debugging output.

10.10.3.6 `void SPL::MDecoder::setDebugLevel (int debugLevel) [static]`

Set the debug level.

10.10.3.7 `void SPL::MDecoder::setDebugStream (std::ostream & debugStream) [static]`

Set the stream to use for debugging output.

10.10.3.8 `void SPL::MDecoder::setNumContexts (int numContexts)`

Set the number of contexts.

10.10.3.9 `int SPL::MDecoder::start ()`

Prepare to decode an arithmetic code word.

Note: This function must be called before attempting to decode any symbols.

10.10.3.10 `int SPL::MDecoder::terminate ()`

Terminate the arithmetic code word.

The documentation for this class was generated from the following files:

- [mCoder.hpp](#)
- [mCoder.cpp](#)

10.11 **SPL::MEncoder Class Reference**

The M-Coder (binary) arithmetic encoder class.

```
#include <mCoder.hpp>
```

Inherits `SPL::MCoder`.

Public Member Functions

- **MEncoder** (int numContexts=0, **OutputBitStream** *out=0)
Create an encoder with a specified number of contexts that sends output to a given bit stream.
- **~MEncoder** ()
Destroy an encoder.
- void **setNumContexts** (int numContexts)
Set the number of contexts.
- int **getNumContexts** () const
Get the number of contexts.
- void **clearContexts** ()
Clear the state of all of the contexts.
- void **setOutput** (**OutputBitStream** *out)
Set the bit stream to use for output.
- **OutputBitStream** * **getOutput** () const
Get the bit stream being used for output.
- long **getSymCount** () const
Get the number of symbols that have been encoded so far.
- long **getBitCount** () const
Get the number of bits (of encoded data) that have been output to the underlying bit stream so far.
- void **start** ()
Start the arithmetic code word.
- int **terminate** ()
Terminate the arithmetic code word.
- int **encodeRegular** (int contextId, int binVal)
Encode a symbol in the specified context.
- int **encodeBypass** (int binVal)
Encode a symbol in bypass mode (i.e., assuming that both symbols are equiprobable).
- void **dump** (std::ostream &out) const
Dump the internal state of the encoder for debugging.

Static Public Member Functions

- static void **setDebugLevel** (int debugLevel)
Set the debug level.
- static void **setDebugStream** (std::ostream &debugStream)
Set the stream for debugging output.
- static std::ostream & **getDebugStream** ()
Get the stream used for debugging output.

10.11.1 Detailed Description

The M-Coder (binary) arithmetic encoder class.

10.11.2 Constructor & Destructor Documentation

10.11.2.1 `SPL::MEncoder::MEncoder (int numContexts = 0, OutputBitStream * out = 0)`

Create an encoder with a specified number of contexts that sends output to a given bit stream.

10.11.2.2 `SPL::MEncoder::~~MEncoder ()`

Destroy an encoder.

10.11.3 Member Function Documentation

10.11.3.1 `void SPL::MEncoder::clearContexts ()`

Clear the state of all of the contexts.

10.11.3.2 `void SPL::MEncoder::dump (std::ostream & out) const`

Dump the internal state of the encoder for debugging.

10.11.3.3 `int SPL::MEncoder::encodeBypass (int binVal)`

Encode a symbol in bypass mode (i.e., assuming that both symbols are equiprobable).

10.11.3.4 `int SPL::MEncoder::encodeRegular (int contextId, int binVal)`

Encode a symbol in the specified context.

The symbol *binVal* is encoded using context *contextId*.

10.11.3.5 `std::ostream & SPL::MEncoder::getDebugStream () [static]`

Get the stream used for debugging output.

10.11.3.6 `void SPL::MEncoder::setDebugLevel (int debugLevel) [static]`

Set the debug level.

10.11.3.7 `void SPL::MEncoder::setDebugStream (std::ostream & debugStream) [static]`

Set the stream for debugging output.

10.11.3.8 `void SPL::MEncoder::setNumContexts (int numContexts)`

Set the number of contexts.

10.11.3.9 void SPL::MEncoder::start ()

Start the arithmetic code word.

10.11.3.10 int SPL::MEncoder::terminate ()

Terminate the arithmetic code word.

The documentation for this class was generated from the following files:

- [mCoder.hpp](#)
- [mCoder.cpp](#)

10.12 SPL::MultiArithDecoder Class Reference

M-ary arithmetic decoder class.

```
#include <arithCoder.hpp>
```

Public Member Functions

- [MultiArithDecoder](#) (int maxContexts, [InputBitStream](#) *in=nullptr)
Create a decoder with the specified maximum number of contexts that sends output to the given bit stream.
- [~MultiArithDecoder](#) ()
Destroy the decoder.
- [InputBitStream](#) * [getInput](#) () const
Get the bit stream from which to read encoded data.
- void [setInput](#) ([InputBitStream](#) *in)
Set the bit stream from which to read encoded data.
- SPL_ArithCoder_ulong [getBitCount](#) () const
Get the number of bits read so far.
- SPL_ArithCoder_ulong [getSymCount](#) () const
Get the number of symbols decoded so far.
- int [getMaxContexts](#) () const
Get the maximum number of contexts.
- void [setContext](#) (int contextId, int numSyms)
Set the specified context to have the given number of symbols which are initially equiprobable.
- void [setContext](#) (int contextId, const std::vector< [ArithCoder::Freq](#) > &symFreqs, bool adaptive)
Set the specified context to have symbols with the given frequencies and the given adaptivity.
- int [start](#) ()
Start a code word.
- int [terminate](#) ()
Terminate a code word (for synchronization with encoder).
- int [decodeRegular](#) (int contextId)
Decode a symbol using the given context.
- int [decodeBypass](#) (int numSyms)
Decode a symbol in bypass mode (i.e., all symbols equiprobable).
- void [dump](#) (std::ostream &out) const
Dump the internal state of the decoder to the specified stream for debugging purposes.

Static Public Member Functions

- static void `setDebugLevel` (int debugLevel)
Set the debug level.
- static void `setDebugStream` (std::ostream &out)
Set the stream to use for debugging output.
- static std::ostream & `getDebugStream` ()
Get the stream used for debugging output.

10.12.1 Detailed Description

M-ary arithmetic decoder class.

10.12.2 Constructor & Destructor Documentation

10.12.2.1 `SPL::MultiArithDecoder::MultiArithDecoder (int maxContexts, InputBitStream * in = nullptr)`

Create a decoder with the specified maximum number of contexts that sends output to the given bit stream.

10.12.2.2 `SPL::MultiArithDecoder::~MultiArithDecoder ()`

Destroy the decoder.

10.12.3 Member Function Documentation

10.12.3.1 `int SPL::MultiArithDecoder::decodeBypass (int numSyms)`

Decode a symbol in bypass mode (i.e., all symbols equiprobable).

10.12.3.2 `int SPL::MultiArithDecoder::decodeRegular (int contextId)`

Decode a symbol using the given context.

10.12.3.3 `void SPL::MultiArithDecoder::dump (std::ostream & out) const`

Dump the internal state of the decoder to the specified stream for debugging purposes.

10.12.3.4 `void SPL::MultiArithDecoder::setContext (int contextId, int numSyms)`

Set the specified context to have the given number of symbols which are initially equiprobable.

10.12.3.5 `void SPL::MultiArithDecoder::setContext (int contextId, const std::vector< ArithCoder::Freq > & symFreqs, bool adaptive)`

Set the specified context to have symbols with the given frequencies and the given adaptivity.

10.12.3.6 int SPL::MultiArithDecoder::start ()

Start a code word.

This function must be called before attempting to decode any symbols.

10.12.3.7 int SPL::MultiArithDecoder::terminate ()

Terminate a code word (for synchronization with encoder).

The documentation for this class was generated from the following files:

- include/SPL/arithCoder.hpp
- arithCoder.cpp

10.13 SPL::MultiArithEncoder Class Reference

M-ary arithmetic encoder class.

```
#include <arithCoder.hpp>
```

Public Member Functions

- [MultiArithEncoder](#) (int maxContexts, [OutputBitStream](#) *out=nullptr)
Create an encoder with the specified number of contexts that sends output to the given bit stream.
- [~MultiArithEncoder](#) ()
Destroy an encoder.
- [OutputBitStream](#) * [getOutput](#) ()
Get the bit stream used for output.
- void [setOutput](#) ([OutputBitStream](#) *out)
Set the bit stream used for output.
- SPL_ArithCoder_ulong [getSymCount](#) () const
Get the number of symbols encoded so far.
- SPL_ArithCoder_ulong [getBitCount](#) () const
Get the number of bits of output generated so far including bits awaiting output.
- int [getMaxContexts](#) () const
Get the maximum number of contexts.
- void [setContext](#) (int contextId, int numSyms)
Set the specified context to have the given number of symbols which are initially equiprobable.
- void [setContext](#) (int contextId, const std::vector< ArithCoder::Freq > &symFreqs, bool adaptive)
Set the specified context to have symbols with the given frequencies and the given adaptivity.
- int [start](#) ()
Start a code word.
- int [terminate](#) ()
Terminate the code word.
- int [encodeRegular](#) (int contextId, int sym)
Encode the given symbol in the specified context.
- int [encodeBypass](#) (int numSyms, int sym)

Encode the given symbol in bypass mode (i.e., a fixed probability distribution where all symbols are equiprobable).

- void `dump` (std::ostream &out) const

Dump the internal state of the encoder to the specified stream for debugging purposes.

Static Public Member Functions

- static void `setDebugLevel` (int debugLevel)
Set the debug level.
- static void `setDebugStream` (std::ostream &out)
Set the stream for debugging output.
- static std::ostream & `getDebugStream` ()
Get the stream for debugging output.

10.13.1 Detailed Description

M-ary arithmetic encoder class.

10.13.2 Constructor & Destructor Documentation

10.13.2.1 `SPL::MultiArithEncoder::MultiArithEncoder (int maxContexts, OutputBitStream * out = nullptr)`

Create an encoder with the specified number of contexts that sends output to the given bit stream.

10.13.2.2 `SPL::MultiArithEncoder::~~MultiArithEncoder ()`

Destroy an encoder.

10.13.3 Member Function Documentation

10.13.3.1 `void SPL::MultiArithEncoder::dump (std::ostream & out) const`

Dump the internal state of the encoder to the specified stream for debugging purposes.

10.13.3.2 `int SPL::MultiArithEncoder::encodeBypass (int numSyms, int sym)`

Encode the given symbol in bypass mode (i.e., a fixed probability distribution where all symbols are equiprobable).

10.13.3.3 `int SPL::MultiArithEncoder::encodeRegular (int contextId, int sym)`

Encode the given symbol in the specified context.

10.13.3.4 `void SPL::MultiArithEncoder::setContext (int contextId, int numSyms)`

Set the specified context to have the given number of symbols which are initially equiprobable.

10.13.3.5 void SPL::MultiArithEncoder::setContext (int *contextId*, const std::vector< ArithCoder::Freq > & *symFreqs*, bool *adaptive*)

Set the specified context to have symbols with the given frequencies and the given adaptivity.

10.13.3.6 int SPL::MultiArithEncoder::start ()

Start a code word.

This function must be called before attempting to encode any symbols.

10.13.3.7 int SPL::MultiArithEncoder::terminate ()

Terminate the code word.

The documentation for this class was generated from the following files:

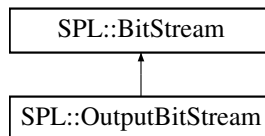
- include/SPL/arithCoder.hpp
- arithCoder.cpp

10.14 SPL::OutputBitStream Class Reference

Output bit stream class.

```
#include <bitStream.hpp>
```

Inheritance diagram for SPL::OutputBitStream:



Public Types

- typedef unsigned [loState](#)
The type used for the error state for a stream.
- typedef unsigned long long [Size](#)
An unsigned integral type (used for sizes/counts).
- typedef long long [Offset](#)
A signed integral type (used for differences).

Public Member Functions

- [OutputBitStream](#) ()
Create a bit stream that is not initially bound to any (character) stream.
- [OutputBitStream](#) (std::ostream &out)
Create a bit stream that sends its output to the specified (character) stream.

- `~OutputStream ()`
Destroy a bit stream.
- `std::ostream * getOutput () const`
Get the output (character) stream associated with the bit stream.
- `void setOutput (std::ostream *out)`
Set the output (character) stream associated with the bit stream.
- `void clearWriteCount ()`
Clear the count of the number of bits written to the bit stream.
- `Size getWriteCount () const`
Get the number of bits written to the bit stream.
- `void setWriteLimit (Offset writeLimit)`
Set the number of bits that may still be written to the bit stream.
- `Offset getWriteLimit () const`
Get the number of bits that may still be written to the underlying (character) stream.
- `int putBits (long data, int numBits)`
Output the specified number of bits to the bit stream.
- `void align ()`
Align the bit stream output position to the nearest byte boundary.
- `void flush ()`
Flush any pending output to the underlying (character) stream.
- `void dump (std::ostream &out) const`
Dump the internal state of the bit stream to the specified (character) stream for debugging purposes.
- `bool isOkay () const`
Test if the bitstream in an okay (i.e., non-error) state.
- `bool isEof () const`
Test if the bitstream has encountered end-of-file (EOF).
- `bool isLimit () const`
Test if the bitstream has encountered a read/write limit.
- `loState getloState () const`
Get the I/O state of a bit stream.
- `void setloState (loState state)`
Set the I/O state of a bit stream.
- `void setloStateBits (loState state)`
Set the specified bits in the I/O state of a bit stream.
- `void clearloStateBits (loState state=allloBits)`
Clear the specified bits in the I/O state of a bit stream.

Static Public Attributes

- static const `loState eofBit = 1`
end of file (EOF) reached on input
- static const `loState limitBit = 2`
read/write limit exceeded
- static const `loState badBit = 4`
I/O error.
- static const `loState allloBits = eofBit | limitBit | badBit`
all error bits

10.14.1 Detailed Description

Output bit stream class.

10.14.2 Member Typedef Documentation

10.14.2.1 `typedef unsigned SPL::BitStream::loState` [inherited]

The type used for the error state for a stream.

10.14.2.2 `typedef long long SPL::BitStream::Offset` [inherited]

A signed integral type (used for differences).

10.14.2.3 `typedef unsigned long long SPL::BitStream::Size` [inherited]

An unsigned integral type (used for sizes/counts).

10.14.3 Constructor & Destructor Documentation

10.14.3.1 `SPL::OutputBitStream::OutputBitStream ()`

Create a bit stream that is not initially bound to any (character) stream.

10.14.3.2 `SPL::OutputBitStream::OutputBitStream (std::ostream & out)`

Create a bit stream that sends its output to the specified (character) stream.

10.14.3.3 `SPL::OutputBitStream::~~OutputBitStream ()`

Destroy a bit stream.

10.14.4 Member Function Documentation

10.14.4.1 `void SPL::OutputBitStream::align ()`

Align the bit stream output position to the nearest byte boundary.

10.14.4.2 `void SPL::OutputBitStream::dump (std::ostream & out) const`

Dump the internal state of the bit stream to the specified (character) stream for debugging purposes.

10.14.4.3 `void SPL::OutputBitStream::flush ()`

Flush any pending output to the underlying (character) stream.

The bit stream is aligned to the nearest byte boundary and any pending output is flushed to the underlying (character) stream.

10.14.4.4 `std::ostream * SPL::OutputBitStream::getOutput () const`

Get the output (character) stream associated with the bit stream.

10.14.4.5 `int SPL::OutputBitStream::putBits (long data, int numBits)`

Output the specified number of bits to the bit stream.

This function returns a nonnegative value upon success and a negative value if an error is encountered.

10.14.4.6 `void SPL::OutputBitStream::setOutput (std::ostream * out)`

Set the output (character) stream associated with the bit stream.

10.14.5 Member Data Documentation

10.14.5.1 `const loState SPL::BitStream::allIoBits = eofBit | limitBit | badBit` `[static], [inherited]`

all error bits

10.14.5.2 `const loState SPL::BitStream::badBit = 4` `[static], [inherited]`

I/O error.

10.14.5.3 `const loState SPL::BitStream::eofBit = 1` `[static], [inherited]`

end of file (EOF) reached on input

10.14.5.4 `const loState SPL::BitStream::limitBit = 2` `[static], [inherited]`

read/write limit exceeded

The documentation for this class was generated from the following files:

- [bitStream.hpp](#)
- [bitStream.cpp](#)

10.15 SPL::PnmHeader Struct Reference

The header information for PNM data.

```
#include <pnmCodec.hpp>
```


Public Attributes

- PnmMagic [magic](#)
The magic number.
- int [width](#)
The image width.
- int [height](#)
The image height.
- int [maxVal](#)
The maximum sample value.
- bool [sgnd](#)
The signedness of the sample data.

10.15.1 Detailed Description

The header information for PNM data.

10.15.2 Member Data Documentation

10.15.2.1 int SPL::PnmHeader::height

The image height.

10.15.2.2 PnmMagic SPL::PnmHeader::magic

The magic number.

10.15.2.3 int SPL::PnmHeader::maxVal

The maximum sample value.

10.15.2.4 bool SPL::PnmHeader::sgnd

The signedness of the sample data.

10.15.2.5 int SPL::PnmHeader::width

The image width.

The documentation for this struct was generated from the following file:

- [pnmCodec.hpp](#)

10.16 SPL::Quaternion< T > Struct Template Reference

A quaternion represented in terms of its scalar and vector parts.

```
#include <cgalUtil.hpp>
```

Public Types

- `typedef T::FT` [Real](#)
The field type for the CGAL kernel.
- `typedef CGAL::Vector_3< T >` [Vector_3](#)
The 3-dimensional vector type.

Public Member Functions

- [Quaternion](#) ()
- [Quaternion](#) ([Real](#) scalar_, const [Vector_3](#) &vector_)

Public Attributes

- [Real](#) scalar
The scalar part of the quaternion.
- [Vector_3](#) vector
The vector part of the quaternion.

10.16.1 Detailed Description

```
template<class T>struct SPL::Quaternion< T >
```

A quaternion represented in terms of its scalar and vector parts.

10.16.2 Member Typedef Documentation

10.16.2.1 `template<class T> typedef T::FT` [SPL::Quaternion< T >::Real](#)

The field type for the CGAL kernel.

10.16.2.2 `template<class T> typedef CGAL::Vector_3<T>` [SPL::Quaternion< T >::Vector_3](#)

The 3-dimensional vector type.

10.16.3 Constructor & Destructor Documentation

10.16.3.1 `template<class T> SPL::Quaternion< T >::Quaternion ()` `[inline]`

Create a quaternion.

10.16.3.2 `template<class T> SPL::Quaternion< T >::Quaternion (Real scalar_, const Vector_3 &vector_)`
`[inline]`

Create a quaternion with the specified scalar and vector parts.

10.16.4 Member Data Documentation

10.16.4.1 `template<class T> Real SPL::Quaternion< T >::scalar`

The scalar part of the quaternion.

10.16.4.2 `template<class T> Vector_3 SPL::Quaternion< T >::vector`

The vector part of the quaternion.

The documentation for this struct was generated from the following file:

- [cgalUtil.hpp](#)

10.17 SPL::Rotation_3< T > Struct Template Reference

A 3-D rotation.

```
#include <cgalUtil.hpp>
```

Public Types

- `typedef T::FT Real`
The field type for the CGAL kernel.
- `typedef T::Vector_3 Vector_3`
The 3-dimensional vector type.

Public Member Functions

- `Rotation_3 (const Vector_3 &axis_, Real angle_)`
Create a rotation.

Public Attributes

- `Vector_3 axis`
The axis of rotation.
- `Real angle`
The angle of rotation.

10.17.1 Detailed Description

```
template<class T>struct SPL::Rotation_3< T >
```

A 3-D rotation.

10.17.2 Member Typedef Documentation

10.17.2.1 `template<class T> typedef T::FT SPL::Rotation_3< T >::Real`

The field type for the CGAL kernel.

10.17.2.2 `template<class T> typedef T::Vector_3 SPL::Rotation_3< T >::Vector_3`

The 3-dimensional vector type.

10.17.3 Constructor & Destructor Documentation

10.17.3.1 `template<class T> SPL::Rotation_3< T >::Rotation_3 (const Vector_3 & axis_, Real angle_) [inline]`

Create a rotation.

10.17.4 Member Data Documentation

10.17.4.1 `template<class T> Real SPL::Rotation_3< T >::angle`

The angle of rotation.

10.17.4.2 `template<class T> Vector_3 SPL::Rotation_3< T >::axis`

The axis of rotation.

The documentation for this struct was generated from the following file:

- [cgalUtil.hpp](#)

10.18 SPL::Sequence1< T > Class Template Reference

A one-dimensional sequence class with lazy copying and reference counting.

```
#include <Sequence1.hpp>
```

Public Types

- typedef T [ElemType](#)
The type of the element in the sequence.
- typedef [Array1](#)< T >::ConstIterator [ConstIterator](#)
The const iterator for the sequence.
- typedef [Array1](#)< T >::Iterator [Iterator](#)
The mutable iterator for the sequence.

Public Member Functions

- [Sequence1](#) ()
The default constructor.
- [Sequence1](#) (int startInd, int size)
Construct a sequence with the specified start index and size.
- [Sequence1](#) (int startInd, int size, const T &value)
Construct a sequence with the specified start index and size, with all elements set to the given value.
- template<class InputIterator >
[Sequence1](#) (int startInd, int size, InputIterator data)
Construct a sequence with the specified start index and size, with the elements initialized to the data read from the given iterator.
- [Sequence1](#) (const [Sequence1](#) &f)
The copy constructor.
- template<class OtherT >
[Sequence1](#) (const [Sequence1](#)< OtherT > &f)
Create a sequence from another sequence having elements of a different type.
- [Sequence1](#) (const [Array1](#)< T > &data)
Create a sequence from an array.
- [Sequence1](#) (int startInd, const [Array1](#)< T > &data)
Create a sequence from an array using the given starting index.
- [~Sequence1](#) ()
The destructor.
- [Sequence1](#) & operator= (const [Sequence1](#) &f)
The assignment operator.
- template<class OtherT >
[Sequence1](#) & operator= (const [Sequence1](#)< OtherT > &f)
Assign another sequence with elements of a different type to this sequence.
- [Sequence1](#) & operator+= (const [Sequence1](#) &f)
Add another sequence to this one.
- [Sequence1](#) & operator-= (const [Sequence1](#) &f)
Subtract another sequence from this one.
- [Sequence1](#) & operator*= (const [Sequence1](#) &f)
Multiply elementwise this sequence by another one.
- [Sequence1](#) & operator/= (const [Sequence1](#) &f)
Divide elementwise this sequence by another one.
- [Sequence1](#) & operator+= (const T &value)
Add a value to each element of this sequence.
- [Sequence1](#) & operator-= (const T &value)
Subtract a value from each element of this sequence.
- [Sequence1](#) & operator*= (const T &value)
Multiply each element of this sequence by the specified value.
- [Sequence1](#) & operator/= (const T &value)
Divide each element of the sequence by the given value.
- int [getStartInd](#) () const
Get the start index for the sequence.
- int [getEndInd](#) () const
Get the end index for the sequence.

- `int getSize () const`
Get the length of the sequence.
- `bool isShared () const`
Is the array for this sequence shared with another array?
- `const T & operator() (int i) const`
Get the specified element in the sequence.
- `T & operator() (int i)`
Get the specified element in the sequence.
- `ConstIterator begin () const`
Get an iterator referencing the first element in the sequence.
- `Iterator begin ()`
Get an iterator referencing the first element in the sequence.
- `ConstIterator end () const`
Get an iterator referencing just after the last element in the sequence.
- `Iterator end ()`
Get an iterator referencing just after the last element in the sequence.
- `T min () const`
Get the minimum element in the sequence.
- `T max () const`
Get the maximum element in the sequence.
- `T sum () const`
Get the sum of the elements in the sequence.
- `Array1< T > getArray () const`
Get a copy of the underlying array.
- `void swapArray (Array1< T > &data)`
Swap the data for the underlying array and the specified array.
- `void fill (const T &value)`
Set all of the elements in the sequence to the specified value.
- `Sequence1 & translate (int delta)`
Translate (i.e., shift) a sequence by the specified displacement.

10.18.1 Detailed Description

`template<class T>class SPL::Sequence1< T >`

A one-dimensional sequence class with lazy copying and reference counting.

10.18.2 Member Typedef Documentation

10.18.2.1 `template<class T> typedef Array1<T>::ConstIterator SPL::Sequence1< T >::ConstIterator`

The const iterator for the sequence.

10.18.2.2 `template<class T> typedef T SPL::Sequence1< T >::ElemType`

The type of the element in the sequence.

10.18.2.3 `template<class T> typedef Array1<T>::Iterator SPL::Sequence1< T >::Iterator`

The mutable iterator for the sequence.

10.18.3 Constructor & Destructor Documentation

10.18.3.1 `template<class T> template<class InputIterator > SPL::Sequence1< T >::Sequence1 (int startInd, int size, InputIterator data)`

Construct a sequence with the specified start index and size, with the elements initialized to the data read from the given iterator.

Effects: A sequence with a starting index of startInd and size size is created, with the elements being initialized by the data pointed to by data.

10.18.3.2 `template<class T> template<class OtherT > SPL::Sequence1< T >::Sequence1 (const Sequence1< OtherT > & f)`

Create a sequence from another sequence having elements of a different type.

10.18.4 Member Function Documentation

10.18.4.1 `template<class T> template<class OtherT > Sequence1& SPL::Sequence1< T >::operator= (const Sequence1< OtherT > & f)`

Assign another sequence with elements of a different type to this sequence.

The type OtherT must be assignable to the type T.

The documentation for this class was generated from the following file:

- [Sequence1.hpp](#)

10.19 SPL::Sequence2< T > Class Template Reference

A two-dimensional sequence class with lazy copying and reference counting.

```
#include <Sequence2.hpp>
```

Public Types

- typedef T [ElemType](#)
The type of the element in the sequence.
- typedef [Array2< T >::ConstIterator](#) [ConstIterator](#)
The const iterator for all elements in the sequence.
- typedef [Array2< T >::Iterator](#) [Iterator](#)
The mutable iterator for all elements in the sequence.
- typedef [Array2< T >::ConstXIterator](#) [ConstXIterator](#)
The const iterator for the elements in a row of the sequence.

- typedef [Array2](#)< T >::XIterator XIterator
The mutable iterator for the elements in a row of the sequence.
- typedef [Array2](#)< T >::ConstYIterator ConstYIterator
The const iterator for the elements in a column of the sequence.
- typedef [Array2](#)< T >::YIterator YIterator
The mutable iterator for the elements in a column of the sequence.

Public Member Functions

- [Sequence2](#) ()
The default constructor.
- [Sequence2](#) (int startX, int startY, int width, int height)
Construct a sequence with the specified start index and size.
- [Sequence2](#) (int startX, int startY, int width, int height, const T &data)
Construct a sequence with the specified start index and size, with all elements set to the given value.
- template<class InputIterator >
[Sequence2](#) (int startX, int startY, int width, int height, InputIterator data)
Construct a sequence with the specified start index and size, with the elements initialized to the data read from the given iterator.
- [Sequence2](#) (const [Sequence2](#) &f)
The copy constructor.
- template<class OtherT >
[Sequence2](#) (const [Sequence2](#)< OtherT > &f)
Create a sequence from another sequence having elements of a different type.
- [Sequence2](#) (const [Array2](#)< T > &data)
Create a sequence from an array.
- [Sequence2](#) (int startX, int startY, const [Array2](#)< T > &data)
Create a sequence from an array using the given starting index.
- ~[Sequence2](#) ()
The destructor.
- [Sequence2](#) & operator= (const [Sequence2](#) &f)
The assignment operator.
- template<class OtherT >
[Sequence2](#) & operator= (const [Sequence2](#)< OtherT > &f)
Assign another sequence with elements of a different type to this sequence.
- [Sequence2](#) & operator+= (const [Sequence2](#) &f)
Add another sequence to this one.
- [Sequence2](#) & operator-= (const [Sequence2](#) &f)
Subtract another sequence from this one.
- [Sequence2](#) & operator*= (const [Sequence2](#) &f)
Multiply elementwise this sequence by another one.
- [Sequence2](#) & operator/= (const [Sequence2](#) &f)
Divide elementwise this sequence by another one.
- [Sequence2](#) & operator+= (const T &value)
Add a value to each element of this sequence.
- [Sequence2](#) & operator-= (const T &value)
Subtract a value from each element of this sequence.

- [Sequence2](#) & [operator*=\(const T &value\)](#)
Multiply each element of this sequence by the specified value.
- [Sequence2](#) & [operator/=\(const T &value\)](#)
Divide each element of the sequence by the given value.
- [int](#) [getStartX\(\)](#) [const](#)
Get the x-coordinate of the start index for the sequence.
- [int](#) [getStartY\(\)](#) [const](#)
Get the y-coordinate of the start index for the sequence.
- [int](#) [getEndX\(\)](#) [const](#)
Get the x-coordinate of the end index for the sequence.
- [int](#) [getEndY\(\)](#) [const](#)
Get the y-coordinate of the end index for the sequence.
- [int](#) [getWidth\(\)](#) [const](#)
Get the width of the sequence.
- [int](#) [getHeight\(\)](#) [const](#)
Get the height of the sequence.
- [int](#) [getSize\(\)](#) [const](#)
Get the number of elements in the sequence.
- [bool](#) [isShared\(\)](#) [const](#)
Is the array for this sequence shared with another array?
- [T](#) & [operator\(\)\(int x, int y\)](#)
Get a mutable reference to the specified element in the sequence.
- [const T](#) & [operator\(\)\(int x, int y\)](#) [const](#)
Get a const reference to the specified element in the sequence.
- [ConstIterator](#) [begin\(\)](#) [const](#)
Get a const iterator for the first element in the sequence.
- [Iterator](#) [begin\(\)](#)
Get a mutable iterator for the first element in the sequence.
- [ConstIterator](#) [end\(\)](#) [const](#)
Get a const iterator for one past the last element in the sequence.
- [Iterator](#) [end\(\)](#)
Get a mutable iterator for one past the last element in the sequence.
- [ConstXIterator](#) [rowBegin\(int y\)](#) [const](#)
Get a const iterator for the first element in the specified row of the sequence.
- [XIterator](#) [rowBegin\(int y\)](#)
Get a mutable iterator for the first element in the specified row of the sequence.
- [ConstXIterator](#) [rowEnd\(int y\)](#) [const](#)
Get a const iterator for one past the end in the specified row of the sequence.
- [XIterator](#) [rowEnd\(int y\)](#)
Get a mutable iterator for one past the end in the specified row of the sequence.
- [ConstYIterator](#) [colBegin\(int x\)](#) [const](#)
Get a const iterator for the first element in the specified column of the sequence.
- [YIterator](#) [colBegin\(int x\)](#)
Get a mutable iterator for the first element in the specified column of the sequence.
- [ConstYIterator](#) [colEnd\(int x\)](#) [const](#)
Get a const iterator for one past the end in the specified column of the sequence.
- [YIterator](#) [colEnd\(int x\)](#)

Get a mutable iterator for one past the end in the specified column of the sequence.

- `T min () const`

Get the minimum element in the sequence.

- `T max () const`

Get the maximum element in the sequence.

- `T sum () const`

Get the sum of the elements in the sequence.

- `std::ostream & output (std::ostream &out, int fieldWidth) const`

Output a sequence to the specified stream using the given field width for each sequence element.

- `Array2< T > getArray () const`

Get a copy of the underlying array.

- `void swapArray (Array2< T > &data)`

Swap the data for the underlying array and the specified array.

- `void fill (const T &value)`

Get a copy of the underlying array.

- `Sequence2 & translate (int x, int y)`

Translate (i.e., shift) a sequence by the specified displacement.

10.19.1 Detailed Description

```
template<class T>class SPL::Sequence2< T >
```

A two-dimensional sequence class with lazy copying and reference counting.

10.19.2 Member Typedef Documentation

10.19.2.1 `template<class T> typedef Array2<T>::ConstIterator SPL::Sequence2< T >::ConstIterator`

The const iterator for all elements in the sequence.

10.19.2.2 `template<class T> typedef Array2<T>::ConstXIterator SPL::Sequence2< T >::ConstXIterator`

The const iterator for the elements in a row of the sequence.

10.19.2.3 `template<class T> typedef Array2<T>::ConstYIterator SPL::Sequence2< T >::ConstYIterator`

The const iterator for the elements in a column of the sequence.

10.19.2.4 `template<class T> typedef T SPL::Sequence2< T >::ElemType`

The type of the element in the sequence.

10.19.2.5 `template<class T> typedef Array2<T>::Iterator SPL::Sequence2< T >::Iterator`

The mutable iterator for all elements in the sequence.

10.19.2.6 `template<class T> typedef Array2<T>::XIterator SPL::Sequence2< T >::XIterator`

The mutable iterator for the elements in a row of the sequence.

10.19.2.7 `template<class T> typedef Array2<T>::YIterator SPL::Sequence2< T >::YIterator`

The mutable iterator for the elements in a column of the sequence.

10.19.3 Constructor & Destructor Documentation

10.19.3.1 `template<class T> template<class InputIterator > SPL::Sequence2< T >::Sequence2 (int startX, int startY, int width, int height, InputIterator data)`

Construct a sequence with the specified start index and size, with the elements initialized to the data read from the given iterator.

10.19.3.2 `template<class T> template<class OtherT > SPL::Sequence2< T >::Sequence2 (const Sequence2< OtherT > & f)`

Create a sequence from another sequence having elements of a different type.

10.19.4 Member Function Documentation

10.19.4.1 `template<class T> template<class OtherT > Sequence2& SPL::Sequence2< T >::operator= (const Sequence2< OtherT > & f)`

Assign another sequence with elements of a different type to this sequence.

The type OtherT must be assignable to the type T.

The documentation for this class was generated from the following file:

- [Sequence2.hpp](#)

10.20 SPL::Timer Class Reference

A class for making timing measurements.

```
#include <Timer.hpp>
```

Public Member Functions

- void [start](#) ()
Start the timer.
- void [stop](#) ()
Stop the timer.
- double [get](#) () const
Get the timer value.

10.20.1 Detailed Description

A class for making timing measurements.

10.20.2 Member Function Documentation

10.20.2.1 `double SPL::Timer::get () const`

Get the timer value.

Effects: Query the elapsed time measured by the timer.

Returns: The elapsed time in seconds is returned. The resolution of the timer depends on the particular platform (e.g., operating system, hardware, etc.). For most mainstream platforms, the resolution of the timer is typically microseconds.

10.20.2.2 `void SPL::Timer::start ()`

Start the timer.

Effects: Starts the timer. The timer should not already be running.

10.20.2.3 `void SPL::Timer::stop ()`

Stop the timer.

Effects: Stops the timer. The timer should already be running.

The documentation for this class was generated from the following file:

- [Timer.hpp](#)

Chapter 11

File Documentation

11.1 Arcball.hpp File Reference

This file contains the Arcball class and related code.

```
#include <SPL/config.hpp>
#include <CGAL/Plane_3.h>
#include <CGAL/Ray_3.h>
#include "cgalUtil.hpp"
```

Classes

- class [SPL::Arcball< T >](#)
Arcball.

Functions

- `template<class T >`
`T::Point_3 SPL::closestPointOnRay (const typename CGAL::Point_3< T > &rayOrigin, const typename CGAL::Vector_3< T > &rayDir, const typename CGAL::Point_3< T > &point)`
Compute the closest point on a ray to the specified point.
- `template<class T >`
`std::pair< bool, typename T::Point_3 > SPL::findRaySphereIntersection (const typename CGAL::Point_3< T > &sphereCenter, typename T::FT sphereRadius, const typename CGAL::Point_3< T > &rayOrigin, const typename CGAL::Vector_3< T > &rayDir)`
Compute the intersection of a ray and a sphere.
- `template<class T >`
`std::pair< bool, typename T::Point_3 > SPL::findRayPlaneIntersection (const typename CGAL::Point_3< T > &planePoint, const typename CGAL::Vector_3< T > &planeNormal, const typename CGAL::Point_3< T > &rayOrigin, const typename CGAL::Vector_3< T > &rayDir)`
Compute the intersection of a ray and a plane.

11.1.1 Detailed Description

This file contains the Arcball class and related code.

11.2 Array1.hpp File Reference

This file contains the Array1 template class and supporting code.

```
#include <SPL/config.hpp>
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <vector>
#include <cassert>
#include <algorithm>
#include <functional>
#include <iterator>
#include <numeric>
#include <SPL/misc.hpp>
```

Classes

- class [SPL::Array1< T >](#)
A one-dimensional array class with lazy copying and reference counting.
- class [SPL::Array1< T >](#)
A one-dimensional array class with lazy copying and reference counting.

Macros

- `#define SPL_ARRAY1_INLINE inline`
Defining this symbol will enable extra code for debugging.

Typedefs

- `typedef Array1< double > SPL::RealArray1`
A one-dimensional array with real elements.
- `typedef Array1< int > SPL::IntArray1`
A one-dimensional array with integer elements.

Functions

- `template<class T >`
`std::ostream & SPL::operator<< (std::ostream &out, const Array1< T > &a)`
Output an array to the specified stream.
- `template<class T >`
`std::istream & SPL::operator>> (std::istream &in, Array1< T > &a)`
Input an array from the specified stream.
- `template<class T >`
`bool SPL::operator== (const Array1< T > &a, const Array1< T > &b)`
Test two arrays for equality.

- `template<class T >`
`SPL_ARRAY1_INLINE` `bool SPL::operator!= (const Array1< T > &a, const Array1< T > &b)`
Test two arrays for inequality.

11.2.1 Detailed Description

This file contains the Array1 template class and supporting code.

11.2.2 Macro Definition Documentation

11.2.2.1 `#define SPL_ARRAY1_INLINE` inline

Defining this symbol will enable extra code for debugging.

Allow the inlining of functions.

11.3 Array2.hpp File Reference

This file contains the Array2 template class and its supporting code.

```
#include <SPL/config.hpp>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <vector>
#include <cassert>
#include <iterator>
#include <algorithm>
#include <numeric>
#include <boost/iterator/iterator_facade.hpp>
#include <SPL/pnmCodec.hpp>
#include <SPL/misc.hpp>
```

Classes

- class `SPL::Array2< T >`
A two-dimensional array class with lazy copying and reference counting.
- class `SPL::Array2< T >`
A two-dimensional array class with lazy copying and reference counting.

Macros

- `#define SPL_ARRAY2_INLINE` inline
Defining this symbol will enable extra code for debugging.

Typedefs

- typedef Array2< double > [SPL::RealArray2](#)
A two-dimensional array with real elements.
- typedef Array2< int > [SPL::IntArray2](#)
A two-dimensional array with integer elements.

Functions

- template<class T >
std::ostream & [SPL::operator<<](#) (std::ostream &out, const Array2< T > &a)
Output an array to the specified stream.
- template<class T >
std::istream & [SPL::operator>>](#) (std::istream &in, Array2< T > &a)
Input an array from the specified stream.
- template<class T >
Array2< T > [SPL::transpose](#) (const Array2< T > &a)
Get the transpose of the array.
- template<class T >
bool [SPL::operator==](#) (const Array2< T > &a, const Array2< T > &b)
Test two arrays for equality.
- template<class T >
bool [SPL::operator!=](#) (const Array2< T > &a, const Array2< T > &b)
Test two arrays for inequality.
- template<class T >
int [SPL::encodePnm](#) (std::ostream &outStream, const std::vector< Array2< T > > &comps, int maxVal, bool sgnd, bool binaryFormat=true)
Output the array as an image in the PNM format.
- template<class T >
int [SPL::encodePbm](#) (std::ostream &outStream, const Array2< T > &bits, bool binaryFormat=true)
Output the array as an image in the PNM format (PBM type).
- template<class T >
int [SPL::encodePgm](#) (std::ostream &outStream, const Array2< T > &gray, int maxVal, bool sgnd, bool binaryFormat=true)
Output the array as an image in the PNM format (PGM type).
- template<class T >
int [SPL::encodePpm](#) (std::ostream &outStream, const Array2< T > &red, const Array2< T > &green, const Array2< T > &blue, int maxVal, bool sgnd, bool binaryFormat=true)
Output the array as an image in the PNM format (PPM type).
- template<class T >
int [SPL::decodePnm](#) (std::istream &inStream, std::vector< Array2< T > > &comps, int &maxVal, bool &sgnd)
Input an array as an image in the PNM format.
- template<class T >
int [SPL::decodePbm](#) (std::istream &inStream, Array2< T > &bits)
Input an array as an image in the PNM format.
- template<class T >
int [SPL::decodePgm](#) (std::istream &inStream, Array2< T > &gray, int &maxVal, bool &sgnd)
Input an array as an image in the PNM format.

- `template<class T >`
`int SPL::decodePpm (std::istream &inStream, Array2< T > &red, Array2< T > &green, Array2< T > &blue, int &maxVal, bool &sgnd)`

Input an array as an image in the PNM format.

11.3.1 Detailed Description

This file contains the Array2 template class and its supporting code.

11.3.2 Macro Definition Documentation

11.3.2.1 `#define SPL_ARRAY2_INLINE inline`

Defining this symbol will enable extra code for debugging.

Allow the inlining of functions.

11.4 audioFile.hpp File Reference

This file contains code for performing reading and writing of audio files in WAV format.

```
#include <SPL/config.hpp>
#include <iostream>
#include <string>
#include <algorithm>
#include <SPL/Array1.hpp>
```

Functions

- `int SPL::loadAudioFile (const std::string &fileName, int &samplingRate, RealArray1 &samples)`
Read audio data from a file in WAV format.
- `int SPL::saveAudioFile (const std::string &fileName, int samplingRate, const RealArray1 &samples)`
Write a sequence to a file in WAV format.

11.4.1 Detailed Description

This file contains code for performing reading and writing of audio files in WAV format.

11.5 bitStream.hpp File Reference

Bit Stream Classes.

```
#include <SPL/config.hpp>
#include <cassert>
#include <iostream>
```

Classes

- class [SPL::BitStream](#)
A common base class for the input and output bit stream classes.
- class [SPL::InputBitStream](#)
Input bit stream class.
- class [SPL::OutputBitStream](#)
Output bit stream class.

11.5.1 Detailed Description

Bit Stream Classes.

11.6 cgalUtil.hpp File Reference

This file contains various CGAL utility code.

```
#include <SPL/config.hpp>
#include <SPL/math.hpp>
#include <cmath>
#include <CGAL/Cartesian.h>
#include <CGAL/Vector_3.h>
#include <CGAL/Point_3.h>
```

Classes

- struct [SPL::Rotation_3< T >](#)
A 3-D rotation.
- struct [SPL::Quaternion< T >](#)
A quaternion represented in terms of its scalar and vector parts.

Functions

- template<class T >
T::FT [SPL::norm](#) (const typename CGAL::Vector_3< T > &v)
Compute the norm of a vector.
- template<class T >
T::Vector_3 [SPL::normalize](#) (const typename CGAL::Vector_3< T > &v)
Compute a unit vector.
- template<class T >
T::FT [SPL::angleBetweenVectors](#) (const typename CGAL::Vector_3< T > &u, const CGAL::Vector_3< T > &v)
Compute the angle between two vectors.
- template<class T >
Quaternion< T > [SPL::operator*](#) (const Quaternion< T > &q, const Quaternion< T > &r)
Compute the product of two quaternions.
- template<class T >
Quaternion< T > [SPL::operator/](#) (const Quaternion< T > &q, const Quaternion< T > &r)

Compute the quotient of two quaternions.

- `template<class T >`
`Quaternion< T > SPL::rotationToQuaternion (const Rotation_3< T > &rot)`

Convert a rotation into its corresponding quaternion.

- `template<class T >`
`Rotation_3< T > SPL::quaternionToRotation (const Quaternion< T > &q)`

Convert a unit-norm quaternion into its corresponding rotation.

11.6.1 Detailed Description

This file contains various CGAL utility code.

11.7 filterDesign.hpp File Reference

This file contains code for performing filter design.

```
#include <SPL/config.hpp>
#include <SPL/Sequence1.hpp>
```

Functions

- `RealSequence1 SPL::lowpassFilter` (double cutoffFreq, double transWidth, double maxPassbandRipple=0.1, double minStopbandAtten=20.0)
Design a zero-phase FIR lowpass filter.
- `RealSequence1 SPL::highpassFilter` (double cutoffFreq, double transWidth, double maxPassbandRipple=0.1, double minStopbandAtten=20.0)
Design a zero-phase FIR highpass filter.
- `RealSequence1 SPL::bandpassFilter` (double cutoffFreq0, double cutoffFreq1, double transWidth0, double transWidth1, double maxPassbandRipple=0.1, double minStopbandAtten=20.0)
Design a zero-phase FIR bandpass filter.

11.7.1 Detailed Description

This file contains code for performing filter design.

11.8 math.hpp File Reference

This file contains various mathematical functions/code.

```
#include <SPL/config.hpp>
#include <iostream>
#include <cmath>
#include <cassert>
#include <boost/tr1/cmath.hpp>
```

Functions

- `template<class T >`
`T SPL::absVal (T x)`
The absolute value function.
- `template<class T >`
`T SPL::signum (T x)`
The signum function.
- `template<class T >`
`T SPL::sqr (const T &x)`
The square function.
- `template<class T >`
`T SPL::clip (T x, T min, T max)`
The clip function.
- `double SPL::sinc (double x)`
The cardinal sine function.
- `long SPL::roundTowardZeroDiv (long x, long y)`
Compute a quotient with the result rounded towards zero.
- `long SPL::floorDiv (long x, long y)`
Compute the floor of a quotient.
- `template<class T >`
`T SPL::mod (T x, T y)`
Compute the remainder after division.
- `long SPL::ceilDiv (long x, long y)`
Compute the ceiling of a quotient.
- `double SPL::radToDeg (double x)`
Convert from radians to degrees.
- `double SPL::degToRad (double x)`
Convert from degrees to radians.

11.8.1 Detailed Description

This file contains various mathematical functions/code.

11.9 mCoder.hpp File Reference

This file contains interface information for an implementation of the M-Coder arithmetic coder from: ISO/IEC 14496-10:2008 (a.k.a. H.264)

```
#include <SPL/config.hpp>
#include <vector>
#include <iostream>
#include <SPL/bitStream.hpp>
```

Classes

- class [SPL::MEncoder](#)
The M-Coder (binary) arithmetic encoder class.
- class [SPL::MDecoder](#)
The M-Coder (binary) arithmetic decoder class.

11.9.1 Detailed Description

This file contains interface information for an implementation of the M-Coder arithmetic coder from: ISO/IEC 14496-10:2008 (a.k.a. H.264)

11.10 misc.hpp File Reference

This file contains miscellaneous code.

```
#include <SPL/config.hpp>
```

Functions

- template<class InputIterator, class Size, class OutputIterator >
OutputIterator **SPL::copy_n** (InputIterator first, Size count, OutputIterator result)
This template function is equivalent to std::copy_n in the new C++ 0x standard.

11.10.1 Detailed Description

This file contains miscellaneous code.

11.11 pnmCodec.cpp File Reference

This file contains a PNM codec.

```
#include <SPL/config.hpp>
#include <iostream>
#include <sstream>
#include <cassert>
#include <cstdlib>
#include <SPL/pnmCodec.hpp>
```

Functions

- int **SPL::pnmPutHeader** (std::ostream &out, PnmHeader &header)
Write a PNM header to the specified stream.
- int **SPL::pnmPutBinInt** (std::ostream &out, int wordSize, bool sgnd, long val)
Write an integer from the specified stream.

- int **SPL::pnmGetHeader** (std::istream &in, PnmHeader &header)
Read a PNM header from the specified stream.
- int **SPL::pnmGetTxtBit** (std::istream &in)
Read a bit from the specified stream.
- long **SPL::pnmGetTxtInt** (std::istream &in, bool sgnd, int &status)
Read an integer from the specified stream.
- int **SPL::pnmGetChar** (std::istream &in)
Read a character from the specified stream.
- long **SPL::pnmGetBinInt** (std::istream &in, int wordSize, bool sgnd, int &status)
Read an integer from the specified stream.
- PnmType **SPL::pnmGetType** (PnmMagic magic)
Determine the type (i.e., PGM or PPM) from the magic number.
- PnmFmt **SPL::pnmGetFmt** (PnmMagic magic)
Determine the format (i.e., text or binary) from magic number.
- int **SPL::pnmGetNumComps** (PnmType type)
Get the number of components from the PNM type.
- int **SPL::pnmMaxValToPrec** (int maxVal)
Determine the precision from the maximum value.

11.11.1 Detailed Description

This file contains a PNM codec.

11.12 pnmCodec.hpp File Reference

This file contains a PNM codec.

```
#include <SPL/config.hpp>
#include <iostream>
#include <sstream>
#include <cassert>
#include <cstdlib>
```

Classes

- struct [SPL::PnmHeader](#)
The header information for PNM data.

Enumerations

- enum **PnmMagic**
The signature values that can appear at the start of the header.
- enum **PnmType**
The type of the PNM data.
- enum **PnmFmt**
The format of the PNM data (i.e., binary or text).

Functions

- PnmType **SPL::pnmGetType** (PnmMagic magic)
Determine the type (i.e., PGM or PPM) from the magic number.
- PnmFmt **SPL::pnmGetFmt** (PnmMagic magic)
Determine the format (i.e., text or binary) from magic number.
- int **SPL::pnmMaxValToPrec** (int maxVal)
Determine the precision from the maximum value.
- int **SPL::pnmGetNumComps** (PnmType type)
Get the number of components from the PNM type.
- long **SPL::pnmOnes** (int n)
Get an integer whose representation in binary consists of the specified number of ones.
- int **SPL::pnmGetHeader** (std::istream &in, PnmHeader &header)
Read a PNM header from the specified stream.
- int **SPL::pnmPutHeader** (std::ostream &out, PnmHeader &header)
Write a PNM header to the specified stream.
- int **SPL::pnmGetChar** (std::istream &in)
Read a character from the specified stream.
- int **SPL::pnmGetTxtBit** (std::istream &in)
Read a bit from the specified stream.
- long **SPL::pnmGetTxtInt** (std::istream &in, bool sgnd, int &status)
Read an integer from the specified stream.
- long **SPL::pnmGetBinInt** (std::istream &in, int wordSize, bool sgnd, int &status)
Read an integer from the specified stream.
- int **SPL::pnmPutBinInt** (std::ostream &out, int wordSize, bool sgnd, long val)
Write an integer from the specified stream.
- template<class GetData >
int **SPL::pnmEncode** (std::ostream &outStream, int width, int height, int numComps, int maxVal, bool sgnd, GetData &getData, bool binaryFormat)
Write data encoded in the PNM format to the specified stream.
- template<class GetData >
int **SPL::putData** (std::ostream &out, PnmHeader &header, GetData &getData)
Write the actual image data to a stream.
- template<class Initialize >
int **SPL::pnmDecode** (std::istream &inStream, Initialize &initialize)
Read data encoded in the PNM format from the specified stream.
- template<class PutData >
int **SPL::getData** (std::istream &in, PnmHeader &header, PutData &putData)
Read the actual image data from the specified stream.

Variables

- const int **SPL::pnmMaxLineLen** = 80
The maximum line length to be produced when encoding in text format.

11.12.1 Detailed Description

This file contains a PNM codec.

11.13 Sequence.hpp File Reference

Common header for sequence classes.

```
#include <SPL/config.hpp>
```

Classes

- struct [SPL::ConvolveMode](#)
Constants identifying various convolution modes.

11.13.1 Detailed Description

Common header for sequence classes.

11.14 Sequence1.hpp File Reference

This file contains code for the Sequence1 template class.

```
#include <SPL/config.hpp>
#include <iostream>
#include <vector>
#include <SPL/Array1.hpp>
#include <SPL/math.hpp>
#include <SPL/Sequence.hpp>
```

Classes

- class [SPL::Sequence1< T >](#)
A one-dimensional sequence class with lazy copying and reference counting.

Macros

- `#define` [SPL_SEQUENCE1_DEBUG](#)
Defining this symbol will enable extra code for debugging.
- `#define` [SPL_SEQUENCE1_USE_NEW_CONV](#)
Defining this symbol will enable the use of new convolution code.
- `#define` [SPL_SEQUENCE1_INLINE](#)
Prevent the inlining of functions.

Typedefs

- `typedef` `Sequence1< double >` [SPL::RealSequence1](#)
Real sequence.
- `typedef` `Sequence1< int >` [SPL::IntSequence1](#)
Integer sequence.

Functions

- `template<class T >`
`std::ostream & SPL::operator<< (std::ostream &out, const Sequence1< T > &f)`
Output a sequence to a stream.
- `template<class T >`
`std::istream & SPL::operator>> (std::istream &in, Sequence1< T > &f)`
Input a sequence from a stream.
- `template<class T >`
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator+ (const Sequence1< T > &f, const Sequence1< T > &g)`
Compute the sum of two sequences.
- `template<class T >`
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator- (const Sequence1< T > &f, const Sequence1< T > &g)`
Compute the difference of two sequences.
- `template<class T >`
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator* (const Sequence1< T > &f, const Sequence1< T > &g)`
Compute the (element-wise) product of two sequences.
- `template<class T >`
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator/ (const Sequence1< T > &f, const Sequence1< T > &g)`
Compute the (element-wise) quotient of two sequences.
- `template<class T >`
`Sequence1< T > SPL::add (const Sequence1< T > &f, const Sequence1< T > &g)`
Compute the sum of two sequences with potentially differing domains.
- `template<class T >`
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator+ (const T &a, const Sequence1< T > &f)`
Add a value to a sequence.
- `template<class T >`
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator+ (const Sequence1< T > &f, const T &a)`
Add a value to a sequence.
- `template<class T >`
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator- (const Sequence1< T > &f, const T &a)`
Subtract a value from a sequence.
- `template<class T >`
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator* (const T &a, const Sequence1< T > &f)`
Compute a scalar multiple of a sequence.
- `template<class T >`
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator* (const Sequence1< T > &f, const T &a)`
Compute a scalar multiple of a sequence.
- `template<class T >`
`SPL_SEQUENCE1_INLINE Sequence1< T > SPL::operator/ (const Sequence1< T > &f, const T &a)`
Divide a sequence by a scalar.
- `template<class T >`
`SPL_SEQUENCE1_INLINE bool SPL::operator== (const Sequence1< T > &f, const Sequence1< T > &g)`
Test two sequences for equality.
- `template<class T >`
`SPL_SEQUENCE1_INLINE bool SPL::operator!= (const Sequence1< T > &f, const Sequence1< T > &g)`

Test two sequences for inequality.

- `template<class T >`
`SPL_SEQUENCE1_INLINE` `bool SPL::approxEqual` (const Sequence1< T > &f, const Sequence1< T > &g, T
threshold=1e-9)

Test two sequences for approximate equality.

- `template<class T >`
Sequence1< T > `SPL::subsequence` (const Sequence1< T > &f, int startIdx, int size)

Extract a subsequence from a sequence.

- `template<class T >`
`SPL_SEQUENCE1_INLINE` Sequence1< T > `SPL::translate` (const Sequence1< T > &f, int delta)

Translate a sequence by the specified amount.

- `template<class T >`
Sequence1< T > `SPL::convolve` (const Sequence1< T > &f, const Sequence1< T > &g, int mode=Convolve↵
Mode::full)

Compute the convolution of two sequences.

- `template<class T >`
Sequence1< T > `SPL::downsample` (const Sequence1< T > &f, int factor)

Downsample a sequence by the specified factor.

- `template<class T >`
Sequence1< T > `SPL::upsample` (const Sequence1< T > &f, int factor, int pad=0)

Upsample a sequence by the specified factor.

- `template<class T >`
Array1< Sequence1< T > > `SPL::polyphaseSplit` (const Sequence1< T > &seq, int type, int numPhases)

Split a sequence into its polyphase components.

- `template<class T >`
Sequence1< T > `SPL::polyphaseJoin` (const Array1< Sequence1< T > > &comps, int type)

Reassemble a sequence from its polyphase components.

11.14.1 Detailed Description

This file contains code for the Sequence1 template class.

11.14.2 Macro Definition Documentation

11.14.2.1 `#define SPL_SEQUENCE1_DEBUG`

Defining this symbol will enable extra code for debugging.

11.14.2.2 `#define SPL_SEQUENCE1_INLINE`

Prevent the inlining of functions.

11.14.2.3 `#define SPL_SEQUENCE1_USE_NEW_CONV`

Defining this symbol will enable the use of new convolution code.

11.15 Sequence2.cpp File Reference

This file contains code for the Sequence2 template class.

```
#include <SPL/config.hpp>
#include <iostream>
#include <cassert>
#include <cstdlib>
#include <SPL/Sequence2.hpp>
```

Functions

- void **SPL::combineDomains** (int firstStartX, int firstStartY, int firstEndX, int firstEndY, int secondStartX, int secondStartY, int secondEndX, int secondEndY, int &startX, int &startY, int &endX, int &endY)

Find the bounding box of the union of the domains of two sequences.

11.15.1 Detailed Description

This file contains code for the Sequence2 template class.

11.16 Sequence2.hpp File Reference

This file contains code for the Sequence2 template class.

```
#include <SPL/config.hpp>
#include <iostream>
#include <vector>
#include <SPL/Array2.hpp>
#include <SPL/Sequence.hpp>
#include <SPL/Sequence1.hpp>
#include <SPL/math.hpp>
```

Classes

- class [SPL::Sequence2< T >](#)

A two-dimensional sequence class with lazy copying and reference counting.

Macros

- [#define SPL_SEQUENCE2_USE_NEW_CONV](#)

Defining this symbol will enable extra code for debugging.

- [#define SPL_SEQUENCE2_INLINE](#) inline

Allow the inlining of functions.

Typedefs

- typedef Sequence2< double > [SPL::RealSequence2](#)
Real sequence.
- typedef Sequence2< int > [SPL::IntSequence2](#)
Integer sequence.

Functions

- template<class T >
std::ostream & [SPL::operator<<](#) (std::ostream &out, const Sequence2< T > &f)
Output a sequence to a stream.
- template<class T >
std::istream & [SPL::operator>>](#) (std::istream &in, Sequence2< T > &f)
Input a sequence from a stream.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator+](#) (const Sequence2< T > &f, const Sequence2< T > &g)
Compute the sum of two sequences.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator-](#) (const Sequence2< T > &f, const Sequence2< T > &g)
Compute the difference of two sequences.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator*](#) (const Sequence2< T > &f, const Sequence2< T > &g)
Compute the (element-wise) product of two sequences.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator/](#) (const Sequence2< T > &f, const Sequence2< T > &g)
Compute the (element-wise) quotient of two sequences.
- template<class T >
Sequence2< T > [SPL::add](#) (const Sequence2< T > &f, const Sequence2< T > &g)
Compute the sum of two sequences with potentially differing domains.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator+](#) (const T &value, const Sequence2< T > &f)
Add a value to a sequence.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator+](#) (const Sequence2< T > &f, const T &value)
Add a value to a sequence.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator-](#) (const Sequence2< T > &f, const T &value)
Subtract a value from a sequence.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator*](#) (const T &value, const Sequence2< T > &f)
Compute a scalar multiple of a sequence.
- template<class T >
[SPL_SEQUENCE2_INLINE](#) Sequence2< T > [SPL::operator*](#) (const Sequence2< T > &f, const T &value)
Compute a scalar multiple of a sequence.

- `template<class T >`
`SPL_SEQUENCE2_INLINE` `Sequence2< T > SPL::operator/` (const `Sequence2< T >` &f, const `T` &value)
Divide a sequence by a scalar.
- `template<class T >`
`bool SPL::operator==` (const `Sequence2< T >` &f, const `Sequence2< T >` &g)
Test two sequences for equality.
- `template<class T >`
`SPL_SEQUENCE2_INLINE` `bool SPL::operator!=` (const `Sequence2< T >` &f, const `Sequence2< T >` &g)
Test two sequences for inequality.
- `template<class T >`
`SPL_SEQUENCE2_INLINE` `bool SPL::approxEqual` (const `Sequence2< T >` &f, const `Sequence2< T >` &g, `T` threshold=1e-9)
Test two sequences for approximate equality.
- `template<class T >`
`Sequence2< T > SPL::subsequence` (const `Sequence2< T >` &f, int startX, int startY, int width, int height)
Extract a subsequence from a sequence.
- `template<class T >`
`SPL_SEQUENCE2_INLINE` `Sequence2< T > SPL::translate` (const `Sequence2< T >` &f, int deltaX, int deltaY)
Translate a sequence by the specified amount.
- `template<class T >`
`Sequence2< T > SPL::convolve` (const `Sequence2< T >` &f, const `Sequence2< T >` &g, int mode)
Compute the convolution of two sequences.
- `template<class T >`
`Sequence2< T > SPL::convolveSeparable` (const `Sequence2< T >` &f, const `Sequence1< T >` &horzFilt, const `Sequence1< T >` &vertFilt, int mode=`ConvolveMode::full`)
Compute the convolution of a sequence with two 1-D filters (i.e., convolution with a separable filter).
- `template<class T >`
`Sequence2< T > SPL::downsample` (const `Sequence2< T >` &f, int factorX, int factorY)
Downsample a sequence in each of the horizontal and vertical directions by the specified factors.
- `template<class T >`
`Sequence2< T > SPL::upsample` (const `Sequence2< T >` &f, int factorX, int factorY)
Upsample a sequence in each of the horizontal and vertical directions by the specified factors.
- `template<class T >`
`Sequence2< T > SPL::upsample` (const `Sequence2< T >` &f, int factorX, int factorY, int padX, int padY)
Upsample a sequence in each of the horizontal and vertical directions by the specified factors.
- `template<class T >`
`Array2< Sequence2< T > > SPL::polyphaseSplit` (const `Sequence2< T >` &seq, int typeX, int numPhasesX, int typeY, int numPhasesY)
Split a sequence into its polyphase components.
- `template<class T >`
`Sequence2< T > SPL::polyphaseJoin` (const `Array2< Sequence2< T > >` &comps, int typeX, int typeY)
Reassemble a sequence from its polyphase components.

11.16.1 Detailed Description

This file contains code for the `Sequence2` template class.

11.16.2 Macro Definition Documentation

11.16.2.1 `#define SPL_SEQUENCE2_INLINE` inline

Allow the inlining of functions.

11.16.2.2 `#define SPL_SEQUENCE2_USE_NEW_CONV`

Defining this symbol will enable extra code for debugging.

Defining this symbol will enable some new code for convolution.

11.17 Timer.cpp File Reference

The file contains code for obtaining timing/memory usage information.

```
#include <SPL/config.hpp>
#include <SPL/Timer.hpp>
#include <iostream>
#include <fstream>
#include <cassert>
#include <iterator>
#include <vector>
#include <string>
#include <unistd.h>
#include <boost/lexical_cast.hpp>
#include <boost/tokenizer.hpp>
```

Functions

- double [SPL::getPeakMemUsage](#) ()
Get the peak memory usage for the process.
- double [SPL::getCurrentMemUsage](#) ()
Get the amount of memory currently being used by the process.

11.17.1 Detailed Description

The file contains code for obtaining timing/memory usage information.

11.18 Timer.hpp File Reference

This file contains code for the Timer class.

```
#include <SPL/config.hpp>
#include <iostream>
#include <cstdlib>
```

Classes

- class [SPL::Timer](#)
A class for making timing measurements.

Functions

- double [SPL::getCurrentMemUsage](#) ()
Get the amount of memory currently being used by the process.
- double [SPL::getPeakMemUsage](#) ()
Get the peak memory usage for the process.

11.18.1 Detailed Description

This file contains code for the Timer class.

