

## Hosting Environment (Daemon)

Generated by Doxygen 1.7.1

Tue Dec 21 2010 22:34:56



# Contents

<b>1</b>	<b>Namespace Index</b>	<b>1</b>
1.1	Namespace List . . . . .	1
<b>2</b>	<b>Data Structure Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Data Structure Index</b>	<b>11</b>
3.1	Data Structures . . . . .	11
<b>4</b>	<b>File Index</b>	<b>19</b>
4.1	File List . . . . .	19
<b>5</b>	<b>Namespace Documentation</b>	<b>23</b>
5.1	Arc Namespace Reference . . . . .	23
5.1.1	Detailed Description . . . . .	35
5.1.2	Typedef Documentation . . . . .	36
5.1.2.1	AttrConstIter . . . . .	36
5.1.2.2	AttrIter . . . . .	36
5.1.2.3	AttrMap . . . . .	36
5.1.2.4	get_plugin_instance . . . . .	36
5.1.3	Enumeration Type Documentation . . . . .	36
5.1.3.1	LogLevel . . . . .	36
5.1.3.2	StatusKind . . . . .	36
5.1.3.3	WSAFault . . . . .	37
5.1.4	Function Documentation . . . . .	37
5.1.4.1	addVOMSAC . . . . .	37
5.1.4.2	ContentFromPayload . . . . .	37
5.1.4.3	CreateThreadFunction . . . . .	37
5.1.4.4	createVOMSAC . . . . .	38
5.1.4.5	FileOpen . . . . .	38

---

5.1.4.6	final_xmlsec . . . . .	38
5.1.4.7	get_cert_str . . . . .	38
5.1.4.8	get_key_from_certfile . . . . .	38
5.1.4.9	get_key_from_certstr . . . . .	38
5.1.4.10	get_key_from_keyfile . . . . .	38
5.1.4.11	get_key_from_keystr . . . . .	38
5.1.4.12	get_node . . . . .	38
5.1.4.13	get_property . . . . .	39
5.1.4.14	GUID . . . . .	39
5.1.4.15	init_xmlsec . . . . .	39
5.1.4.16	istring_to_level . . . . .	39
5.1.4.17	load_key_from_certfile . . . . .	39
5.1.4.18	load_key_from_certstr . . . . .	39
5.1.4.19	load_key_from_keyfile . . . . .	39
5.1.4.20	load_trusted_cert_file . . . . .	40
5.1.4.21	load_trusted_cert_str . . . . .	40
5.1.4.22	load_trusted_certs . . . . .	40
5.1.4.23	MatchXMLName . . . . .	40
5.1.4.24	MatchXMLName . . . . .	40
5.1.4.25	MatchXMLName . . . . .	40
5.1.4.26	MatchXMLNamespace . . . . .	40
5.1.4.27	MatchXMLNamespace . . . . .	40
5.1.4.28	MatchXMLNamespace . . . . .	40
5.1.4.29	OpenSSLInit . . . . .	40
5.1.4.30	operator<< . . . . .	41
5.1.4.31	operator<< . . . . .	41
5.1.4.32	operator<< . . . . .	41
5.1.4.33	parseVOMSAC . . . . .	41
5.1.4.34	parseVOMSAC . . . . .	42
5.1.4.35	passphrase_callback . . . . .	42
5.1.4.36	string . . . . .	42
5.1.4.37	TimeStamp . . . . .	42
5.1.4.38	TimeStamp . . . . .	42
5.1.4.39	VOMSDecode . . . . .	42
5.1.4.40	WSAFaultAssign . . . . .	42
5.1.4.41	WSAFaultExtract . . . . .	42

5.1.5	Variable Documentation . . . . .	43
5.1.5.1	CredentialLogger . . . . .	43
5.1.5.2	plugins_table_name . . . . .	43
5.1.5.3	thread_stacksize . . . . .	43
5.2	ArcCredential Namespace Reference . . . . .	43
5.2.1	Detailed Description . . . . .	44
5.2.2	Enumeration Type Documentation . . . . .	44
5.2.2.1	certType . . . . .	44
<b>6</b>	<b>Data Structure Documentation</b>	<b>45</b>
6.1	ArcCredential::ACACI Struct Reference . . . . .	45
6.2	ArcCredential::ACATTHOLDER Struct Reference . . . . .	45
6.3	ArcCredential::ACATTR Struct Reference . . . . .	45
6.4	ArcCredential::ACATTRIBUTE Struct Reference . . . . .	45
6.5	ArcCredential::ACC Struct Reference . . . . .	45
6.6	ArcCredential::ACCERTS Struct Reference . . . . .	46
6.7	ArcCredential::ACDIGEST Struct Reference . . . . .	46
6.8	ArcCredential::ACFORM Struct Reference . . . . .	46
6.9	ArcCredential::ACFULLATTRIBUTES Struct Reference . . . . .	46
6.10	ArcCredential::ACHOLDER Struct Reference . . . . .	46
6.11	ArcCredential::ACIETFATTR Struct Reference . . . . .	46
6.12	ArcCredential::ACINFO Struct Reference . . . . .	46
6.13	ArcCredential::ACIS Struct Reference . . . . .	47
6.14	ArcCredential::ACSEQ Struct Reference . . . . .	47
6.15	ArcCredential::ACTARGET Struct Reference . . . . .	47
6.16	ArcCredential::ACTARGETS Struct Reference . . . . .	47
6.17	ArcCredential::ACVAL Struct Reference . . . . .	47
6.18	Arc::Adler32Sum Class Reference . . . . .	47
6.18.1	Detailed Description . . . . .	48
6.19	ArcSec::AlgFactory Class Reference . . . . .	48
6.19.1	Detailed Description . . . . .	48
6.19.2	Member Function Documentation . . . . .	48
6.19.2.1	createAlg . . . . .	48
6.20	ArcSec::AnyURIAttribute Class Reference . . . . .	49
6.20.1	Member Function Documentation . . . . .	49
6.20.1.1	encode . . . . .	49
6.20.1.2	equal . . . . .	49

6.20.1.3	getId	49
6.20.1.4	getType	49
6.21	Arc::ApplicationEnvironment Class Reference	50
6.21.1	Detailed Description	50
6.22	Arc::ApplicationType Class Reference	50
6.23	Arc::ARCJSDDLParser Class Reference	50
6.24	Arc::ArcLocation Class Reference	51
6.24.1	Detailed Description	51
6.24.2	Member Function Documentation	51
6.24.2.1	GetPlugins	51
6.24.2.2	Init	51
6.25	ArcSec::ArcPeriod Struct Reference	51
6.26	Arc::ARCPolicyHandlerConfig Class Reference	51
6.27	ArcSec::Attr Struct Reference	52
6.27.1	Detailed Description	52
6.28	ArcSec::AttributeFactory Class Reference	52
6.28.1	Detailed Description	52
6.29	Arc::AttributeIterator Class Reference	53
6.29.1	Detailed Description	53
6.29.2	Constructor & Destructor Documentation	53
6.29.2.1	AttributeIterator	53
6.29.2.2	AttributeIterator	54
6.29.3	Member Function Documentation	54
6.29.3.1	hasMore	54
6.29.3.2	key	54
6.29.3.3	operator*	54
6.29.3.4	operator++	54
6.29.3.5	operator++	55
6.29.3.6	operator->	55
6.29.4	Friends And Related Function Documentation	55
6.29.4.1	MessageAttributes	55
6.29.5	Field Documentation	55
6.29.5.1	current_	55
6.29.5.2	end_	55
6.30	ArcSec::AttributeProxy Class Reference	55
6.30.1	Detailed Description	56

---

6.30.2	Member Function Documentation	56
6.30.2.1	getAttribute	56
6.31	ArcSec::AttributeValue Class Reference	56
6.31.1	Detailed Description	57
6.31.2	Member Function Documentation	57
6.31.2.1	encode	57
6.31.2.2	equal	58
6.31.2.3	getId	58
6.31.2.4	getType	58
6.32	ArcSec::Attrs Class Reference	58
6.32.1	Detailed Description	58
6.33	ArcSec::AuthzRequest Struct Reference	59
6.34	ArcSec::AuthzRequestSection Struct Reference	59
6.34.1	Detailed Description	59
6.35	Arc::AutoPointer< T > Class Template Reference	59
6.35.1	Detailed Description	59
6.36	Arc::Base64 Class Reference	60
6.37	Arc::BaseConfig Class Reference	60
6.37.1	Detailed Description	60
6.37.2	Member Function Documentation	60
6.37.2.1	AddCADir	60
6.37.2.2	AddCAFile	60
6.37.2.3	AddCertificate	61
6.37.2.4	AddOverlay	61
6.37.2.5	AddPluginsPath	61
6.37.2.6	AddPrivateKey	61
6.37.2.7	AddProxy	61
6.37.2.8	GetOverlay	61
6.37.2.9	MakeConfig	61
6.38	ArcSec::BooleanAttribute Class Reference	61
6.38.1	Member Function Documentation	62
6.38.1.1	encode	62
6.38.1.2	equal	62
6.38.1.3	getId	62
6.38.1.4	getType	62
6.39	Arc::Broker Class Reference	62

6.39.1	Member Function Documentation . . . . .	63
6.39.1.1	GetBestTarget . . . . .	63
6.39.1.2	PreFilterTargets . . . . .	63
6.39.1.3	SortTargets . . . . .	63
6.39.2	Field Documentation . . . . .	64
6.39.2.1	PossibleTargets . . . . .	64
6.40	Arc::BrokerLoader Class Reference . . . . .	64
6.40.1	Detailed Description . . . . .	64
6.40.2	Constructor & Destructor Documentation . . . . .	64
6.40.2.1	BrokerLoader . . . . .	64
6.40.2.2	~BrokerLoader . . . . .	64
6.40.3	Member Function Documentation . . . . .	65
6.40.3.1	GetBrokers . . . . .	65
6.40.3.2	load . . . . .	65
6.41	Arc::BrokerPluginArgument Class Reference . . . . .	65
6.42	Arc::ByteArray Class Reference . . . . .	65
6.43	Arc::CacheParameters Struct Reference . . . . .	66
6.43.1	Detailed Description . . . . .	66
6.44	ArcCredential::cert_verify_context Struct Reference . . . . .	66
6.45	Arc::ChainContext Class Reference . . . . .	66
6.45.1	Detailed Description . . . . .	66
6.45.2	Member Function Documentation . . . . .	66
6.45.2.1	operator PluginsFactory * . . . . .	66
6.46	Arc::Checksum Class Reference . . . . .	67
6.46.1	Detailed Description . . . . .	67
6.47	Arc::ChecksumAny Class Reference . . . . .	67
6.47.1	Detailed Description . . . . .	67
6.48	Arc::CStringValue Class Reference . . . . .	67
6.48.1	Detailed Description . . . . .	68
6.48.2	Constructor & Destructor Documentation . . . . .	68
6.48.2.1	CStringValue . . . . .	68
6.48.2.2	CStringValue . . . . .	68
6.48.2.3	CStringValue . . . . .	68
6.48.3	Member Function Documentation . . . . .	68
6.48.3.1	equal . . . . .	68
6.48.3.2	operator bool . . . . .	69

6.49	Arc::ClassLoader Class Reference . . . . .	69
6.50	Arc::ClassLoaderPluginArgument Class Reference . . . . .	69
6.51	Arc::ClientHTTP Class Reference . . . . .	69
6.51.1	Detailed Description . . . . .	70
6.52	Arc::ClientHTTPwithSAML2SSO Class Reference . . . . .	70
6.52.1	Constructor & Destructor Documentation . . . . .	70
6.52.1.1	ClientHTTPwithSAML2SSO . . . . .	70
6.52.2	Member Function Documentation . . . . .	70
6.52.2.1	process . . . . .	70
6.53	Arc::ClientInterface Class Reference . . . . .	71
6.53.1	Detailed Description . . . . .	71
6.54	Arc::ClientSOAP Class Reference . . . . .	71
6.54.1	Detailed Description . . . . .	72
6.54.2	Constructor & Destructor Documentation . . . . .	72
6.54.2.1	ClientSOAP . . . . .	72
6.54.3	Member Function Documentation . . . . .	72
6.54.3.1	AddSecHandler . . . . .	72
6.54.3.2	GetEntry . . . . .	72
6.54.3.3	Load . . . . .	72
6.54.3.4	process . . . . .	72
6.54.3.5	process . . . . .	72
6.55	Arc::ClientSOAPwithSAML2SSO Class Reference . . . . .	73
6.55.1	Constructor & Destructor Documentation . . . . .	73
6.55.1.1	ClientSOAPwithSAML2SSO . . . . .	73
6.55.2	Member Function Documentation . . . . .	73
6.55.2.1	process . . . . .	73
6.55.2.2	process . . . . .	73
6.56	Arc::ClientTCP Class Reference . . . . .	73
6.56.1	Detailed Description . . . . .	74
6.57	Arc::ClientX509Delegation Class Reference . . . . .	74
6.57.1	Constructor & Destructor Documentation . . . . .	74
6.57.1.1	ClientX509Delegation . . . . .	74
6.57.2	Member Function Documentation . . . . .	74
6.57.2.1	acquireDelegation . . . . .	74
6.57.2.2	createDelegation . . . . .	75
6.58	ArcSec::CombiningAlg Class Reference . . . . .	75

---

6.58.1	Detailed Description	76
6.58.2	Member Function Documentation	76
6.58.2.1	combine	76
6.58.2.2	getalgId	76
6.59	Arc::Config Class Reference	76
6.59.1	Detailed Description	77
6.59.2	Constructor & Destructor Documentation	77
6.59.2.1	Config	77
6.59.2.2	Config	77
6.59.2.3	Config	77
6.59.2.4	Config	77
6.59.2.5	Config	77
6.59.2.6	Config	78
6.59.3	Member Function Documentation	78
6.59.3.1	getFileName	78
6.59.3.2	parse	78
6.59.3.3	print	78
6.59.3.4	save	78
6.59.3.5	setFileName	78
6.60	Arc::ConfusaCertHandler Class Reference	78
6.60.1	Detailed Description	78
6.60.2	Constructor & Destructor Documentation	79
6.60.2.1	ConfusaCertHandler	79
6.60.3	Member Function Documentation	79
6.60.3.1	createCertRequest	79
6.60.3.2	getCertRequestB64	79
6.61	Arc::ConfusaParserUtils Class Reference	79
6.61.1	Detailed Description	79
6.61.2	Member Function Documentation	79
6.61.2.1	destroy_doc	79
6.61.2.2	evaluate_path	80
6.61.2.3	extract_body_information	80
6.61.2.4	get_doc	80
6.61.2.5	handle_redirect_step	80
6.61.2.6	urlencode	80
6.61.2.7	urlencode_params	80

---

6.62	Arc::CountedPointer< T > Class Template Reference . . . . .	80
6.62.1	Detailed Description . . . . .	81
6.63	Arc::Counter Class Reference . . . . .	81
6.63.1	Detailed Description . . . . .	82
6.63.2	Member Typedef Documentation . . . . .	83
6.63.2.1	IDType . . . . .	83
6.63.3	Constructor & Destructor Documentation . . . . .	83
6.63.3.1	Counter . . . . .	83
6.63.3.2	~Counter . . . . .	83
6.63.4	Member Function Documentation . . . . .	84
6.63.4.1	cancel . . . . .	84
6.63.4.2	changeExcess . . . . .	84
6.63.4.3	changeLimit . . . . .	84
6.63.4.4	extend . . . . .	84
6.63.4.5	getCounterTicket . . . . .	85
6.63.4.6	getCurrentTime . . . . .	85
6.63.4.7	getExcess . . . . .	85
6.63.4.8	getExpirationReminder . . . . .	86
6.63.4.9	getExpiryTime . . . . .	86
6.63.4.10	getLimit . . . . .	86
6.63.4.11	getValue . . . . .	86
6.63.4.12	reserve . . . . .	87
6.63.4.13	setExcess . . . . .	87
6.63.4.14	setLimit . . . . .	87
6.64	Arc::CounterTicket Class Reference . . . . .	88
6.64.1	Detailed Description . . . . .	88
6.64.2	Constructor & Destructor Documentation . . . . .	89
6.64.2.1	CounterTicket . . . . .	89
6.64.3	Member Function Documentation . . . . .	89
6.64.3.1	cancel . . . . .	89
6.64.3.2	extend . . . . .	89
6.64.3.3	isValid . . . . .	89
6.65	Arc::CRC32Sum Class Reference . . . . .	89
6.65.1	Detailed Description . . . . .	90
6.66	Arc::Credential Class Reference . . . . .	90
6.66.1	Constructor & Destructor Documentation . . . . .	91

---

6.66.1.1	Credential	91
6.66.1.2	Credential	91
6.66.1.3	Credential	91
6.66.1.4	Credential	92
6.66.1.5	Credential	92
6.66.2	Member Function Documentation	92
6.66.2.1	AddCertExtObj	92
6.66.2.2	AddExtension	92
6.66.2.3	AddExtension	93
6.66.2.4	GenerateEECRequest	93
6.66.2.5	GenerateEECRequest	93
6.66.2.6	GenerateEECRequest	93
6.66.2.7	GenerateRequest	93
6.66.2.8	GenerateRequest	93
6.66.2.9	GenerateRequest	93
6.66.2.10	GetCert	93
6.66.2.11	GetCertNumofChain	94
6.66.2.12	GetCertReq	94
6.66.2.13	GetDN	94
6.66.2.14	GetEndTime	94
6.66.2.15	getFormat	94
6.66.2.16	GetIdentityName	94
6.66.2.17	GetLifeTime	94
6.66.2.18	GetPrivKey	94
6.66.2.19	GetProxyPolicy	94
6.66.2.20	GetPubKey	94
6.66.2.21	GetStartTime	94
6.66.2.22	GetType	95
6.66.2.23	GetVerification	95
6.66.2.24	InitProxyCertInfo	95
6.66.2.25	InquireRequest	95
6.66.2.26	InquireRequest	95
6.66.2.27	InquireRequest	95
6.66.2.28	LogError	95
6.66.2.29	OutputCertificate	95
6.66.2.30	OutputCertificateChain	95

---

6.66.2.31	OutputPrivatekey	96
6.66.2.32	OutputPublickey	96
6.66.2.33	SetLifeTime	96
6.66.2.34	SetProxyPolicy	96
6.66.2.35	SetStartTime	96
6.66.2.36	SignEECRequest	96
6.66.2.37	SignEECRequest	96
6.66.2.38	SignEECRequest	96
6.66.2.39	SignRequest	97
6.66.2.40	SignRequest	97
6.66.2.41	SignRequest	97
6.66.2.42	STACK_OF	97
6.67	Arc::CredentialError Class Reference	97
6.67.1	Detailed Description	97
6.67.2	Constructor & Destructor Documentation	98
6.67.2.1	CredentialError	98
6.68	Arc::CredentialStore Class Reference	98
6.68.1	Detailed Description	98
6.69	Arc::Database Class Reference	98
6.69.1	Detailed Description	99
6.69.2	Constructor & Destructor Documentation	99
6.69.2.1	Database	99
6.69.2.2	Database	99
6.69.2.3	Database	99
6.69.2.4	~Database	99
6.69.3	Member Function Documentation	99
6.69.3.1	close	99
6.69.3.2	connect	99
6.69.3.3	enable_ssl	100
6.69.3.4	isconnected	100
6.69.3.5	shutdown	100
6.70	Arc::DataBuffer Class Reference	100
6.70.1	Detailed Description	101
6.70.2	Constructor & Destructor Documentation	101
6.70.2.1	DataBuffer	101
6.70.2.2	DataBuffer	102

6.70.3	Member Function Documentation . . . . .	102
6.70.3.1	add . . . . .	102
6.70.3.2	buffer_size . . . . .	102
6.70.3.3	checksum_object . . . . .	102
6.70.3.4	checksum_valid . . . . .	102
6.70.3.5	eof_read . . . . .	102
6.70.3.6	eof_read . . . . .	103
6.70.3.7	eof_write . . . . .	103
6.70.3.8	eof_write . . . . .	103
6.70.3.9	error . . . . .	103
6.70.3.10	error_read . . . . .	103
6.70.3.11	error_write . . . . .	103
6.70.3.12	for_read . . . . .	103
6.70.3.13	for_read . . . . .	103
6.70.3.14	for_write . . . . .	104
6.70.3.15	for_write . . . . .	104
6.70.3.16	is_notwritten . . . . .	104
6.70.3.17	is_notwritten . . . . .	104
6.70.3.18	is_read . . . . .	104
6.70.3.19	is_read . . . . .	104
6.70.3.20	is_written . . . . .	105
6.70.3.21	is_written . . . . .	105
6.70.3.22	set . . . . .	105
6.70.3.23	wait_any . . . . .	105
6.71	Arc::DataCallback Class Reference . . . . .	105
6.71.1	Detailed Description . . . . .	106
6.72	Arc::DataHandle Class Reference . . . . .	106
6.72.1	Detailed Description . . . . .	106
6.73	Arc::DataMover Class Reference . . . . .	106
6.73.1	Detailed Description . . . . .	107
6.73.2	Member Function Documentation . . . . .	107
6.73.2.1	checks . . . . .	107
6.73.2.2	checks . . . . .	107
6.73.2.3	force_to_meta . . . . .	107
6.73.2.4	secure . . . . .	107
6.73.2.5	set_default_max_inactivity_time . . . . .	107

---

6.73.2.6	set_default_min_average_speed . . . . .	107
6.73.2.7	set_default_min_speed . . . . .	107
6.73.2.8	Transfer . . . . .	108
6.73.2.9	Transfer . . . . .	108
6.73.2.10	verbose . . . . .	108
6.74	Arc::DataPoint Class Reference . . . . .	108
6.74.1	Detailed Description . . . . .	111
6.74.2	Member Enumeration Documentation . . . . .	111
6.74.2.1	DataPointAccessLatency . . . . .	111
6.74.2.2	DataPointInfoType . . . . .	111
6.74.3	Constructor & Destructor Documentation . . . . .	111
6.74.3.1	DataPoint . . . . .	111
6.74.4	Member Function Documentation . . . . .	112
6.74.4.1	AddChecksumObject . . . . .	112
6.74.4.2	AddLocation . . . . .	112
6.74.4.3	Check . . . . .	112
6.74.4.4	CompareLocationMetadata . . . . .	112
6.74.4.5	CompareMeta . . . . .	112
6.74.4.6	CurrentLocationMetadata . . . . .	113
6.74.4.7	GetFailureReason . . . . .	113
6.74.4.8	List . . . . .	113
6.74.4.9	NextLocation . . . . .	113
6.74.4.10	Passive . . . . .	113
6.74.4.11	PostRegister . . . . .	113
6.74.4.12	PreRegister . . . . .	114
6.74.4.13	PreUnregister . . . . .	114
6.74.4.14	ProvidesMeta . . . . .	114
6.74.4.15	Range . . . . .	114
6.74.4.16	ReadOutOfOrder . . . . .	115
6.74.4.17	Registered . . . . .	115
6.74.4.18	Resolve . . . . .	115
6.74.4.19	SetAdditionalChecks . . . . .	115
6.74.4.20	SetMeta . . . . .	115
6.74.4.21	SetSecure . . . . .	116
6.74.4.22	SortLocations . . . . .	116
6.74.4.23	StartReading . . . . .	116

6.74.4.24	StartWriting	116
6.74.4.25	Stat	117
6.74.4.26	StopReading	117
6.74.4.27	StopWriting	117
6.74.4.28	Unregister	117
6.74.4.29	WriteOutOfOrder	118
6.74.5	Field Documentation	118
6.74.5.1	valid_url_options	118
6.75	Arc::DataPointDirect Class Reference	118
6.75.1	Detailed Description	119
6.75.2	Member Function Documentation	119
6.75.2.1	AddChecksumObject	119
6.75.2.2	AddLocation	119
6.75.2.3	CompareLocationMetadata	120
6.75.2.4	CurrentLocationMetadata	120
6.75.2.5	NextLocation	120
6.75.2.6	Passive	120
6.75.2.7	PostRegister	120
6.75.2.8	PreRegister	120
6.75.2.9	PreUnregister	121
6.75.2.10	ProvidesMeta	121
6.75.2.11	Range	121
6.75.2.12	ReadOutOfOrder	121
6.75.2.13	Registered	122
6.75.2.14	Resolve	122
6.75.2.15	SetAdditionalChecks	122
6.75.2.16	SetSecure	122
6.75.2.17	SortLocations	122
6.75.2.18	Unregister	123
6.75.2.19	WriteOutOfOrder	123
6.76	Arc::DataPointIndex Class Reference	123
6.76.1	Detailed Description	124
6.76.2	Member Function Documentation	124
6.76.2.1	AddChecksumObject	124
6.76.2.2	AddLocation	125
6.76.2.3	Check	125

---

6.76.2.4	CompareLocationMetadata	125
6.76.2.5	CurrentLocationMetadata	125
6.76.2.6	NextLocation	125
6.76.2.7	Passive	125
6.76.2.8	ProvidesMeta	126
6.76.2.9	Range	126
6.76.2.10	ReadOutOfOrder	126
6.76.2.11	Registered	126
6.76.2.12	SetAdditionalChecks	126
6.76.2.13	SetMeta	127
6.76.2.14	SetSecure	127
6.76.2.15	SortLocations	127
6.76.2.16	StartReading	127
6.76.2.17	StartWriting	127
6.76.2.18	StopReading	128
6.76.2.19	StopWriting	128
6.76.2.20	WriteOutOfOrder	128
6.77	Arc::DataPointLoader Class Reference	128
6.78	Arc::DataPointPluginArgument Class Reference	129
6.79	Arc::DataSourceType Class Reference	129
6.80	Arc::DataSpeed Class Reference	129
6.80.1	Detailed Description	130
6.80.2	Constructor & Destructor Documentation	130
6.80.2.1	DataSpeed	130
6.80.2.2	DataSpeed	130
6.80.3	Member Function Documentation	130
6.80.3.1	hold	130
6.80.3.2	set_base	130
6.80.3.3	set_max_data	131
6.80.3.4	set_max_inactivity_time	131
6.80.3.5	set_min_average_speed	131
6.80.3.6	set_min_speed	131
6.80.3.7	set_progress_indicator	131
6.80.3.8	transfer	131
6.80.3.9	verbose	132
6.80.3.10	verbose	132

---

6.81	Arc::DataStagingType Class Reference . . . . .	132
6.82	Arc::DataStatus Class Reference . . . . .	132
6.82.1	Detailed Description . . . . .	132
6.82.2	Member Enumeration Documentation . . . . .	133
6.82.2.1	DataStatusType . . . . .	133
6.83	Arc::DataTargetType Class Reference . . . . .	134
6.84	Arc::DataType Class Reference . . . . .	134
6.85	ArcSec::DateAttribute Class Reference . . . . .	134
6.85.1	Member Function Documentation . . . . .	134
6.85.1.1	encode . . . . .	134
6.85.1.2	equal . . . . .	135
6.85.1.3	getId . . . . .	135
6.85.1.4	getType . . . . .	135
6.86	ArcSec::DateTimeAttribute Class Reference . . . . .	135
6.86.1	Detailed Description . . . . .	135
6.86.2	Member Function Documentation . . . . .	136
6.86.2.1	encode . . . . .	136
6.86.2.2	equal . . . . .	136
6.86.2.3	getId . . . . .	136
6.86.2.4	getType . . . . .	136
6.87	Arc::DBranch Class Reference . . . . .	136
6.88	Arc::DelegationConsumer Class Reference . . . . .	136
6.88.1	Detailed Description . . . . .	137
6.88.2	Constructor & Destructor Documentation . . . . .	137
6.88.2.1	DelegationConsumer . . . . .	137
6.88.2.2	DelegationConsumer . . . . .	137
6.88.3	Member Function Documentation . . . . .	137
6.88.3.1	Acquire . . . . .	137
6.88.3.2	Acquire . . . . .	137
6.88.3.3	Backup . . . . .	137
6.88.3.4	Generate . . . . .	138
6.88.3.5	ID . . . . .	138
6.88.3.6	LogError . . . . .	138
6.88.3.7	Request . . . . .	138
6.88.3.8	Restore . . . . .	138
6.89	Arc::DelegationConsumerSOAP Class Reference . . . . .	138

6.89.1	Detailed Description	139
6.89.2	Constructor & Destructor Documentation	139
6.89.2.1	DelegationConsumerSOAP	139
6.89.2.2	DelegationConsumerSOAP	139
6.89.3	Member Function Documentation	139
6.89.3.1	DelegateCredentialsInit	139
6.89.3.2	DelegatedToken	139
6.89.3.3	UpdateCredentials	139
6.89.3.4	UpdateCredentials	139
6.90	Arc::DelegationContainerSOAP Class Reference	139
6.90.1	Detailed Description	140
6.90.2	Member Function Documentation	140
6.90.2.1	DelegateCredentialsInit	140
6.90.2.2	DelegatedToken	140
6.90.2.3	UpdateCredentials	140
6.90.3	Field Documentation	140
6.90.3.1	context_lock_	140
6.90.3.2	max_duration_	140
6.90.3.3	max_size_	141
6.90.3.4	max_usage_	141
6.90.3.5	restricted_	141
6.91	Arc::DelegationProvider Class Reference	141
6.91.1	Detailed Description	141
6.91.2	Constructor & Destructor Documentation	141
6.91.2.1	DelegationProvider	141
6.91.2.2	DelegationProvider	142
6.91.3	Member Function Documentation	142
6.91.3.1	Delegate	142
6.92	Arc::DelegationProviderSOAP Class Reference	142
6.92.1	Detailed Description	143
6.92.2	Constructor & Destructor Documentation	143
6.92.2.1	DelegationProviderSOAP	143
6.92.2.2	DelegationProviderSOAP	143
6.92.3	Member Function Documentation	143
6.92.3.1	DelegateCredentialsInit	143
6.92.3.2	DelegateCredentialsInit	143

---

6.92.3.3	DelegatedToken	143
6.92.3.4	ID	143
6.92.3.5	UpdateCredentials	144
6.92.3.6	UpdateCredentials	144
6.93	ArcSec::DenyOverridesCombiningAlg Class Reference	144
6.93.1	Detailed Description	144
6.93.2	Member Function Documentation	145
6.93.2.1	combine	145
6.93.2.2	getalgId	145
6.94	Arc::DirectoryType Class Reference	145
6.95	Arc::DiskSpaceRequirementType Class Reference	145
6.96	Arc::DItem Class Reference	146
6.97	Arc::DItemString Class Reference	146
6.98	Arc::DNListHandlerConfig Class Reference	146
6.99	ArcSec::DurationAttribute Class Reference	147
6.99.1	Detailed Description	147
6.99.2	Member Function Documentation	147
6.99.2.1	encode	147
6.99.2.2	equal	147
6.99.2.3	getId	147
6.99.2.4	getType	147
6.100	ArcSec::EqualFunction Class Reference	148
6.100.1	Detailed Description	148
6.100.2	Member Function Documentation	148
6.100.2.1	evaluate	148
6.100.2.2	evaluate	148
6.100.2.3	getFunctionName	149
6.101	ArcSec::EvalResult Struct Reference	149
6.101.1	Detailed Description	149
6.102	ArcSec::EvaluationCtx Class Reference	149
6.102.1	Detailed Description	149
6.102.2	Constructor & Destructor Documentation	149
6.102.2.1	EvaluationCtx	149
6.103	ArcSec::Evaluator Class Reference	150
6.103.1	Detailed Description	150
6.103.2	Member Function Documentation	150

---

6.103.2.1 addPolicy . . . . .	150
6.103.2.2 addPolicy . . . . .	151
6.103.2.3 evaluate . . . . .	151
6.103.2.4 evaluate . . . . .	151
6.103.2.5 evaluate . . . . .	151
6.103.2.6 evaluate . . . . .	151
6.103.2.7 evaluate . . . . .	151
6.103.2.8 evaluate . . . . .	151
6.103.2.9 evaluate . . . . .	151
6.103.2.10getAlgFactory . . . . .	152
6.103.2.11getAttrFactory . . . . .	152
6.103.2.12getFnFactory . . . . .	152
6.103.2.13getName . . . . .	152
6.103.2.14setCombiningAlg . . . . .	152
6.103.2.15setCombiningAlg . . . . .	152
6.104ArcSec::EvaluatorContext Class Reference . . . . .	152
6.104.1 Detailed Description . . . . .	152
6.104.2 Member Function Documentation . . . . .	153
6.104.2.1 operator AlgFactory * . . . . .	153
6.104.2.2 operator AttributeFactory * . . . . .	153
6.104.2.3 operator FnFactory * . . . . .	153
6.105ArcSec::EvaluatorLoader Class Reference . . . . .	153
6.105.1 Detailed Description . . . . .	153
6.105.2 Member Function Documentation . . . . .	153
6.105.2.1 getEvaluator . . . . .	153
6.105.2.2 getEvaluator . . . . .	154
6.105.2.3 getEvaluator . . . . .	154
6.105.2.4 getPolicy . . . . .	154
6.105.2.5 getPolicy . . . . .	154
6.105.2.6 getRequest . . . . .	154
6.105.2.7 getRequest . . . . .	154
6.106Arc::ExecutableType Class Reference . . . . .	154
6.107Arc::ExecutionTarget Class Reference . . . . .	154
6.107.1 Detailed Description . . . . .	155
6.107.2 Constructor & Destructor Documentation . . . . .	155
6.107.2.1 ExecutionTarget . . . . .	155

6.107.2.2 ExecutionTarget . . . . .	155
6.107.2.3 ExecutionTarget . . . . .	155
6.107.3 Member Function Documentation . . . . .	156
6.107.3.1 GetSubmitter . . . . .	156
6.107.3.2 operator= . . . . .	156
6.107.3.3 Print . . . . .	156
6.107.3.4 Update . . . . .	156
6.107.4 Field Documentation . . . . .	156
6.107.4.1 ApplicationEnvironments . . . . .	156
6.107.4.2 ComputingShareName . . . . .	157
6.107.4.3 FreeSlotsWithDuration . . . . .	157
6.107.4.4 MaxDiskSpace . . . . .	157
6.107.4.5 MaxMainMemory . . . . .	157
6.107.4.6 MaxVirtualMemory . . . . .	157
6.107.4.7 OperatingSystem . . . . .	157
6.108 Arc::ExpirationReminder Class Reference . . . . .	158
6.108.1 Detailed Description . . . . .	158
6.108.2 Member Function Documentation . . . . .	158
6.108.2.1 getExpiryTime . . . . .	158
6.108.2.2 getReservationID . . . . .	158
6.108.2.3 operator< . . . . .	158
6.109 Arc::FileCache Class Reference . . . . .	159
6.109.1 Detailed Description . . . . .	159
6.109.2 Constructor & Destructor Documentation . . . . .	160
6.109.2.1 FileCache . . . . .	160
6.109.2.2 FileCache . . . . .	160
6.109.2.3 FileCache . . . . .	160
6.109.2.4 FileCache . . . . .	161
6.109.3 Member Function Documentation . . . . .	161
6.109.3.1 AddDN . . . . .	161
6.109.3.2 CheckCreated . . . . .	161
6.109.3.3 CheckDN . . . . .	161
6.109.3.4 CheckValid . . . . .	161
6.109.3.5 Copy . . . . .	162
6.109.3.6 File . . . . .	162
6.109.3.7 GetCreated . . . . .	162

---

6.109.3.8	GetValid	162
6.109.3.9	Link	162
6.109.3.10	operator bool	162
6.109.3.11	operator==	162
6.109.3.12	Release	162
6.109.3.13	SetValid	163
6.109.3.14	Start	163
6.109.3.15	Stop	163
6.109.3.16	StopAndDelete	163
6.110	FileCacheHash Class Reference	164
6.110.1	Detailed Description	164
6.110.2	Member Function Documentation	164
6.110.2.1	getHash	164
6.110.2.2	maxLength	164
6.111	Arc::FileInfo Class Reference	164
6.111.1	Detailed Description	164
6.112	Arc::FileLock Class Reference	164
6.112.1	Detailed Description	165
6.113	Arc::FileType Class Reference	165
6.114	Arc::FinderLoader Class Reference	165
6.115	ArcSec::FnFactory Class Reference	165
6.115.1	Detailed Description	166
6.115.2	Member Function Documentation	166
6.115.2.1	createFn	166
6.116	ArcSec::Function Class Reference	166
6.116.1	Detailed Description	167
6.116.2	Member Function Documentation	167
6.116.2.1	evaluate	167
6.116.2.2	evaluate	167
6.117	ArcSec::GenericAttribute Class Reference	167
6.117.1	Member Function Documentation	167
6.117.1.1	encode	167
6.117.1.2	equal	168
6.117.1.3	getId	168
6.117.1.4	getType	168
6.118	Arc::GlobusResult Class Reference	168

---

6.119Arc::GSSCredential Class Reference . . . . .	168
6.120Arc::HakaClient Class Reference . . . . .	168
6.120.1 Member Function Documentation . . . . .	169
6.120.1.1 processConsent . . . . .	169
6.120.1.2 processIdP2Confusa . . . . .	169
6.120.1.3 processIdPLogin . . . . .	169
6.121Arc::HTTPClientInfo Struct Reference . . . . .	169
6.122Arc::InfoCache Class Reference . . . . .	169
6.122.1 Detailed Description . . . . .	170
6.122.2 Constructor & Destructor Documentation . . . . .	170
6.122.2.1 InfoCache . . . . .	170
6.123Arc::InfoCacheInterface Class Reference . . . . .	170
6.123.1 Member Function Documentation . . . . .	170
6.123.1.1 Get . . . . .	170
6.124Arc::InfoFilter Class Reference . . . . .	170
6.124.1 Detailed Description . . . . .	171
6.124.2 Constructor & Destructor Documentation . . . . .	171
6.124.2.1 InfoFilter . . . . .	171
6.124.3 Member Function Documentation . . . . .	171
6.124.3.1 Filter . . . . .	171
6.124.3.2 Filter . . . . .	171
6.125Arc::InfoRegister Class Reference . . . . .	171
6.125.1 Detailed Description . . . . .	172
6.126Arc::InfoRegisterContainer Class Reference . . . . .	172
6.126.1 Detailed Description . . . . .	172
6.126.2 Member Function Documentation . . . . .	172
6.126.2.1 addRegistrar . . . . .	172
6.126.2.2 addService . . . . .	172
6.126.2.3 removeService . . . . .	172
6.127Arc::InfoRegisters Class Reference . . . . .	173
6.127.1 Detailed Description . . . . .	173
6.127.2 Constructor & Destructor Documentation . . . . .	173
6.127.2.1 InfoRegisters . . . . .	173
6.128Arc::InfoRegistrar Class Reference . . . . .	173
6.128.1 Detailed Description . . . . .	173
6.128.2 Member Function Documentation . . . . .	174

6.128.2.1 addService . . . . .	174
6.128.2.2 registration . . . . .	174
6.129Arc::InformationContainer Class Reference . . . . .	174
6.129.1 Detailed Description . . . . .	174
6.129.2 Constructor & Destructor Documentation . . . . .	175
6.129.2.1 InformationContainer . . . . .	175
6.129.3 Member Function Documentation . . . . .	175
6.129.3.1 Acquire . . . . .	175
6.129.3.2 Assign . . . . .	175
6.129.3.3 Get . . . . .	175
6.129.4 Field Documentation . . . . .	175
6.129.4.1 doc_ . . . . .	175
6.130Arc::InformationInterface Class Reference . . . . .	175
6.130.1 Detailed Description . . . . .	176
6.130.2 Constructor & Destructor Documentation . . . . .	176
6.130.2.1 InformationInterface . . . . .	176
6.130.3 Member Function Documentation . . . . .	176
6.130.3.1 Get . . . . .	176
6.130.4 Field Documentation . . . . .	176
6.130.4.1 lock_ . . . . .	176
6.131Arc::InformationRequest Class Reference . . . . .	176
6.131.1 Detailed Description . . . . .	177
6.131.2 Constructor & Destructor Documentation . . . . .	177
6.131.2.1 InformationRequest . . . . .	177
6.131.2.2 InformationRequest . . . . .	177
6.131.2.3 InformationRequest . . . . .	177
6.131.2.4 InformationRequest . . . . .	177
6.131.3 Member Function Documentation . . . . .	177
6.131.3.1 SOAP . . . . .	177
6.132Arc::InformationResponse Class Reference . . . . .	178
6.132.1 Detailed Description . . . . .	178
6.132.2 Constructor & Destructor Documentation . . . . .	178
6.132.2.1 InformationResponse . . . . .	178
6.132.3 Member Function Documentation . . . . .	178
6.132.3.1 Result . . . . .	178
6.133Arc::IniConfig Class Reference . . . . .	178

6.134Arc::initializeCredentialsType Class Reference . . . . .	179
6.135ArcSec::InRangeFunction Class Reference . . . . .	179
6.135.1 Member Function Documentation . . . . .	179
6.135.1.1 evaluate . . . . .	179
6.135.1.2 evaluate . . . . .	179
6.136Arc::IntraProcessCounter Class Reference . . . . .	179
6.136.1 Detailed Description . . . . .	180
6.136.2 Constructor & Destructor Documentation . . . . .	180
6.136.2.1 IntraProcessCounter . . . . .	180
6.136.2.2 ~IntraProcessCounter . . . . .	181
6.136.3 Member Function Documentation . . . . .	181
6.136.3.1 cancel . . . . .	181
6.136.3.2 changeExcess . . . . .	181
6.136.3.3 changeLimit . . . . .	181
6.136.3.4 extend . . . . .	182
6.136.3.5 getExcess . . . . .	182
6.136.3.6 getLimit . . . . .	182
6.136.3.7 getValue . . . . .	182
6.136.3.8 reserve . . . . .	183
6.136.3.9 setExcess . . . . .	183
6.136.3.10setLimit . . . . .	183
6.137Arc::ISIS_description Struct Reference . . . . .	184
6.138Arc::IString Class Reference . . . . .	184
6.139Arc::JDLParser Class Reference . . . . .	184
6.140Arc::Job Class Reference . . . . .	184
6.140.1 Detailed Description . . . . .	184
6.140.2 Constructor & Destructor Documentation . . . . .	185
6.140.2.1 Job . . . . .	185
6.140.3 Member Function Documentation . . . . .	185
6.140.3.1 Print . . . . .	185
6.141Arc::JobController Class Reference . . . . .	185
6.141.1 Detailed Description . . . . .	185
6.141.2 Member Function Documentation . . . . .	186
6.141.2.1 FillJobStore . . . . .	186
6.141.2.2 Migrate . . . . .	186
6.141.2.3 PrintJobStatus . . . . .	186

---

6.142Arc::JobControllerLoader Class Reference . . . . .	187
6.142.1 Detailed Description . . . . .	187
6.142.2 Constructor & Destructor Documentation . . . . .	187
6.142.2.1 JobControllerLoader . . . . .	187
6.142.2.2 ~JobControllerLoader . . . . .	187
6.142.3 Member Function Documentation . . . . .	188
6.142.3.1 GetJobControllers . . . . .	188
6.142.3.2 load . . . . .	188
6.143Arc::JobControllerPluginArgument Class Reference . . . . .	188
6.144Arc::JobDescription Class Reference . . . . .	188
6.145Arc::JobDescriptionParser Class Reference . . . . .	189
6.146Arc::JobIdentificationType Class Reference . . . . .	189
6.147Arc::JobMetaType Class Reference . . . . .	189
6.148Arc::JobState Class Reference . . . . .	189
6.148.1 Detailed Description . . . . .	189
6.149Arc::JobSupervisor Class Reference . . . . .	190
6.149.1 Detailed Description . . . . .	190
6.149.2 Constructor & Destructor Documentation . . . . .	190
6.149.2.1 JobSupervisor . . . . .	190
6.149.3 Member Function Documentation . . . . .	190
6.149.3.1 GetJobControllers . . . . .	190
6.150Arc::LoadableModuleDescription Class Reference . . . . .	190
6.151Arc::Loader Class Reference . . . . .	191
6.151.1 Detailed Description . . . . .	191
6.151.2 Constructor & Destructor Documentation . . . . .	191
6.151.2.1 Loader . . . . .	191
6.151.2.2 ~Loader . . . . .	191
6.151.3 Field Documentation . . . . .	191
6.151.3.1 factory_ . . . . .	191
6.152Arc::LogDestination Class Reference . . . . .	192
6.152.1 Detailed Description . . . . .	192
6.152.2 Constructor & Destructor Documentation . . . . .	192
6.152.2.1 LogDestination . . . . .	192
6.152.2.2 LogDestination . . . . .	192
6.153Arc::LogFile Class Reference . . . . .	192
6.153.1 Detailed Description . . . . .	193

6.153.2 Constructor & Destructor Documentation . . . . .	193
6.153.2.1 LogFile . . . . .	193
6.153.2.2 LogFile . . . . .	193
6.153.3 Member Function Documentation . . . . .	194
6.153.3.1 log . . . . .	194
6.153.3.2 setBackups . . . . .	194
6.153.3.3 setMaxSize . . . . .	194
6.153.3.4 setReopen . . . . .	194
6.154Arc::Logger Class Reference . . . . .	195
6.154.1 Detailed Description . . . . .	195
6.154.2 Constructor & Destructor Documentation . . . . .	195
6.154.2.1 Logger . . . . .	195
6.154.2.2 Logger . . . . .	196
6.154.2.3 ~Logger . . . . .	196
6.154.3 Member Function Documentation . . . . .	196
6.154.3.1 addDestination . . . . .	196
6.154.3.2 getRootLogger . . . . .	196
6.154.3.3 getThreshold . . . . .	196
6.154.3.4 msg . . . . .	196
6.154.3.5 msg . . . . .	197
6.154.3.6 setThreshold . . . . .	197
6.155Arc::LoggerFormat Struct Reference . . . . .	197
6.156Arc::LogMessage Class Reference . . . . .	197
6.156.1 Detailed Description . . . . .	198
6.156.2 Constructor & Destructor Documentation . . . . .	198
6.156.2.1 LogMessage . . . . .	198
6.156.2.2 LogMessage . . . . .	198
6.156.3 Member Function Documentation . . . . .	198
6.156.3.1 getLevel . . . . .	198
6.156.3.2 setIdentifier . . . . .	199
6.156.4 Friends And Related Function Documentation . . . . .	199
6.156.4.1 Logger . . . . .	199
6.156.4.2 operator<< . . . . .	199
6.157Arc::LogStream Class Reference . . . . .	199
6.157.1 Detailed Description . . . . .	200
6.157.2 Constructor & Destructor Documentation . . . . .	200

---

6.157.2.1	LogStream . . . . .	200
6.157.2.2	LogStream . . . . .	200
6.157.3	Member Function Documentation . . . . .	200
6.157.3.1	log . . . . .	200
6.158	ArcSec::MatchFunction Class Reference . . . . .	201
6.158.1	Detailed Description . . . . .	201
6.158.2	Member Function Documentation . . . . .	201
6.158.2.1	evaluate . . . . .	201
6.158.2.2	evaluate . . . . .	201
6.158.2.3	getFunctionName . . . . .	201
6.159	Arc::MCC Class Reference . . . . .	202
6.159.1	Detailed Description . . . . .	202
6.159.2	Constructor & Destructor Documentation . . . . .	203
6.159.2.1	MCC . . . . .	203
6.159.3	Member Function Documentation . . . . .	203
6.159.3.1	AddSecHandler . . . . .	203
6.159.3.2	Next . . . . .	203
6.159.3.3	process . . . . .	203
6.159.3.4	ProcessSecHandlers . . . . .	203
6.159.3.5	Unlink . . . . .	203
6.159.4	Field Documentation . . . . .	204
6.159.4.1	logger . . . . .	204
6.159.4.2	next_ . . . . .	204
6.159.4.3	sechandlers_ . . . . .	204
6.160	Arc::MCC_Status Class Reference . . . . .	204
6.160.1	Detailed Description . . . . .	204
6.160.2	Constructor & Destructor Documentation . . . . .	205
6.160.2.1	MCC_Status . . . . .	205
6.160.3	Member Function Documentation . . . . .	205
6.160.3.1	getExplanation . . . . .	205
6.160.3.2	getKind . . . . .	205
6.160.3.3	getOrigin . . . . .	205
6.160.3.4	isOk . . . . .	205
6.160.3.5	operator bool . . . . .	206
6.160.3.6	operator std::string . . . . .	206
6.160.3.7	operator! . . . . .	206

6.161Arc::MCCConfig Class Reference . . . . .	206
6.161.1 Member Function Documentation . . . . .	207
6.161.1.1 MakeConfig . . . . .	207
6.162Arc::MCCInterface Class Reference . . . . .	207
6.162.1 Detailed Description . . . . .	207
6.162.2 Member Function Documentation . . . . .	207
6.162.2.1 process . . . . .	207
6.163Arc::MCCLoader Class Reference . . . . .	208
6.163.1 Detailed Description . . . . .	208
6.163.2 Constructor & Destructor Documentation . . . . .	209
6.163.2.1 MCCLoader . . . . .	209
6.163.2.2 ~MCCLoader . . . . .	209
6.163.3 Member Function Documentation . . . . .	209
6.163.3.1 operator[] . . . . .	209
6.164Arc::MCCPluginArgument Class Reference . . . . .	209
6.165Arc::MD5Sum Class Reference . . . . .	209
6.165.1 Detailed Description . . . . .	210
6.166Arc::MemoryAllocationException Class Reference . . . . .	210
6.167Arc::Message Class Reference . . . . .	210
6.167.1 Detailed Description . . . . .	210
6.167.2 Constructor & Destructor Documentation . . . . .	211
6.167.2.1 Message . . . . .	211
6.167.2.2 Message . . . . .	211
6.167.2.3 Message . . . . .	211
6.167.2.4 ~Message . . . . .	211
6.167.3 Member Function Documentation . . . . .	211
6.167.3.1 Attributes . . . . .	211
6.167.3.2 Auth . . . . .	211
6.167.3.3 AuthContext . . . . .	211
6.167.3.4 AuthContext . . . . .	212
6.167.3.5 Context . . . . .	212
6.167.3.6 Context . . . . .	212
6.167.3.7 operator= . . . . .	212
6.167.3.8 Payload . . . . .	212
6.167.3.9 Payload . . . . .	212
6.168Arc::MessageAttributes Class Reference . . . . .	212

---

6.168.1 Detailed Description . . . . .	213
6.168.2 Constructor & Destructor Documentation . . . . .	213
6.168.2.1 MessageAttributes . . . . .	213
6.168.3 Member Function Documentation . . . . .	213
6.168.3.1 add . . . . .	213
6.168.3.2 count . . . . .	213
6.168.3.3 get . . . . .	214
6.168.3.4 getAll . . . . .	214
6.168.3.5 remove . . . . .	214
6.168.3.6 removeAll . . . . .	214
6.168.3.7 set . . . . .	215
6.168.4 Field Documentation . . . . .	215
6.168.4.1 attributes_ . . . . .	215
6.169Arc::MessageAuth Class Reference . . . . .	215
6.169.1 Detailed Description . . . . .	216
6.169.2 Member Function Documentation . . . . .	216
6.169.2.1 Export . . . . .	216
6.169.2.2 Filter . . . . .	216
6.170Arc::MessageAuthContext Class Reference . . . . .	216
6.170.1 Detailed Description . . . . .	216
6.171Arc::MessageContext Class Reference . . . . .	217
6.171.1 Detailed Description . . . . .	217
6.171.2 Member Function Documentation . . . . .	217
6.171.2.1 Add . . . . .	217
6.172Arc::MessageContextElement Class Reference . . . . .	217
6.172.1 Detailed Description . . . . .	217
6.173Arc::MessagePayload Class Reference . . . . .	218
6.173.1 Detailed Description . . . . .	218
6.174Arc::ModuleDesc Class Reference . . . . .	218
6.174.1 Detailed Description . . . . .	218
6.175Arc::ModuleManager Class Reference . . . . .	218
6.175.1 Detailed Description . . . . .	219
6.175.2 Constructor & Destructor Documentation . . . . .	219
6.175.2.1 ModuleManager . . . . .	219
6.175.3 Member Function Documentation . . . . .	219
6.175.3.1 find . . . . .	219

---

6.175.3.2 findLocation . . . . .	219
6.175.3.3 load . . . . .	219
6.175.3.4 makePersistent . . . . .	220
6.175.3.5 makePersistent . . . . .	220
6.175.3.6 reload . . . . .	220
6.175.3.7 setCfg . . . . .	220
6.175.3.8 unload . . . . .	220
6.175.3.9 unload . . . . .	220
6.176Arc::MultiSecAttr Class Reference . . . . .	220
6.176.1 Detailed Description . . . . .	221
6.176.2 Member Function Documentation . . . . .	221
6.176.2.1 Export . . . . .	221
6.176.2.2 operator bool . . . . .	221
6.177Arc::MySQLDatabase Class Reference . . . . .	221
6.177.1 Detailed Description . . . . .	222
6.177.2 Member Function Documentation . . . . .	222
6.177.2.1 close . . . . .	222
6.177.2.2 connect . . . . .	222
6.177.2.3 enable_ssl . . . . .	222
6.177.2.4 isconnected . . . . .	222
6.177.2.5 shutdown . . . . .	222
6.178Arc::MySQLQuery Class Reference . . . . .	223
6.178.1 Member Function Documentation . . . . .	223
6.178.1.1 execute . . . . .	223
6.178.1.2 get_array . . . . .	223
6.178.1.3 get_num_columns . . . . .	224
6.178.1.4 get_num_rows . . . . .	224
6.178.1.5 get_row . . . . .	224
6.178.1.6 get_row . . . . .	224
6.178.1.7 get_row_field . . . . .	224
6.179Arc::NotificationType Class Reference . . . . .	225
6.180Arc::NS Class Reference . . . . .	225
6.181Arc::OAuthConsumer Class Reference . . . . .	225
6.181.1 Detailed Description . . . . .	225
6.181.2 Constructor & Destructor Documentation . . . . .	226
6.181.2.1 OAuthConsumer . . . . .	226

---

6.181.3 Member Function Documentation . . . . .	226
6.181.3.1 approveCSR . . . . .	226
6.181.3.2 parseDN . . . . .	226
6.181.3.3 processLogin . . . . .	226
6.181.3.4 pushCSR . . . . .	226
6.181.3.5 storeCert . . . . .	226
6.182Arc::OpenIdpClient Class Reference . . . . .	226
6.182.1 Member Function Documentation . . . . .	227
6.182.1.1 processConsent . . . . .	227
6.182.1.2 processIdP2Confusa . . . . .	227
6.182.1.3 processIdPLogin . . . . .	227
6.183Arc::OptionParser Class Reference . . . . .	227
6.184ArcSec::OrderedCombiningAlg Class Reference . . . . .	228
6.185passwd Struct Reference . . . . .	228
6.186Arc::PathIterator Class Reference . . . . .	228
6.186.1 Detailed Description . . . . .	228
6.186.2 Constructor & Destructor Documentation . . . . .	228
6.186.2.1 PathIterator . . . . .	228
6.186.3 Member Function Documentation . . . . .	229
6.186.3.1 operator bool . . . . .	229
6.186.3.2 operator* . . . . .	229
6.186.3.3 operator++ . . . . .	229
6.186.3.4 operator-- . . . . .	229
6.186.3.5 Rest . . . . .	229
6.187Arc::PayloadRaw Class Reference . . . . .	229
6.187.1 Detailed Description . . . . .	230
6.187.2 Constructor & Destructor Documentation . . . . .	230
6.187.2.1 PayloadRaw . . . . .	230
6.187.2.2 ~PayloadRaw . . . . .	230
6.187.3 Member Function Documentation . . . . .	230
6.187.3.1 Buffer . . . . .	230
6.187.3.2 BufferPos . . . . .	230
6.187.3.3 BufferSize . . . . .	230
6.187.3.4 Content . . . . .	230
6.187.3.5 Insert . . . . .	231
6.187.3.6 Insert . . . . .	231

6.187.3.7 operator[] . . . . .	231
6.187.3.8 Size . . . . .	231
6.187.3.9 Truncate . . . . .	231
6.188Arc::PayloadRawBuf Struct Reference . . . . .	231
6.188.1 Field Documentation . . . . .	232
6.188.1.1 allocated . . . . .	232
6.188.1.2 length . . . . .	232
6.188.1.3 size . . . . .	232
6.189Arc::PayloadRawInterface Class Reference . . . . .	232
6.189.1 Detailed Description . . . . .	233
6.189.2 Member Function Documentation . . . . .	233
6.189.2.1 Buffer . . . . .	233
6.189.2.2 BufferPos . . . . .	233
6.189.2.3 BufferSize . . . . .	233
6.189.2.4 Content . . . . .	233
6.189.2.5 Insert . . . . .	233
6.189.2.6 Insert . . . . .	233
6.189.2.7 operator[] . . . . .	234
6.189.2.8 Size . . . . .	234
6.189.2.9 Truncate . . . . .	234
6.190Arc::PayloadSOAP Class Reference . . . . .	234
6.190.1 Detailed Description . . . . .	234
6.190.2 Constructor & Destructor Documentation . . . . .	235
6.190.2.1 PayloadSOAP . . . . .	235
6.190.2.2 PayloadSOAP . . . . .	235
6.190.2.3 PayloadSOAP . . . . .	235
6.191Arc::PayloadStream Class Reference . . . . .	235
6.191.1 Detailed Description . . . . .	236
6.191.2 Constructor & Destructor Documentation . . . . .	236
6.191.2.1 PayloadStream . . . . .	236
6.191.2.2 ~PayloadStream . . . . .	236
6.191.3 Member Function Documentation . . . . .	236
6.191.3.1 Get . . . . .	236
6.191.3.2 Get . . . . .	236
6.191.3.3 Get . . . . .	236
6.191.3.4 Limit . . . . .	237

6.191.3.5 operator bool . . . . .	237
6.191.3.6 operator! . . . . .	237
6.191.3.7 Pos . . . . .	237
6.191.3.8 Put . . . . .	237
6.191.3.9 Put . . . . .	237
6.191.3.10Put . . . . .	237
6.191.3.11Size . . . . .	238
6.191.3.12Timeout . . . . .	238
6.191.3.13Timeout . . . . .	238
6.191.4 Field Documentation . . . . .	238
6.191.4.1 handle_ . . . . .	238
6.191.4.2 seekable_ . . . . .	238
6.192Arc::PayloadStreamInterface Class Reference . . . . .	238
6.192.1 Detailed Description . . . . .	239
6.192.2 Member Function Documentation . . . . .	239
6.192.2.1 Get . . . . .	239
6.192.2.2 Get . . . . .	239
6.192.2.3 Get . . . . .	239
6.192.2.4 Limit . . . . .	239
6.192.2.5 operator bool . . . . .	240
6.192.2.6 operator! . . . . .	240
6.192.2.7 Pos . . . . .	240
6.192.2.8 Put . . . . .	240
6.192.2.9 Put . . . . .	240
6.192.2.10Put . . . . .	240
6.192.2.11Size . . . . .	240
6.192.2.12Timeout . . . . .	240
6.192.2.13Timeout . . . . .	241
6.193Arc::PayloadWSRF Class Reference . . . . .	241
6.193.1 Detailed Description . . . . .	241
6.193.2 Constructor & Destructor Documentation . . . . .	241
6.193.2.1 PayloadWSRF . . . . .	241
6.193.2.2 PayloadWSRF . . . . .	241
6.193.2.3 PayloadWSRF . . . . .	242
6.194ArcSec::PDP Class Reference . . . . .	242
6.194.1 Detailed Description . . . . .	242

6.195ArcSec::PDPCfgContext Class Reference . . . . .	242
6.196ArcSec::PDPluginArgument Class Reference . . . . .	243
6.197Arc::Period Class Reference . . . . .	243
6.197.1 Constructor & Destructor Documentation . . . . .	243
6.197.1.1 Period . . . . .	243
6.197.1.2 Period . . . . .	243
6.197.1.3 Period . . . . .	243
6.197.1.4 Period . . . . .	244
6.197.2 Member Function Documentation . . . . .	244
6.197.2.1 GetPeriod . . . . .	244
6.197.2.2 istr . . . . .	244
6.197.2.3 operator std::string . . . . .	244
6.197.2.4 operator!= . . . . .	244
6.197.2.5 operator< . . . . .	244
6.197.2.6 operator<= . . . . .	244
6.197.2.7 operator= . . . . .	244
6.197.2.8 operator= . . . . .	244
6.197.2.9 operator== . . . . .	244
6.197.2.10operator> . . . . .	244
6.197.2.11operator>= . . . . .	245
6.197.2.12SetPeriod . . . . .	245
6.198ArcSec::PeriodAttribute Class Reference . . . . .	245
6.198.1 Detailed Description . . . . .	245
6.198.2 Member Function Documentation . . . . .	245
6.198.2.1 encode . . . . .	245
6.198.2.2 equal . . . . .	245
6.198.2.3 getId . . . . .	246
6.198.2.4 getType . . . . .	246
6.199ArcSec::PermitOverridesCombiningAlg Class Reference . . . . .	246
6.199.1 Detailed Description . . . . .	246
6.199.2 Member Function Documentation . . . . .	246
6.199.2.1 combine . . . . .	246
6.199.2.2 getalgId . . . . .	247
6.200Arc::Plexer Class Reference . . . . .	247
6.200.1 Detailed Description . . . . .	248
6.200.2 Constructor & Destructor Documentation . . . . .	248

---

6.200.2.1 Plexer . . . . .	248
6.200.2.2 ~Plexer . . . . .	248
6.200.3 Member Function Documentation . . . . .	248
6.200.3.1 Next . . . . .	248
6.200.3.2 process . . . . .	248
6.200.4 Field Documentation . . . . .	248
6.200.4.1 logger . . . . .	248
6.201 Arc::PlexerEntry Class Reference . . . . .	249
6.201.1 Detailed Description . . . . .	249
6.202 Arc::Plugin Class Reference . . . . .	249
6.202.1 Detailed Description . . . . .	250
6.203 Arc::PluginArgument Class Reference . . . . .	250
6.203.1 Detailed Description . . . . .	251
6.203.2 Member Function Documentation . . . . .	251
6.203.2.1 get_factory . . . . .	251
6.203.2.2 get_module . . . . .	251
6.204 Arc::PluginDesc Class Reference . . . . .	252
6.204.1 Detailed Description . . . . .	252
6.205 Arc::PluginDescriptor Struct Reference . . . . .	252
6.205.1 Detailed Description . . . . .	252
6.206 Arc::PluginsFactory Class Reference . . . . .	252
6.206.1 Detailed Description . . . . .	253
6.206.2 Constructor & Destructor Documentation . . . . .	253
6.206.2.1 PluginsFactory . . . . .	253
6.206.3 Member Function Documentation . . . . .	253
6.206.3.1 load . . . . .	253
6.206.3.2 report . . . . .	253
6.206.3.3 scan . . . . .	253
6.206.3.4 TryLoad . . . . .	254
6.207 ArcSec::Policy Class Reference . . . . .	254
6.207.1 Detailed Description . . . . .	255
6.207.2 Constructor & Destructor Documentation . . . . .	255
6.207.2.1 Policy . . . . .	255
6.207.2.2 Policy . . . . .	255
6.207.3 Member Function Documentation . . . . .	255
6.207.3.1 addPolicy . . . . .	255

6.207.3.2 eval . . . . .	255
6.207.3.3 getEffect . . . . .	255
6.207.3.4 getEvalName . . . . .	255
6.207.3.5 getEvalResult . . . . .	255
6.207.3.6 getName . . . . .	256
6.207.3.7 make_policy . . . . .	256
6.207.3.8 setEvalResult . . . . .	256
6.207.3.9 setEvaluatorContext . . . . .	256
6.208ArcSec::PolicyStore::PolicyElement Class Reference . . . . .	256
6.209ArcSec::PolicyParser Class Reference . . . . .	256
6.209.1 Detailed Description . . . . .	256
6.209.2 Member Function Documentation . . . . .	257
6.209.2.1 parsePolicy . . . . .	257
6.210ArcSec::PolicyStore Class Reference . . . . .	257
6.210.1 Detailed Description . . . . .	257
6.210.2 Constructor & Destructor Documentation . . . . .	257
6.210.2.1 PolicyStore . . . . .	257
6.211Arc::PrintF< T0, T1, T2, T3, T4, T5, T6, T7 > Class Template Reference . . . . .	258
6.212Arc::PrintFBase Class Reference . . . . .	258
6.213Arc::Profile Class Reference . . . . .	258
6.214ArcCredential::PROXYCERTINFO_st Struct Reference . . . . .	259
6.215ArcCredential::PROXYPOLICY_st Struct Reference . . . . .	259
6.216Arc::Query Class Reference . . . . .	259
6.216.1 Constructor & Destructor Documentation . . . . .	259
6.216.1.1 Query . . . . .	259
6.216.1.2 Query . . . . .	259
6.216.1.3 ~Query . . . . .	260
6.216.2 Member Function Documentation . . . . .	260
6.216.2.1 execute . . . . .	260
6.216.2.2 get_array . . . . .	260
6.216.2.3 get_num_columns . . . . .	260
6.216.2.4 get_num_rows . . . . .	260
6.216.2.5 get_row . . . . .	260
6.216.2.6 get_row . . . . .	261
6.216.2.7 get_row_field . . . . .	261
6.217Arc::Range< T > Class Template Reference . . . . .	261

6.218Arc::Register_Info_Type Struct Reference . . . . .	261
6.219Arc::RegisteredService Class Reference . . . . .	262
6.219.1 Detailed Description . . . . .	262
6.219.2 Constructor & Destructor Documentation . . . . .	262
6.219.2.1 RegisteredService . . . . .	262
6.220Arc::RegularExpression Class Reference . . . . .	262
6.220.1 Detailed Description . . . . .	263
6.220.2 Member Function Documentation . . . . .	263
6.220.2.1 match . . . . .	263
6.221ArcSec::Request Class Reference . . . . .	263
6.221.1 Detailed Description . . . . .	264
6.221.2 Constructor & Destructor Documentation . . . . .	264
6.221.2.1 Request . . . . .	264
6.221.2.2 Request . . . . .	264
6.221.3 Member Function Documentation . . . . .	264
6.221.3.1 addRequestItem . . . . .	264
6.221.3.2 getEvalName . . . . .	264
6.221.3.3 getName . . . . .	264
6.221.3.4 getRequestItems . . . . .	264
6.221.3.5 make_request . . . . .	264
6.221.3.6 setAttributeFactory . . . . .	265
6.221.3.7 setRequestItems . . . . .	265
6.222ArcSec::RequestAttribute Class Reference . . . . .	265
6.222.1 Detailed Description . . . . .	265
6.222.2 Constructor & Destructor Documentation . . . . .	265
6.222.2.1 RequestAttribute . . . . .	265
6.222.3 Member Function Documentation . . . . .	265
6.222.3.1 duplicate . . . . .	265
6.223ArcSec::RequestItem Class Reference . . . . .	266
6.223.1 Detailed Description . . . . .	266
6.223.2 Constructor & Destructor Documentation . . . . .	266
6.223.2.1 RequestItem . . . . .	266
6.224ArcSec::RequestTuple Class Reference . . . . .	266
6.225Arc::ResourceSlotType Class Reference . . . . .	266
6.226Arc::ResourcesType Class Reference . . . . .	266
6.227Arc::ResourceTargetType Class Reference . . . . .	267

6.228ArcSec::Response Class Reference . . . . .	267
6.228.1 Detailed Description . . . . .	267
6.229ArcSec::ResponseItem Class Reference . . . . .	267
6.229.1 Detailed Description . . . . .	267
6.230ArcSec::ResponseList Class Reference . . . . .	267
6.231Arc::RSL Class Reference . . . . .	267
6.232Arc::RSLBoolean Class Reference . . . . .	268
6.233Arc::RSLConcat Class Reference . . . . .	268
6.234Arc::RSLCondition Class Reference . . . . .	268
6.235Arc::RSLList Class Reference . . . . .	269
6.236Arc::RSLLiteral Class Reference . . . . .	269
6.237Arc::RSLParser Class Reference . . . . .	269
6.238Arc::RSLSequence Class Reference . . . . .	270
6.239Arc::RSLValue Class Reference . . . . .	270
6.240Arc::RSLVariable Class Reference . . . . .	270
6.241Arc::Run Class Reference . . . . .	270
6.241.1 Detailed Description . . . . .	271
6.241.2 Constructor & Destructor Documentation . . . . .	271
6.241.2.1 Run . . . . .	271
6.241.2.2 Run . . . . .	271
6.241.2.3 ~Run . . . . .	272
6.241.3 Member Function Documentation . . . . .	272
6.241.3.1 Abandon . . . . .	272
6.241.3.2 AfterFork . . . . .	272
6.241.3.3 AssignStderr . . . . .	272
6.241.3.4 AssignStdin . . . . .	272
6.241.3.5 AssignStdout . . . . .	272
6.241.3.6 AssignWorkingDirectory . . . . .	272
6.241.3.7 CloseStderr . . . . .	272
6.241.3.8 CloseStdin . . . . .	272
6.241.3.9 CloseStdout . . . . .	272
6.241.3.10KeepStderr . . . . .	273
6.241.3.11KeepStdin . . . . .	273
6.241.3.12KeepStdout . . . . .	273
6.241.3.13Kill . . . . .	273
6.241.3.14operator bool . . . . .	273

6.241.3.15	operator!	273
6.241.3.16	ReadStderr	273
6.241.3.17	ReadStdout	273
6.241.3.18	Result	273
6.241.3.19	Running	273
6.241.3.20	Start	274
6.241.3.21	Wait	274
6.241.3.22	Wait	274
6.241.3.23	WriteStdin	274
6.242	Arc::SAML2LoginClient Class Reference	274
6.242.1	Constructor & Destructor Documentation	274
6.242.1.1	SAML2LoginClient	274
6.242.2	Member Function Documentation	275
6.242.2.1	findSimpleSAMLInstallation	275
6.242.2.2	processLogin	275
6.243	Arc::SAML2SSOHTTPClient Class Reference	275
6.243.1	Member Function Documentation	276
6.243.1.1	approveCSR	276
6.243.1.2	parseDN	276
6.243.1.3	processConsent	276
6.243.1.4	processIdP2Confusa	276
6.243.1.5	processIdPLogin	276
6.243.1.6	processLogin	276
6.243.1.7	pushCSR	276
6.243.1.8	storeCert	277
6.244	Arc::SAMLToken Class Reference	277
6.244.1	Detailed Description	277
6.244.2	Member Enumeration Documentation	278
6.244.2.1	SAMLVersion	278
6.244.3	Constructor & Destructor Documentation	278
6.244.3.1	SAMLToken	278
6.244.3.2	SAMLToken	278
6.244.3.3	~SAMLToken	279
6.244.4	Member Function Documentation	279
6.244.4.1	Authenticate	279
6.244.4.2	Authenticate	279

6.244.4.3 operator bool . . . . .	279
6.245 Arc::ScalableTime< T > Class Template Reference . . . . .	279
6.246 Arc::ScalableTime< int > Class Template Reference . . . . .	280
6.247 Arc::SecAttr Class Reference . . . . .	280
6.247.1 Detailed Description . . . . .	280
6.247.2 Member Function Documentation . . . . .	281
6.247.2.1 Export . . . . .	281
6.247.2.2 Export . . . . .	281
6.247.2.3 Import . . . . .	281
6.247.2.4 operator bool . . . . .	281
6.247.2.5 operator!= . . . . .	281
6.247.2.6 operator== . . . . .	281
6.248 Arc::SecAttrFormat Class Reference . . . . .	281
6.248.1 Detailed Description . . . . .	282
6.249 Arc::SecAttrValue Class Reference . . . . .	282
6.249.1 Detailed Description . . . . .	282
6.249.2 Member Function Documentation . . . . .	282
6.249.2.1 operator bool . . . . .	282
6.249.2.2 operator!= . . . . .	282
6.249.2.3 operator== . . . . .	283
6.250 ArcSec::SecHandler Class Reference . . . . .	283
6.250.1 Detailed Description . . . . .	283
6.251 Arc::SecHandlerConfig Class Reference . . . . .	283
6.252 ArcSec::SecHandlerConfig Class Reference . . . . .	284
6.252.1 Detailed Description . . . . .	284
6.253 ArcSec::SecHandlerPluginArgument Class Reference . . . . .	284
6.254 ArcSec::Security Class Reference . . . . .	284
6.254.1 Detailed Description . . . . .	284
6.255 Arc::Service Class Reference . . . . .	285
6.255.1 Detailed Description . . . . .	285
6.255.2 Constructor & Destructor Documentation . . . . .	286
6.255.2.1 Service . . . . .	286
6.255.3 Member Function Documentation . . . . .	286
6.255.3.1 AddSecHandler . . . . .	286
6.255.3.2 getID . . . . .	286
6.255.3.3 ProcessSecHandlers . . . . .	286

---

6.255.3.4 RegistrationCollector . . . . .	286
6.255.4 Field Documentation . . . . .	286
6.255.4.1 logger . . . . .	286
6.255.4.2 sechandlers_ . . . . .	287
6.256Arc::ServicePluginArgument Class Reference . . . . .	287
6.257Arc::SimpleCondition Class Reference . . . . .	287
6.257.1 Detailed Description . . . . .	287
6.257.2 Member Function Documentation . . . . .	288
6.257.2.1 broadcast . . . . .	288
6.257.2.2 lock . . . . .	288
6.257.2.3 reset . . . . .	288
6.257.2.4 signal . . . . .	288
6.257.2.5 signal_nonblock . . . . .	288
6.257.2.6 unlock . . . . .	288
6.257.2.7 wait . . . . .	288
6.257.2.8 wait . . . . .	288
6.257.2.9 wait_nonblock . . . . .	288
6.258Arc::SimpleCounter Class Reference . . . . .	288
6.258.1 Member Function Documentation . . . . .	289
6.258.1.1 wait . . . . .	289
6.259Arc::SOAPMessage Class Reference . . . . .	289
6.259.1 Detailed Description . . . . .	289
6.259.2 Constructor & Destructor Documentation . . . . .	289
6.259.2.1 SOAPMessage . . . . .	289
6.259.2.2 SOAPMessage . . . . .	289
6.259.2.3 SOAPMessage . . . . .	289
6.259.2.4 ~SOAPMessage . . . . .	289
6.259.3 Member Function Documentation . . . . .	290
6.259.3.1 Attributes . . . . .	290
6.259.3.2 Payload . . . . .	290
6.259.3.3 Payload . . . . .	290
6.260Arc::Software Class Reference . . . . .	290
6.260.1 Detailed Description . . . . .	291
6.260.2 Member Typedef Documentation . . . . .	291
6.260.2.1 ComparisonOperator . . . . .	291
6.260.3 Member Enumeration Documentation . . . . .	292

6.260.3.1 ComparisonOperatorEnum . . . . .	292
6.260.4 Constructor & Destructor Documentation . . . . .	292
6.260.4.1 Software . . . . .	292
6.260.4.2 Software . . . . .	292
6.260.4.3 Software . . . . .	292
6.260.4.4 Software . . . . .	293
6.260.5 Member Function Documentation . . . . .	293
6.260.5.1 convert . . . . .	293
6.260.5.2 empty . . . . .	293
6.260.5.3 getFamily . . . . .	293
6.260.5.4 getName . . . . .	294
6.260.5.5 getVersion . . . . .	294
6.260.5.6 operator std::string . . . . .	294
6.260.5.7 operator!= . . . . .	294
6.260.5.8 operator() . . . . .	294
6.260.5.9 operator< . . . . .	295
6.260.5.10operator<= . . . . .	295
6.260.5.11operator== . . . . .	295
6.260.5.12operator> . . . . .	296
6.260.5.13operator>= . . . . .	296
6.260.5.14toString . . . . .	297
6.260.6 Friends And Related Function Documentation . . . . .	297
6.260.6.1 operator<< . . . . .	297
6.260.7 Field Documentation . . . . .	297
6.260.7.1 VERSIONTOKENS . . . . .	297
6.261 Arc::SoftwareRequirement Class Reference . . . . .	297
6.261.1 Detailed Description . . . . .	298
6.261.2 Constructor & Destructor Documentation . . . . .	298
6.261.2.1 SoftwareRequirement . . . . .	298
6.261.2.2 SoftwareRequirement . . . . .	299
6.261.2.3 SoftwareRequirement . . . . .	299
6.261.2.4 SoftwareRequirement . . . . .	299
6.261.3 Member Function Documentation . . . . .	299
6.261.3.1 add . . . . .	299
6.261.3.2 add . . . . .	300
6.261.3.3 clear . . . . .	300

6.261.3.4 empty	300
6.261.3.5 getComparisonOperatorList	300
6.261.3.6 getSoftwareList	301
6.261.3.7 isRequiringAll	301
6.261.3.8 isResolved	301
6.261.3.9 isSatisfied	301
6.261.3.10 isSatisfied	302
6.261.3.11 isSatisfied	302
6.261.3.12 operator=	303
6.261.3.13 selectSoftware	303
6.261.3.14 selectSoftware	304
6.261.3.15 selectSoftware	304
6.261.3.16 setRequirement	304
6.262 ArcSec::Source Class Reference	305
6.262.1 Detailed Description	305
6.262.2 Constructor & Destructor Documentation	305
6.262.2.1 Source	305
6.262.2.2 Source	306
6.263 ArcSec::SourceFile Class Reference	306
6.263.1 Detailed Description	306
6.264 ArcSec::SourceURL Class Reference	306
6.264.1 Detailed Description	307
6.265 ArcSec::StringAttribute Class Reference	307
6.265.1 Member Function Documentation	307
6.265.1.1 encode	307
6.265.1.2 equal	307
6.265.1.3 getId	308
6.265.1.4 getType	308
6.266 Arc::Submitter Class Reference	308
6.266.1 Detailed Description	308
6.266.2 Member Function Documentation	308
6.266.2.1 Migrate	308
6.266.2.2 Submit	309
6.267 Arc::SubmitterLoader Class Reference	309
6.267.1 Detailed Description	309
6.267.2 Constructor & Destructor Documentation	309

---

6.267.2.1 SubmitterLoader . . . . .	309
6.267.2.2 ~SubmitterLoader . . . . .	309
6.267.3 Member Function Documentation . . . . .	310
6.267.3.1 GetSubmitters . . . . .	310
6.267.3.2 load . . . . .	310
6.268 Arc::SubmitterPluginArgument Class Reference . . . . .	310
6.269 Arc::TargetGenerator Class Reference . . . . .	310
6.269.1 Detailed Description . . . . .	311
6.269.2 Constructor & Destructor Documentation . . . . .	311
6.269.2.1 TargetGenerator . . . . .	311
6.269.3 Member Function Documentation . . . . .	311
6.269.3.1 AddIndexServer . . . . .	311
6.269.3.2 AddJob . . . . .	312
6.269.3.3 AddService . . . . .	312
6.269.3.4 AddTarget . . . . .	312
6.269.3.5 FoundJobs . . . . .	312
6.269.3.6 FoundTargets . . . . .	312
6.269.3.7 GetTargets . . . . .	312
6.269.3.8 ModifyFoundTargets . . . . .	313
6.269.3.9 PrintTargetInfo . . . . .	313
6.269.3.10 ServiceCounter . . . . .	313
6.270 Arc::TargetRetriever Class Reference . . . . .	313
6.270.1 Detailed Description . . . . .	314
6.270.2 Constructor & Destructor Documentation . . . . .	314
6.270.2.1 TargetRetriever . . . . .	314
6.270.3 Member Function Documentation . . . . .	314
6.270.3.1 GetTargets . . . . .	314
6.271 Arc::TargetRetrieverLoader Class Reference . . . . .	315
6.271.1 Detailed Description . . . . .	315
6.271.2 Constructor & Destructor Documentation . . . . .	315
6.271.2.1 TargetRetrieverLoader . . . . .	315
6.271.2.2 ~TargetRetrieverLoader . . . . .	315
6.271.3 Member Function Documentation . . . . .	315
6.271.3.1 GetTargetRetrievers . . . . .	315
6.271.3.2 load . . . . .	316
6.272 Arc::TargetRetrieverPluginArgument Class Reference . . . . .	316

---

6.273	Test::TestMCC Class Reference	316
6.274	Test::TestService Class Reference	317
6.274.1	Member Function Documentation	317
6.274.1.1	process	317
6.275	Arc::ThreadInitializer Class Reference	318
6.276	Arc::ThreadRegistry Class Reference	318
6.276.1	Detailed Description	318
6.276.2	Member Function Documentation	318
6.276.2.1	WaitForExit	318
6.276.2.2	WaitOrCancel	318
6.277	Arc::Time Class Reference	318
6.277.1	Detailed Description	319
6.277.2	Constructor & Destructor Documentation	319
6.277.2.1	Time	319
6.277.2.2	Time	319
6.277.2.3	Time	319
6.277.2.4	Time	319
6.277.3	Member Function Documentation	320
6.277.3.1	GetFormat	320
6.277.3.2	GetTime	320
6.277.3.3	operator std::string	320
6.277.3.4	operator!=	320
6.277.3.5	operator+	320
6.277.3.6	operator-	320
6.277.3.7	operator-	320
6.277.3.8	operator<	320
6.277.3.9	operator<=	320
6.277.3.10	operator=	320
6.277.3.11	operator=	320
6.277.3.12	operator=	321
6.277.3.13	operator=	321
6.277.3.14	operator==	321
6.277.3.15	operator>	321
6.277.3.16	operator>=	321
6.277.3.17	SetFormat	321
6.277.3.18	SetTime	321

6.277.3.19	SetTime	321
6.277.3.20	str	321
6.278	ArcSec::TimeAttribute Class Reference	321
6.278.1	Detailed Description	322
6.278.2	Member Function Documentation	322
6.278.2.1	encode	322
6.278.2.2	equal	322
6.278.2.3	getId	322
6.278.2.4	getType	322
6.279	Arc::URL Class Reference	323
6.279.1	Member Enumeration Documentation	324
6.279.1.1	Scope	324
6.279.2	Constructor & Destructor Documentation	325
6.279.2.1	URL	325
6.279.2.2	URL	325
6.279.2.3	~URL	325
6.279.3	Member Function Documentation	325
6.279.3.1	AddLDAPAttribute	325
6.279.3.2	AddMetaDataOption	325
6.279.3.3	AddOption	325
6.279.3.4	BaseDN2Path	325
6.279.3.5	ChangeHost	325
6.279.3.6	ChangeLDAPFilter	325
6.279.3.7	ChangeLDAPScope	325
6.279.3.8	ChangePath	326
6.279.3.9	ChangePort	326
6.279.3.10	ChangeProtocol	326
6.279.3.11	CommonLocOption	326
6.279.3.12	CommonLocOptions	326
6.279.3.13	ConnectionURL	326
6.279.3.14	FullPath	326
6.279.3.15	fullstr	326
6.279.3.16	Host	326
6.279.3.17	HTTPOption	327
6.279.3.18	HTTPOptions	327
6.279.3.19	IsSecureProtocol	327

6.279.3.20LDAPAttributes . . . . .	327
6.279.3.21LDAPFilter . . . . .	327
6.279.3.22LDAPScope . . . . .	327
6.279.3.23Locations . . . . .	327
6.279.3.24MetaDataOption . . . . .	327
6.279.3.25MetaDataOptions . . . . .	327
6.279.3.26operator bool . . . . .	328
6.279.3.27operator< . . . . .	328
6.279.3.28operator== . . . . .	328
6.279.3.29Option . . . . .	328
6.279.3.30Options . . . . .	328
6.279.3.31OptionString . . . . .	328
6.279.3.32ParseOptions . . . . .	328
6.279.3.33Passwd . . . . .	328
6.279.3.34Path . . . . .	328
6.279.3.35Path2BaseDN . . . . .	328
6.279.3.36plainstr . . . . .	329
6.279.3.37Port . . . . .	329
6.279.3.38Protocol . . . . .	329
6.279.3.39str . . . . .	329
6.279.3.40Username . . . . .	329
6.279.4 Friends And Related Function Documentation . . . . .	329
6.279.4.1 operator<< . . . . .	329
6.279.5 Field Documentation . . . . .	329
6.279.5.1 commonlocoptions . . . . .	329
6.279.5.2 host . . . . .	329
6.279.5.3 httpoptions . . . . .	329
6.279.5.4 ip6addr . . . . .	329
6.279.5.5 ldapattributes . . . . .	330
6.279.5.6 ldapfilter . . . . .	330
6.279.5.7 ldapscope . . . . .	330
6.279.5.8 locations . . . . .	330
6.279.5.9 metadataoptions . . . . .	330
6.279.5.10passwd . . . . .	330
6.279.5.11lpath . . . . .	330
6.279.5.12port . . . . .	330

6.279.5.13protocol . . . . .	330
6.279.5.14urloptions . . . . .	330
6.279.5.15username . . . . .	330
6.279.5.16valid . . . . .	331
6.280Arc::URLLocation Class Reference . . . . .	331
6.280.1 Detailed Description . . . . .	331
6.280.2 Constructor & Destructor Documentation . . . . .	331
6.280.2.1 URLLocation . . . . .	331
6.280.2.2 URLLocation . . . . .	332
6.280.2.3 URLLocation . . . . .	332
6.280.2.4 URLLocation . . . . .	332
6.280.2.5 URLLocation . . . . .	332
6.280.2.6 ~URLLocation . . . . .	332
6.280.3 Member Function Documentation . . . . .	332
6.280.3.1 fullstr . . . . .	332
6.280.3.2 Name . . . . .	332
6.280.3.3 str . . . . .	332
6.280.4 Field Documentation . . . . .	332
6.280.4.1 name . . . . .	332
6.281Arc::URLMap Class Reference . . . . .	333
6.282Arc::User Class Reference . . . . .	333
6.283Arc::UserConfig Class Reference . . . . .	333
6.283.1 Detailed Description . . . . .	335
6.283.2 Constructor & Destructor Documentation . . . . .	336
6.283.2.1 UserConfig . . . . .	336
6.283.2.2 UserConfig . . . . .	336
6.283.2.3 UserConfig . . . . .	337
6.283.2.4 UserConfig . . . . .	337
6.283.3 Member Function Documentation . . . . .	338
6.283.3.1 AddBartender . . . . .	338
6.283.3.2 AddServices . . . . .	338
6.283.3.3 AddServices . . . . .	338
6.283.3.4 ApplyToConfig . . . . .	339
6.283.3.5 Bartender . . . . .	340
6.283.3.6 Bartender . . . . .	340
6.283.3.7 Broker . . . . .	340

---

6.283.3.8 Broker	341
6.283.3.9 Broker	341
6.283.3.10CACertificatePath	341
6.283.3.11CACertificatePath	342
6.283.3.12CACertificatesDirectory	342
6.283.3.13CACertificatesDirectory	342
6.283.3.14CertificateLifeTime	343
6.283.3.15CertificateLifeTime	343
6.283.3.16CertificatePath	343
6.283.3.17CertificatePath	344
6.283.3.18ClearRejectedServices	344
6.283.3.19ClearRejectedServices	344
6.283.3.20ClearSelectedServices	345
6.283.3.21ClearSelectedServices	345
6.283.3.22CredentialsFound	345
6.283.3.23GetRejectedServices	346
6.283.3.24GetSelectedServices	346
6.283.3.25IdPName	346
6.283.3.26IdPName	347
6.283.3.27InitializeCredentials	347
6.283.3.28JobListFile	348
6.283.3.29JobListFile	348
6.283.3.30KeyPassword	349
6.283.3.31KeyPassword	349
6.283.3.32KeyPath	349
6.283.3.33KeyPath	350
6.283.3.34KeySize	350
6.283.3.35KeySize	351
6.283.3.36LoadConfigurationFile	351
6.283.3.37operator bool	352
6.283.3.38operator!	353
6.283.3.39OverlayFile	353
6.283.3.40OverlayFile	353
6.283.3.41Password	353
6.283.3.42Password	354
6.283.3.43ProxyPath	354

6.283.3.44	ProxyPath	354
6.283.3.45	SaveToFile	355
6.283.3.46	SLCS	355
6.283.3.47	SLCS	355
6.283.3.48	StoreDirectory	355
6.283.3.49	StoreDirectory	356
6.283.3.50	Timeout	356
6.283.3.51	Timeout	356
6.283.3.52	UserName	357
6.283.3.53	UserName	357
6.283.3.54	UtilsDirPath	357
6.283.3.55	UtilsDirPath	358
6.283.3.56	Verbosity	358
6.283.3.57	Verbosity	358
6.283.3.58	VOMSServerPath	358
6.283.3.59	VOMSServerPath	359
6.283.4	Field Documentation	359
6.283.4.1	ARCUSERDIRECTORY	359
6.283.4.2	DEFAULT_BROKER	359
6.283.4.3	DEFAULT_TIMEOUT	360
6.283.4.4	DEFAULTCONFIG	360
6.283.4.5	EXAMPLECONFIG	360
6.283.4.6	SYSCONFIG	360
6.283.4.7	SYSCONFIGARCLOC	360
6.284	Arc::UsernameToken Class Reference	360
6.284.1	Detailed Description	361
6.284.2	Member Enumeration Documentation	361
6.284.2.1	PasswordType	361
6.284.3	Constructor & Destructor Documentation	361
6.284.3.1	UsernameToken	361
6.284.3.2	UsernameToken	361
6.284.3.3	UsernameToken	362
6.284.4	Member Function Documentation	362
6.284.4.1	Authenticate	362
6.284.4.2	Authenticate	362
6.284.4.3	operator bool	362

---

6.284.4.4 Username . . . . .	362
6.285Arc::UserSwitch Class Reference . . . . .	362
6.285.1 Detailed Description . . . . .	362
6.286Arc::VOMSTrustList Class Reference . . . . .	363
6.286.1 Detailed Description . . . . .	363
6.286.2 Constructor & Destructor Documentation . . . . .	363
6.286.2.1 VOMSTrustList . . . . .	363
6.286.2.2 VOMSTrustList . . . . .	364
6.286.3 Member Function Documentation . . . . .	364
6.286.3.1 AddChain . . . . .	364
6.286.3.2 AddChain . . . . .	364
6.286.3.3 AddRegex . . . . .	364
6.287Arc::WSAEndpointReference Class Reference . . . . .	364
6.287.1 Detailed Description . . . . .	365
6.287.2 Constructor & Destructor Documentation . . . . .	365
6.287.2.1 WSAEndpointReference . . . . .	365
6.287.2.2 WSAEndpointReference . . . . .	365
6.287.2.3 WSAEndpointReference . . . . .	365
6.287.2.4 WSAEndpointReference . . . . .	365
6.287.2.5 ~WSAEndpointReference . . . . .	365
6.287.3 Member Function Documentation . . . . .	365
6.287.3.1 Address . . . . .	365
6.287.3.2 Address . . . . .	366
6.287.3.3 MetaData . . . . .	366
6.287.3.4 operator XMLNode . . . . .	366
6.287.3.5 operator= . . . . .	366
6.287.3.6 ReferenceParameters . . . . .	366
6.288Arc::WSAHeader Class Reference . . . . .	366
6.288.1 Detailed Description . . . . .	367
6.288.2 Constructor & Destructor Documentation . . . . .	367
6.288.2.1 WSAHeader . . . . .	367
6.288.2.2 WSAHeader . . . . .	367
6.288.3 Member Function Documentation . . . . .	367
6.288.3.1 Action . . . . .	367
6.288.3.2 Action . . . . .	367
6.288.3.3 Check . . . . .	367

---

6.288.3.4	FaultTo . . . . .	368
6.288.3.5	From . . . . .	368
6.288.3.6	MessageID . . . . .	368
6.288.3.7	MessageID . . . . .	368
6.288.3.8	NewReferenceParameter . . . . .	368
6.288.3.9	operator XmlNode . . . . .	368
6.288.3.10	ReferenceParameter . . . . .	368
6.288.3.11	ReferenceParameter . . . . .	368
6.288.3.12	RelatesTo . . . . .	368
6.288.3.13	RelatesTo . . . . .	368
6.288.3.14	RelationshipType . . . . .	368
6.288.3.15	RelationshipType . . . . .	369
6.288.3.16	ReplyTo . . . . .	369
6.288.3.17	To . . . . .	369
6.288.3.18	To . . . . .	369
6.288.4	Field Documentation . . . . .	369
6.288.4.1	header_allocated_ . . . . .	369
6.289Arc::WSRF	Class Reference . . . . .	369
6.289.1	Detailed Description . . . . .	370
6.289.2	Constructor & Destructor Documentation . . . . .	370
6.289.2.1	WSRF . . . . .	370
6.289.2.2	WSRF . . . . .	371
6.289.3	Member Function Documentation . . . . .	371
6.289.3.1	operator bool . . . . .	371
6.289.3.2	set_namespaces . . . . .	371
6.289.3.3	SOAP . . . . .	371
6.289.4	Field Documentation . . . . .	371
6.289.4.1	allocated_ . . . . .	371
6.289.4.2	valid_ . . . . .	371
6.290Arc::WSRFBaseFault	Class Reference . . . . .	371
6.290.1	Detailed Description . . . . .	372
6.290.2	Constructor & Destructor Documentation . . . . .	372
6.290.2.1	WSRFBaseFault . . . . .	372
6.290.2.2	WSRFBaseFault . . . . .	372
6.290.3	Member Function Documentation . . . . .	372
6.290.3.1	set_namespaces . . . . .	372

6.291Arc::WSRFResourceUnavailableFault Class Reference . . . . .	372
6.292Arc::WSRFResourceUnknownFault Class Reference . . . . .	373
6.293Arc::WSRP Class Reference . . . . .	373
6.293.1 Detailed Description . . . . .	374
6.293.2 Constructor & Destructor Documentation . . . . .	374
6.293.2.1 WSRP . . . . .	374
6.293.2.2 WSRP . . . . .	375
6.293.3 Member Function Documentation . . . . .	375
6.293.3.1 set_namespaces . . . . .	375
6.294Arc::WSRPDeleteResourceProperties Class Reference . . . . .	375
6.295Arc::WSRPDeleteResourcePropertiesRequest Class Reference . . . . .	375
6.296Arc::WSRPDeleteResourcePropertiesRequestFailedFault Class Reference . . . . .	376
6.297Arc::WSRPDeleteResourcePropertiesResponse Class Reference . . . . .	376
6.298Arc::WSRPFault Class Reference . . . . .	376
6.298.1 Detailed Description . . . . .	377
6.298.2 Constructor & Destructor Documentation . . . . .	377
6.298.2.1 WSRPFault . . . . .	377
6.298.2.2 WSRPFault . . . . .	377
6.299Arc::WSRPGetMultipleResourcePropertiesRequest Class Reference . . . . .	377
6.300Arc::WSRPGetMultipleResourcePropertiesResponse Class Reference . . . . .	378
6.301Arc::WSRPGetResourcePropertyDocumentRequest Class Reference . . . . .	378
6.302Arc::WSRPGetResourcePropertyDocumentResponse Class Reference . . . . .	378
6.303Arc::WSRPGetResourcePropertyRequest Class Reference . . . . .	379
6.304Arc::WSRPGetResourcePropertyResponse Class Reference . . . . .	379
6.305Arc::WSRPInsertResourceProperties Class Reference . . . . .	380
6.306Arc::WSRPInsertResourcePropertiesRequest Class Reference . . . . .	380
6.307Arc::WSRPInsertResourcePropertiesRequestFailedFault Class Reference . . . . .	380
6.308Arc::WSRPInsertResourcePropertiesResponse Class Reference . . . . .	381
6.309Arc::WSRPInvalidModificationFault Class Reference . . . . .	381
6.310Arc::WSRPInvalidResourcePropertyQNameFault Class Reference . . . . .	382
6.311Arc::WSRPModifyResourceProperties Class Reference . . . . .	382
6.312Arc::WSRPPutResourcePropertyDocumentRequest Class Reference . . . . .	383
6.313Arc::WSRPPutResourcePropertyDocumentResponse Class Reference . . . . .	383
6.314Arc::WSRPQueryResourcePropertiesRequest Class Reference . . . . .	383
6.315Arc::WSRPQueryResourcePropertiesResponse Class Reference . . . . .	384
6.316Arc::WSRPResourcePropertyChangeFailure Class Reference . . . . .	384

6.316.1 Detailed Description . . . . .	385
6.316.2 Constructor & Destructor Documentation . . . . .	385
6.316.2.1 WSRPResourcePropertyChangeFailure . . . . .	385
6.316.2.2 WSRPResourcePropertyChangeFailure . . . . .	385
6.317Arc::WSRPSetResourcePropertiesRequest Class Reference . . . . .	385
6.318Arc::WSRPSetResourcePropertiesResponse Class Reference . . . . .	385
6.319Arc::WSRPSetResourcePropertyRequestFailedFault Class Reference . . . . .	386
6.320Arc::WSRPUUnableToModifyResourcePropertyFault Class Reference . . . . .	386
6.321Arc::WSRPUUnableToPutResourcePropertyDocumentFault Class Reference . . . . .	387
6.322Arc::WSRPUUpdateResourceProperties Class Reference . . . . .	387
6.323Arc::WSRPUUpdateResourcePropertiesRequest Class Reference . . . . .	387
6.324Arc::WSRPUUpdateResourcePropertiesRequestFailedFault Class Reference . . . . .	388
6.325Arc::WSRPUUpdateResourcePropertiesResponse Class Reference . . . . .	388
6.326ArcSec::X500NameAttribute Class Reference . . . . .	389
6.326.1 Member Function Documentation . . . . .	389
6.326.1.1 encode . . . . .	389
6.326.1.2 equal . . . . .	389
6.326.1.3 getId . . . . .	389
6.326.1.4 getType . . . . .	389
6.327Arc::X509Token Class Reference . . . . .	390
6.327.1 Detailed Description . . . . .	390
6.327.2 Member Enumeration Documentation . . . . .	390
6.327.2.1 X509TokenType . . . . .	390
6.327.3 Constructor & Destructor Documentation . . . . .	390
6.327.3.1 X509Token . . . . .	390
6.327.3.2 X509Token . . . . .	390
6.327.3.3 ~X509Token . . . . .	391
6.327.4 Member Function Documentation . . . . .	391
6.327.4.1 Authenticate . . . . .	391
6.327.4.2 Authenticate . . . . .	391
6.327.4.3 operator bool . . . . .	391
6.328Arc::XmlContainer Class Reference . . . . .	391
6.329Arc::XmlDatabase Class Reference . . . . .	392
6.330Arc::XMLNode Class Reference . . . . .	392
6.330.1 Detailed Description . . . . .	394
6.330.2 Constructor & Destructor Documentation . . . . .	394

---

6.330.2.1 XMLNode . . . . .	394
6.330.2.2 XMLNode . . . . .	394
6.330.2.3 XMLNode . . . . .	394
6.330.2.4 XMLNode . . . . .	394
6.330.2.5 XMLNode . . . . .	395
6.330.2.6 XMLNode . . . . .	395
6.330.2.7 XMLNode . . . . .	395
6.330.2.8 ~XMLNode . . . . .	395
6.330.3 Member Function Documentation . . . . .	395
6.330.3.1 Attribute . . . . .	395
6.330.3.2 Attribute . . . . .	395
6.330.3.3 Attribute . . . . .	395
6.330.3.4 AttributesSize . . . . .	395
6.330.3.5 Child . . . . .	395
6.330.3.6 Destroy . . . . .	396
6.330.3.7 Exchange . . . . .	396
6.330.3.8 FullName . . . . .	396
6.330.3.9 Get . . . . .	396
6.330.3.10GetDoc . . . . .	396
6.330.3.11GetRoot . . . . .	396
6.330.3.12GetXML . . . . .	396
6.330.3.13GetXML . . . . .	396
6.330.3.14Move . . . . .	397
6.330.3.15Name . . . . .	397
6.330.3.16Name . . . . .	397
6.330.3.17Name . . . . .	397
6.330.3.18Namespace . . . . .	397
6.330.3.19NamespacePrefix . . . . .	397
6.330.3.20Namespaces . . . . .	397
6.330.3.21Namespaces . . . . .	397
6.330.3.22New . . . . .	397
6.330.3.23NewAttribute . . . . .	398
6.330.3.24NewAttribute . . . . .	398
6.330.3.25NewChild . . . . .	398
6.330.3.26NewChild . . . . .	398
6.330.3.27NewChild . . . . .	398

6.330.3.28	NewChild	398
6.330.3.29	NewChild	398
6.330.3.30	operator bool	398
6.330.3.31	operator std::string	399
6.330.3.32	operator!	399
6.330.3.33	operator!=	399
6.330.3.34	operator!=	399
6.330.3.35	operator!=	399
6.330.3.36	operator!=	399
6.330.3.37	operator++	399
6.330.3.38	operator--	399
6.330.3.39	operator=	399
6.330.3.40	operator=	399
6.330.3.41	operator=	400
6.330.3.42	operator==	400
6.330.3.43	operator==	400
6.330.3.44	operator==	400
6.330.3.45	operator==	400
6.330.3.46	operator[]	400
6.330.3.47	operator[]	400
6.330.3.48	operator[]	400
6.330.3.49	Parent	401
6.330.3.50	Path	401
6.330.3.51	Prefix	401
6.330.3.52	ReadFromFile	401
6.330.3.53	ReadFromStream	401
6.330.3.54	Replace	401
6.330.3.55	Same	401
6.330.3.56	SaveToFile	401
6.330.3.57	SaveToStream	401
6.330.3.58	Set	401
6.330.3.59	Size	402
6.330.3.60	Swap	402
6.330.3.61	Validate	402
6.330.3.62	XPathLookup	402
6.330.4	Friends And Related Function Documentation	402

6.330.4.1 MatchXMLName . . . . .	402
6.330.4.2 MatchXMLName . . . . .	402
6.330.4.3 MatchXMLName . . . . .	402
6.330.4.4 MatchXMLNamespace . . . . .	402
6.330.4.5 MatchXMLNamespace . . . . .	403
6.330.4.6 MatchXMLNamespace . . . . .	403
6.330.5 Field Documentation . . . . .	403
6.330.5.1 is_owner_ . . . . .	403
6.330.5.2 is_temporary_ . . . . .	403
6.331 Arc::XMLNodeContainer Class Reference . . . . .	403
6.331.1 Detailed Description . . . . .	403
6.331.2 Constructor & Destructor Documentation . . . . .	404
6.331.2.1 XMLNodeContainer . . . . .	404
6.331.2.2 XMLNodeContainer . . . . .	404
6.331.3 Member Function Documentation . . . . .	404
6.331.3.1 Add . . . . .	404
6.331.3.2 Add . . . . .	404
6.331.3.3 AddNew . . . . .	404
6.331.3.4 AddNew . . . . .	404
6.331.3.5 Nodes . . . . .	404
6.331.3.6 operator= . . . . .	404
6.331.3.7 operator[] . . . . .	404
6.331.3.8 Size . . . . .	404
6.332 Arc::XMLSecNode Class Reference . . . . .	405
6.332.1 Detailed Description . . . . .	405
6.332.2 Constructor & Destructor Documentation . . . . .	405
6.332.2.1 XMLSecNode . . . . .	405
6.332.3 Member Function Documentation . . . . .	405
6.332.3.1 AddSignatureTemplate . . . . .	405
6.332.3.2 DecryptNode . . . . .	406
6.332.3.3 EncryptNode . . . . .	406
6.332.3.4 SignNode . . . . .	406
6.332.3.5 VerifyNode . . . . .	406
6.333 Arc::XRSLParser Class Reference . . . . .	407
<b>7 File Documentation</b>	<b>409</b>
7.1 URL.h File Reference . . . . .	409

7.1.1	Detailed Description . . . . .	410
7.1.2	Define Documentation . . . . .	410
7.1.2.1	RC_DEFAULT_PORT . . . . .	410

# Chapter 1

## Namespace Index

### 1.1 Namespace List

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

<b>Arc</b>	(Some utility methods for using xml security library ( <a href="http://www.aleksey.com/xmlsec/">http://www.aleksey.com/xmlsec/</a> )) . . . . .	23
<b>ArcCredential</b>	. . . . .	43



## Chapter 2

# Data Structure Index

### 2.1 Class Hierarchy

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

ArcCredential::ACACI . . . . .	45
ArcCredential::ACATTHOLDER . . . . .	45
ArcCredential::ACATTR . . . . .	45
ArcCredential::ACATTRIBUTE . . . . .	45
ArcCredential::ACC . . . . .	45
ArcCredential::ACCERTS . . . . .	46
ArcCredential::ACDIGEST . . . . .	46
ArcCredential::ACFORM . . . . .	46
ArcCredential::ACFULLATTRIBUTES . . . . .	46
ArcCredential::ACHOLDER . . . . .	46
ArcCredential::ACIETFATTR . . . . .	46
ArcCredential::ACINFO . . . . .	46
ArcCredential::ACIS . . . . .	47
ArcCredential::ACSEQ . . . . .	47
ArcCredential::ACTARGET . . . . .	47
ArcCredential::ACTARGETS . . . . .	47
ArcCredential::ACVAL . . . . .	47
Arc::ApplicationType . . . . .	50
Arc::ArcLocation . . . . .	51
ArcSec::ArcPeriod . . . . .	51
ArcSec::Attr . . . . .	52
Arc::AttributeIterator . . . . .	53
ArcSec::AttributeProxy . . . . .	55
ArcSec::AttributeValue . . . . .	56
ArcSec::AnyURIAttribute . . . . .	49
ArcSec::BooleanAttribute . . . . .	61
ArcSec::DateAttribute . . . . .	134
ArcSec::DateTimeAttribute . . . . .	135
ArcSec::DurationAttribute . . . . .	147
ArcSec::GenericAttribute . . . . .	167
ArcSec::PeriodAttribute . . . . .	245
ArcSec::StringAttribute . . . . .	307
ArcSec::TimeAttribute . . . . .	321

ArcSec::X500NameAttribute . . . . .	389
ArcSec::Attr . . . . .	58
ArcSec::AuthzRequest . . . . .	59
ArcSec::AuthzRequestSection . . . . .	59
Arc::AutoPointer< T > . . . . .	59
Arc::Base64 . . . . .	60
Arc::BaseConfig . . . . .	60
Arc::MCCConfig . . . . .	206
Arc::ByteArray . . . . .	65
Arc::CacheParameters . . . . .	66
ArcCredential::cert_verify_context . . . . .	66
Arc::ChainContext . . . . .	66
Arc::Checksum . . . . .	67
Arc::Adler32Sum . . . . .	47
Arc::ChecksumAny . . . . .	67
Arc::CRC32Sum . . . . .	89
Arc::MD5Sum . . . . .	209
Arc::ClientHTTPwithSAML2SSO . . . . .	70
Arc::ClientInterface . . . . .	71
Arc::ClientTCP . . . . .	73
Arc::ClientHTTP . . . . .	69
Arc::ClientSOAP . . . . .	71
Arc::ClientSOAPwithSAML2SSO . . . . .	73
Arc::ClientX509Delegation . . . . .	74
ArcSec::CombiningAlg . . . . .	75
ArcSec::DenyOverridesCombiningAlg . . . . .	144
ArcSec::OrderedCombiningAlg . . . . .	228
ArcSec::PermitOverridesCombiningAlg . . . . .	246
Arc::ConfusaCertHandler . . . . .	78
Arc::ConfusaParserUtils . . . . .	79
Arc::CountedPointer< T > . . . . .	80
Arc::Counter . . . . .	81
Arc::IntraProcessCounter . . . . .	179
Arc::CounterTicket . . . . .	88
Arc::Credential . . . . .	90
Arc::CredentialError . . . . .	97
Arc::CredentialStore . . . . .	98
Arc::Database . . . . .	98
Arc::MySQLDatabase . . . . .	221
Arc::DataBuffer . . . . .	100
Arc::DataCallback . . . . .	105
Arc::DataHandle . . . . .	106
Arc::DataMover . . . . .	106
Arc::DataSourceType . . . . .	129
Arc::DataSpeed . . . . .	129
Arc::DataStagingType . . . . .	132
Arc::DataStatus . . . . .	132
Arc::DataTargetType . . . . .	134
Arc::DataType . . . . .	134
Arc::DirectoryType . . . . .	145
Arc::FileType . . . . .	165

Arc::DBranch . . . . .	136
Arc::DelegationConsumer . . . . .	136
Arc::DelegationConsumerSOAP . . . . .	138
Arc::DelegationContainerSOAP . . . . .	139
Arc::DelegationProvider . . . . .	141
Arc::DelegationProviderSOAP . . . . .	142
Arc::DiskSpaceRequirementType . . . . .	145
Arc::DItem . . . . .	146
Arc::DItemString . . . . .	146
ArcSec::EvalResult . . . . .	149
ArcSec::EvaluationCtx . . . . .	149
ArcSec::EvaluatorContext . . . . .	152
ArcSec::EvaluatorLoader . . . . .	153
Arc::ExecutableType . . . . .	154
Arc::ExecutionTarget . . . . .	154
Arc::ExpirationReminder . . . . .	158
Arc::FileCache . . . . .	159
FileCacheHash . . . . .	164
Arc::FileInfo . . . . .	164
Arc::FileLock . . . . .	164
Arc::FinderLoader . . . . .	165
ArcSec::Function . . . . .	166
ArcSec::EqualFunction . . . . .	148
ArcSec::InRangeFunction . . . . .	179
ArcSec::MatchFunction . . . . .	201
Arc::GlobusResult . . . . .	168
Arc::GSSCredential . . . . .	168
Arc::HTTPClientInfo . . . . .	169
Arc::InfoCache . . . . .	169
Arc::InfoFilter . . . . .	170
Arc::InfoRegister . . . . .	171
Arc::InfoRegisterContainer . . . . .	172
Arc::InfoRegisters . . . . .	173
Arc::InfoRegistrar . . . . .	173
Arc::InformationInterface . . . . .	175
Arc::InfoCacheInterface . . . . .	170
Arc::InformationContainer . . . . .	174
Arc::InformationRequest . . . . .	176
Arc::InformationResponse . . . . .	178
Arc::initializeCredentialsType . . . . .	179
Arc::ISIS_description . . . . .	184
Arc::IString . . . . .	184
Arc::Job . . . . .	184
Arc::JobDescription . . . . .	188
Arc::JobDescriptionParser . . . . .	189
Arc::ARCJSDLParser . . . . .	50
Arc::JDLParser . . . . .	184
Arc::XRSLParser . . . . .	407
Arc::JobIdentificationType . . . . .	189
Arc::JobMetaType . . . . .	189
Arc::JobState . . . . .	189
Arc::JobSupervisor . . . . .	190

Arc::LoadableModuleDescription . . . . .	190
Arc::Loader . . . . .	191
Arc::BrokerLoader . . . . .	64
Arc::DataPointLoader . . . . .	128
Arc::JobControllerLoader . . . . .	187
Arc::MCCLoader . . . . .	208
Arc::SubmitterLoader . . . . .	309
Arc::TargetRetrieverLoader . . . . .	315
Arc::LogDestination . . . . .	192
Arc::LogFile . . . . .	192
Arc::LogStream . . . . .	199
Arc::Logger . . . . .	195
Arc::LoggerFormat . . . . .	197
Arc::LogMessage . . . . .	197
Arc::MCC_Status . . . . .	204
Arc::MemoryAllocationException . . . . .	210
Arc::Message . . . . .	210
Arc::MessageAttributes . . . . .	212
Arc::MessageAuth . . . . .	215
Arc::MessageAuthContext . . . . .	216
Arc::MessageContext . . . . .	217
Arc::MessageContextElement . . . . .	217
ArcSec::PDPCConfigContext . . . . .	242
Arc::MessagePayload . . . . .	218
Arc::PayloadRawInterface . . . . .	232
Arc::PayloadRaw . . . . .	229
Arc::PayloadSOAP . . . . .	234
Arc::PayloadStreamInterface . . . . .	238
Arc::PayloadStream . . . . .	235
Arc::PayloadWSRF . . . . .	241
Arc::ModuleDesc . . . . .	218
Arc::ModuleManager . . . . .	218
Arc::PluginsFactory . . . . .	252
Arc::ClassLoader . . . . .	69
Arc::NotificationType . . . . .	225
Arc::NS . . . . .	225
Arc::OptionParser . . . . .	227
passwd . . . . .	228
Arc::PathIterator . . . . .	228
Arc::PayloadRawBuf . . . . .	231
Arc::Period . . . . .	243
Arc::PlexerEntry . . . . .	249
Arc::Plugin . . . . .	249
Arc::Broker . . . . .	62
Arc::DataPoint . . . . .	108
Arc::DataPointDirect . . . . .	118
Arc::DataPointIndex . . . . .	123
Arc::JobController . . . . .	185
Arc::MCCInterface . . . . .	207
Arc::MCC . . . . .	202
Arc::Plexer . . . . .	247

Test::TestMCC . . . . .	316
Test::TestMCC . . . . .	316
Arc::Service . . . . .	285
Arc::RegisteredService . . . . .	262
Test::TestService . . . . .	317
Arc::Submitter . . . . .	308
Arc::TargetRetriever . . . . .	313
ArcSec::AlgFactory . . . . .	48
ArcSec::AttributeFactory . . . . .	52
ArcSec::Evaluator . . . . .	150
ArcSec::FnFactory . . . . .	165
ArcSec::PDP . . . . .	242
ArcSec::Policy . . . . .	254
ArcSec::Request . . . . .	263
ArcSec::SecHandler . . . . .	283
Arc::PluginArgument . . . . .	250
Arc::BrokerPluginArgument . . . . .	65
Arc::ClassLoaderPluginArgument . . . . .	69
Arc::DataPointPluginArgument . . . . .	129
Arc::JobControllerPluginArgument . . . . .	188
Arc::MCCPluginArgument . . . . .	209
Arc::ServicePluginArgument . . . . .	287
Arc::SubmitterPluginArgument . . . . .	310
Arc::TargetRetrieverPluginArgument . . . . .	316
ArcSec::PDPPluginArgument . . . . .	243
ArcSec::SecHandlerPluginArgument . . . . .	284
Arc::PluginDesc . . . . .	252
Arc::PluginDescriptor . . . . .	252
ArcSec::PolicyStore::PolicyElement . . . . .	256
ArcSec::PolicyParser . . . . .	256
ArcSec::PolicyStore . . . . .	257
Arc::PrintFBase . . . . .	258
Arc::PrintF< T0, T1, T2, T3, T4, T5, T6, T7 > . . . . .	258
ArcCredential::PROXYCERTINFO_st . . . . .	259
ArcCredential::PROXYPOLICY_st . . . . .	259
Arc::Query . . . . .	259
Arc::MySQLQuery . . . . .	223
Arc::Range< T > . . . . .	261
Arc::Register_Info_Type . . . . .	261
Arc::RegularExpression . . . . .	262
ArcSec::RequestAttribute . . . . .	265
ArcSec::RequestItem . . . . .	266
ArcSec::RequestTuple . . . . .	266
Arc::ResourceSlotType . . . . .	266
Arc::ResourcesType . . . . .	266
Arc::ResourceTargetType . . . . .	267
ArcSec::Response . . . . .	267
ArcSec::ResponseItem . . . . .	267
ArcSec::ResponseList . . . . .	267
Arc::RSL . . . . .	267
Arc::RSLBoolean . . . . .	268
Arc::RSLCondition . . . . .	268

Arc::RSLParser . . . . .	269
Arc::RSLValue . . . . .	270
Arc::RSLConcat . . . . .	268
Arc::RSLList . . . . .	269
Arc::RSLLiteral . . . . .	269
Arc::RSLSequence . . . . .	270
Arc::RSLVariable . . . . .	270
Arc::Run . . . . .	270
Arc::SAML2LoginClient . . . . .	274
Arc::OAuthConsumer . . . . .	225
Arc::SAML2SSOHTTPClient . . . . .	275
Arc::HakaClient . . . . .	168
Arc::OpenIdpClient . . . . .	226
Arc::SAMLToken . . . . .	277
Arc::ScalableTime< T > . . . . .	279
Arc::ScalableTime< int > . . . . .	280
Arc::SecAttr . . . . .	280
Arc::MultiSecAttr . . . . .	220
Arc::SecAttrFormat . . . . .	281
Arc::SecAttrValue . . . . .	282
Arc::CISStringValue . . . . .	67
ArcSec::Security . . . . .	284
Arc::SimpleCondition . . . . .	287
Arc::SimpleCounter . . . . .	288
Arc::SOAPMessage . . . . .	289
Arc::Software . . . . .	290
Arc::ApplicationEnvironment . . . . .	50
Arc::SoftwareRequirement . . . . .	297
ArcSec::Source . . . . .	305
ArcSec::SourceFile . . . . .	306
ArcSec::SourceURL . . . . .	306
Arc::TargetGenerator . . . . .	310
Arc::ThreadInitializer . . . . .	318
Arc::ThreadRegistry . . . . .	318
Arc::Time . . . . .	318
Arc::URL . . . . .	323
Arc::URLLocation . . . . .	331
Arc::URLMap . . . . .	333
Arc::User . . . . .	333
Arc::UserConfig . . . . .	333
Arc::UsernameToken . . . . .	360
Arc::UserSwitch . . . . .	362
Arc::VOMSTrustList . . . . .	363
Arc::WSAEndpointReference . . . . .	364
Arc::WSAHeader . . . . .	366
Arc::WSRF . . . . .	369
Arc::WSRFBaseFault . . . . .	371
Arc::WSRFResourceUnavailableFault . . . . .	372
Arc::WSRFResourceUnknownFault . . . . .	373
Arc::WSRPFault . . . . .	376
Arc::WSRPInvalidResourcePropertyQNameFault . . . . .	382

Arc::WSRPResourcePropertyChangeFailure . . . . .	384
Arc::WSRPDeleteResourcePropertiesRequestFailedFault . . . . .	376
Arc::WSRPInsertResourcePropertiesRequestFailedFault . . . . .	380
Arc::WSRPInvalidModificationFault . . . . .	381
Arc::WSRPSetResourcePropertyRequestFailedFault . . . . .	386
Arc::WSRPUnableToModifyResourcePropertyFault . . . . .	386
Arc::WSRPUnableToPutResourcePropertyDocumentFault . . . . .	387
Arc::WSRPUpdateResourcePropertiesRequestFailedFault . . . . .	388
Arc::WSRP . . . . .	373
Arc::WSRPDeleteResourcePropertiesRequest . . . . .	375
Arc::WSRPDeleteResourcePropertiesResponse . . . . .	376
Arc::WSRPGetMultipleResourcePropertiesRequest . . . . .	377
Arc::WSRPGetMultipleResourcePropertiesResponse . . . . .	378
Arc::WSRPGetResourcePropertyDocumentRequest . . . . .	378
Arc::WSRPGetResourcePropertyDocumentResponse . . . . .	378
Arc::WSRPGetResourcePropertyRequest . . . . .	379
Arc::WSRPGetResourcePropertyResponse . . . . .	379
Arc::WSRPInsertResourcePropertiesRequest . . . . .	380
Arc::WSRPInsertResourcePropertiesResponse . . . . .	381
Arc::WSRPPutResourcePropertyDocumentRequest . . . . .	383
Arc::WSRPPutResourcePropertyDocumentResponse . . . . .	383
Arc::WSRPQueryResourcePropertiesRequest . . . . .	383
Arc::WSRPQueryResourcePropertiesResponse . . . . .	384
Arc::WSRPSetResourcePropertiesRequest . . . . .	385
Arc::WSRPSetResourcePropertiesResponse . . . . .	385
Arc::WSRPUpdateResourcePropertiesRequest . . . . .	387
Arc::WSRPUpdateResourcePropertiesResponse . . . . .	388
Arc::WSRPModifyResourceProperties . . . . .	382
Arc::WSRPDeleteResourceProperties . . . . .	375
Arc::WSRPInsertResourceProperties . . . . .	380
Arc::WSRPUpdateResourceProperties . . . . .	387
Arc::X509Token . . . . .	390
Arc::XmlContainer . . . . .	391
Arc::XmlDatabase . . . . .	392
Arc::XMLNode . . . . .	392
Arc::Config . . . . .	76
Arc::IniConfig . . . . .	178
Arc::Profile . . . . .	258
Arc::SecHandlerConfig . . . . .	283
Arc::ARCPolicyHandlerConfig . . . . .	51
Arc::DNListHandlerConfig . . . . .	146
Arc::XMLSecNode . . . . .	405
ArcSec::SecHandlerConfig . . . . .	284
Arc::XMLNodeContainer . . . . .	403



# Chapter 3

## Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<b>ArcCredential::ACACI</b> . . . . .	45
<b>ArcCredential::ACATTHOLDER</b> . . . . .	45
<b>ArcCredential::ACATTR</b> . . . . .	45
<b>ArcCredential::ACATTRIBUTE</b> . . . . .	45
<b>ArcCredential::ACC</b> . . . . .	45
<b>ArcCredential::ACCERTS</b> . . . . .	46
<b>ArcCredential::ACDIGEST</b> . . . . .	46
<b>ArcCredential::ACFORM</b> . . . . .	46
<b>ArcCredential::ACFULLATTRIBUTES</b> . . . . .	46
<b>ArcCredential::ACHOLDER</b> . . . . .	46
<b>ArcCredential::ACIETFATTR</b> . . . . .	46
<b>ArcCredential::ACINFO</b> . . . . .	46
<b>ArcCredential::ACIS</b> . . . . .	47
<b>ArcCredential::ACSEQ</b> . . . . .	47
<b>ArcCredential::ACTARGET</b> . . . . .	47
<b>ArcCredential::ACTARGETS</b> . . . . .	47
<b>ArcCredential::ACVAL</b> . . . . .	47
<b>Arc::Adler32Sum</b> (Implementation of Adler32 checksum) . . . . .	47
<b>ArcSec::AlgFactory</b> (Interface for algorithm factory class) . . . . .	48
<b>ArcSec::AnyURIAttribute</b> . . . . .	49
<b>Arc::ApplicationEnvironment</b> ( <b>ApplicationEnvironment</b> (p. 50)) . . . . .	50
<b>Arc::ApplicationType</b> . . . . .	50
<b>Arc::ARCJSDLParser</b> . . . . .	50
<b>Arc::ArcLocation</b> (Determines ARC installation location) . . . . .	51
<b>ArcSec::ArcPeriod</b> . . . . .	51
<b>Arc::ARCPolicyHandlerConfig</b> . . . . .	51
<b>ArcSec::Attr</b> ( <b>Attr</b> (p. 52) contains a tuple of attribute type and value) . . . . .	52
<b>ArcSec::AttributeFactory</b> . . . . .	52
<b>Arc::AttributeIterator</b> (A const iterator class for accessing multiple values of an attribute) . . . . .	53
<b>ArcSec::AttributeProxy</b> (Interface for creating the <b>AttributeValue</b> (p. 56) object, it will be used by <b>AttributeFactory</b> (p. 52)) . . . . .	55
<b>ArcSec::AttributeValue</b> (Interface for containing different type of <Attribute> node for both policy and request) . . . . .	56

<b>ArcSec::Attrs</b> ( <b>Attrs</b> (p. 58) is a container for one or more <b>Attr</b> (p. 52) ) . . . . .	58
<b>ArcSec::AuthzRequest</b> . . . . .	59
<b>ArcSec::AuthzRequestSection</b> . . . . .	59
<b>Arc::AutoPointer</b> < <b>T</b> > (Wrapper for pointer with automatic destruction ) . . . . .	59
<b>Arc::Base64</b> . . . . .	60
<b>Arc::BaseConfig</b> . . . . .	60
<b>ArcSec::BooleanAttribute</b> . . . . .	61
<b>Arc::Broker</b> . . . . .	62
<b>Arc::BrokerLoader</b> . . . . .	64
<b>Arc::BrokerPluginArgument</b> . . . . .	65
<b>Arc::ByteArray</b> . . . . .	65
<b>Arc::CacheParameters</b> . . . . .	66
<b>ArcCredential::cert_verify_context</b> . . . . .	66
<b>Arc::ChainContext</b> (Interface to chain specific functionality ) . . . . .	66
<b>Arc::Checksum</b> (Defines interface for variuos checksum manipulations ) . . . . .	67
<b>Arc::ChecksumAny</b> (Wraper for <b>Checksum</b> (p. 67) class ) . . . . .	67
<b>Arc::CIStrngValue</b> (This class implements case insensitive strings as security attributes ) . . . . .	67
<b>Arc::ClassLoader</b> . . . . .	69
<b>Arc::ClassLoaderPluginArgument</b> . . . . .	69
<b>Arc::ClientHTTP</b> (Class for setting up a <b>MCC</b> (p. 202) chain for HTTP communication ) . . . . .	69
<b>Arc::ClientHTTPwithSAML2SSO</b> . . . . .	70
<b>Arc::ClientInterface</b> (Utility base class for <b>MCC</b> (p. 202) ) . . . . .	71
<b>Arc::ClientSOAP</b> . . . . .	71
<b>Arc::ClientSOAPwithSAML2SSO</b> . . . . .	73
<b>Arc::ClientTCP</b> (Class for setting up a <b>MCC</b> (p. 202) chain for TCP communication ) . . . . .	73
<b>Arc::ClientX509Delegation</b> . . . . .	74
<b>ArcSec::CombiningAlg</b> (Interface for combining algorithm ) . . . . .	75
<b>Arc::Config</b> (Configuration element - represents (sub)tree of ARC configuration ) . . . . .	76
<b>Arc::ConfusaCertHandler</b> . . . . .	78
<b>Arc::ConfusaParserUtils</b> . . . . .	79
<b>Arc::CountedPointer</b> < <b>T</b> > (Wrapper for pointer with automatic destruction and mutiple references ) . . . . .	80
<b>Arc::Counter</b> (A class defining a common interface for counters ) . . . . .	81
<b>Arc::CounterTicket</b> (A class for "tickets" that correspond to counter reservations ) . . . . .	88
<b>Arc::CRC32Sum</b> (Implementation of CRC32 checksum ) . . . . .	89
<b>Arc::Credential</b> . . . . .	90
<b>Arc::CredentialError</b> . . . . .	97
<b>Arc::CredentialStore</b> . . . . .	98
<b>Arc::Database</b> (Interface for calling database client library ) . . . . .	98
<b>Arc::DataBuffer</b> (Represents set of buffers ) . . . . .	100
<b>Arc::DataCallback</b> . . . . .	105
<b>Arc::DataHandle</b> (This class is a wrapper around the <b>DataPoint</b> (p. 108) class ) . . . . .	106
<b>Arc::DataMover</b> . . . . .	106
<b>Arc::DataPoint</b> (This base class is an abstraction of <b>URL</b> (p. 323) ) . . . . .	108
<b>Arc::DataPointDirect</b> (This is a kind of generalized file handle ) . . . . .	118
<b>Arc::DataPointIndex</b> (Complements <b>DataPoint</b> (p. 108) with attributes common for Indexing Service (p. 285) URLs ) . . . . .	123
<b>Arc::DataPointLoader</b> . . . . .	128
<b>Arc::DataPointPluginArgument</b> . . . . .	129
<b>Arc::DataSourceType</b> . . . . .	129
<b>Arc::DataSpeed</b> (Keeps track of average and instantaneous transfer speed ) . . . . .	129
<b>Arc::DataStagingType</b> . . . . .	132
<b>Arc::DataStatus</b> . . . . .	132
<b>Arc::DataTargetType</b> . . . . .	134

<b>Arc::DataType</b> . . . . .	134
<b>ArcSec::DateAttribute</b> . . . . .	134
<b>ArcSec::DateTimeAttribute</b> . . . . .	135
<b>Arc::DBranch</b> . . . . .	136
<b>Arc::DelegationConsumer</b> . . . . .	136
<b>Arc::DelegationConsumerSOAP</b> . . . . .	138
<b>Arc::DelegationContainerSOAP</b> . . . . .	139
<b>Arc::DelegationProvider</b> . . . . .	141
<b>Arc::DelegationProviderSOAP</b> . . . . .	142
<b>ArcSec::DenyOverridesCombiningAlg</b> (Implement the "Deny-Overrides" algorithm) . . . . .	144
<b>Arc::DirectoryType</b> . . . . .	145
<b>Arc::DiskSpaceRequirementType</b> . . . . .	145
<b>Arc::DItem</b> . . . . .	146
<b>Arc::DItemString</b> . . . . .	146
<b>Arc::DNListHandlerConfig</b> . . . . .	146
<b>ArcSec::DurationAttribute</b> . . . . .	147
<b>ArcSec::EqualFunction</b> (Evaluate whether the two values are equal) . . . . .	148
<b>ArcSec::EvalResult</b> (Struct to record the xml node and effect, which will be used by <b>Evaluator</b> (p. 150) to get the information about which rule/policy(in xmlnode) is satisfied) . . . . .	149
<b>ArcSec::EvaluationCtx</b> ( <b>EvaluationCtx</b> (p. 149), in charge of storing some context information for) . . . . .	149
<b>ArcSec::Evaluator</b> (Interface for policy evaluation. Execute the policy evaluation, based on the request and policy) . . . . .	150
<b>ArcSec::EvaluatorContext</b> (Context for evaluator. It includes the factories which will be used to create related objects) . . . . .	152
<b>ArcSec::EvaluatorLoader</b> ( <b>EvaluatorLoader</b> (p. 153) is implemented as a helper class for loading different <b>Evaluator</b> (p. 150) objects, like <b>ArcEvaluator</b> ) . . . . .	153
<b>Arc::ExecutableType</b> . . . . .	154
<b>Arc::ExecutionTarget</b> ( <b>ExecutionTarget</b> (p. 154)) . . . . .	154
<b>Arc::ExpirationReminder</b> (A class intended for internal use within counters) . . . . .	158
<b>Arc::FileCache</b> . . . . .	159
<b>FileCacheHash</b> . . . . .	164
<b>Arc::FileInfo</b> ( <b>FileInfo</b> (p. 164) stores information about files (metadata)) . . . . .	164
<b>Arc::FileLock</b> (A general file locking class) . . . . .	164
<b>Arc::FileType</b> . . . . .	165
<b>Arc::FinderLoader</b> . . . . .	165
<b>ArcSec::FnFactory</b> (Interface for function factory class) . . . . .	165
<b>ArcSec::Function</b> (Interface for function, which is in charge of evaluating two <b>AttributeValue</b> (p. 56)) . . . . .	166
<b>ArcSec::GenericAttribute</b> . . . . .	167
<b>Arc::GlobusResult</b> . . . . .	168
<b>Arc::GSSCredential</b> . . . . .	168
<b>Arc::HakaClient</b> . . . . .	168
<b>Arc::HTTPClientInfo</b> . . . . .	169
<b>Arc::InfoCache</b> (Stores XML document in filesystem split into parts) . . . . .	169
<b>Arc::InfoCacheInterface</b> . . . . .	170
<b>Arc::InfoFilter</b> (Filters information document according to identity of requestor) . . . . .	170
<b>Arc::InfoRegister</b> (Registration to ISIS interface) . . . . .	171
<b>Arc::InfoRegisterContainer</b> . . . . .	172
<b>Arc::InfoRegisters</b> (Handling multiple registrations to ISISes) . . . . .	173
<b>Arc::InfoRegistrar</b> (Registration process associated with particular ISIS) . . . . .	173
<b>Arc::InformationContainer</b> (Information System document container and processor) . . . . .	174
<b>Arc::InformationInterface</b> (Information System message processor) . . . . .	175
<b>Arc::InformationRequest</b> (Request for information in InfoSystem) . . . . .	176

<b>Arc::InformationResponse</b> (Informational response from InfoSystem ) . . . . .	178
<b>Arc::IniConfig</b> . . . . .	178
<b>Arc::initializeCredentialsType</b> . . . . .	179
<b>ArcSec::InRangeFunction</b> . . . . .	179
<b>Arc::IntraProcessCounter</b> (A class for counters used by threads within a single process ) . . .	179
<b>Arc::ISIS_description</b> . . . . .	184
<b>Arc::IString</b> . . . . .	184
<b>Arc::JDLParser</b> . . . . .	184
<b>Arc::Job</b> ( <b>Job</b> (p. 184) ) . . . . .	184
<b>Arc::JobController</b> (Base class for the JobControllers ) . . . . .	185
<b>Arc::JobControllerLoader</b> . . . . .	187
<b>Arc::JobControllerPluginArgument</b> . . . . .	188
<b>Arc::JobDescription</b> . . . . .	188
<b>Arc::JobDescriptionParser</b> . . . . .	189
<b>Arc::JobIdentificationType</b> . . . . .	189
<b>Arc::JobMetaType</b> . . . . .	189
<b>Arc::JobState</b> . . . . .	189
<b>Arc::JobSupervisor</b> (% <b>JobSupervisor</b> (p. 190) class ) . . . . .	190
<b>Arc::LoadableModuleDescription</b> . . . . .	190
<b>Arc::Loader</b> (Plugins loader ) . . . . .	191
<b>Arc::LogDestination</b> (A base class for log destinations ) . . . . .	192
<b>Arc::LogFile</b> (A class for logging to files ) . . . . .	192
<b>Arc::Logger</b> (A logger class ) . . . . .	195
<b>Arc::LoggerFormat</b> . . . . .	197
<b>Arc::LogMessage</b> (A class for log messages ) . . . . .	197
<b>Arc::LogStream</b> (A class for logging to ostreams ) . . . . .	199
<b>ArcSec::MatchFunction</b> (Evaluate whether arg1 (value in regular expression) matched arg0 (table in regular expression) ) . . . . .	201
<b>Arc::MCC</b> ( <b>Message</b> (p. 210) Chain Component - base class for every <b>MCC</b> (p. 202) plugin ) . . . . .	202
<b>Arc::MCC_Status</b> (A class for communication of <b>MCC</b> (p. 202) processing results ) . . . . .	204
<b>Arc::MCCConfig</b> . . . . .	206
<b>Arc::MCCInterface</b> (Interface for communication between <b>MCC</b> (p. 202), <b>Service</b> (p. 285) and <b>Plexer</b> (p. 247) objects ) . . . . .	207
<b>Arc::MCCLoader</b> (Creator of <b>Message</b> (p. 210) Component Chains ( <b>MCC</b> (p. 202))) . . . . .	208
<b>Arc::MCCPluginArgument</b> . . . . .	209
<b>Arc::MD5Sum</b> (Implementation of MD5 checksum ) . . . . .	209
<b>Arc::MemoryAllocationException</b> . . . . .	210
<b>Arc::Message</b> (Object being passed through chain of MCCs ) . . . . .	210
<b>Arc::MessageAttributes</b> (A class for storage of attribute values ) . . . . .	212
<b>Arc::MessageAuth</b> (Contains authenticity information, authorization tokens and decisions ) . . . . .	215
<b>Arc::MessageAuthContext</b> (Handler for content of message auth* context ) . . . . .	216
<b>Arc::MessageContext</b> (Handler for content of message context ) . . . . .	217
<b>Arc::MessageContextElement</b> (Top class for elements contained in message context ) . . . . .	217
<b>Arc::MessagePayload</b> (Base class for content of message passed through chain ) . . . . .	218
<b>Arc::ModuleDesc</b> (Description of loadable module ) . . . . .	218
<b>Arc::ModuleManager</b> (Manager of shared libraries ) . . . . .	218
<b>Arc::MultiSecAttr</b> (Container of multiple <b>SecAttr</b> (p. 280) attributes ) . . . . .	220
<b>Arc::MySQLDatabase</b> . . . . .	221
<b>Arc::MySQLQuery</b> . . . . .	223
<b>Arc::NotificationType</b> . . . . .	225
<b>Arc::NS</b> . . . . .	225
<b>Arc::OAuthConsumer</b> . . . . .	225
<b>Arc::OpenIdpClient</b> . . . . .	226
<b>Arc::OptionParser</b> . . . . .	227

<b>ArcSec::OrderedCombiningAlg</b> . . . . .	228
<b>passwd</b> . . . . .	228
<b>Arc::PathIterator</b> (Class to iterate through elements of path ) . . . . .	228
<b>Arc::PayloadRaw</b> (Raw byte multi-buffer ) . . . . .	229
<b>Arc::PayloadRawBuf</b> . . . . .	231
<b>Arc::PayloadRawInterface</b> (Random Access Payload for <b>Message</b> (p. 210) objects ) . . . . .	232
<b>Arc::PayloadSOAP</b> (Payload of <b>Message</b> (p. 210) with SOAP content ) . . . . .	234
<b>Arc::PayloadStream</b> (POSIX handle as Payload ) . . . . .	235
<b>Arc::PayloadStreamInterface</b> (Stream-like Payload for <b>Message</b> (p. 210) object ) . . . . .	238
<b>Arc::PayloadWSRF</b> (This class combines <b>MessagePayload</b> (p. 218) with <b>WSRF</b> (p. 369) ) . . . . .	241
<b>ArcSec::PDP</b> (Base class for <b>Policy</b> (p. 254) Decision Point plugins ) . . . . .	242
<b>ArcSec::PDPConfigContext</b> . . . . .	242
<b>ArcSec::PDPPluginArgument</b> . . . . .	243
<b>Arc::Period</b> . . . . .	243
<b>ArcSec::PeriodAttribute</b> . . . . .	245
<b>ArcSec::PermitOverridesCombiningAlg</b> (Implement the "Permit-Overrides" algorithm ) . . . . .	246
<b>Arc::Plexer</b> (The <b>Plexer</b> (p. 247) class, used for routing messages to services ) . . . . .	247
<b>Arc::PlexerEntry</b> (A pair of label (regex) and pointer to <b>MCC</b> (p. 202) ) . . . . .	249
<b>Arc::Plugin</b> (Base class for loadable ARC components ) . . . . .	249
<b>Arc::PluginArgument</b> (Base class for passing arguments to loadable ARC components ) . . . . .	250
<b>Arc::PluginDesc</b> (Description of plugin ) . . . . .	252
<b>Arc::PluginDescriptor</b> (Description of ARC loadable component ) . . . . .	252
<b>Arc::PluginsFactory</b> (Generic ARC plugins loader ) . . . . .	252
<b>ArcSec::Policy</b> (Interface for containing and processing different types of policy ) . . . . .	254
<b>ArcSec::PolicyStore::PolicyElement</b> . . . . .	256
<b>ArcSec::PolicyParser</b> (A interface which will isolate the policy object from actual policy storage (files, urls, database) ) . . . . .	256
<b>ArcSec::PolicyStore</b> (Storage place for policy objects ) . . . . .	257
<b>Arc::Printf&lt; T0, T1, T2, T3, T4, T5, T6, T7 &gt;</b> . . . . .	258
<b>Arc::PrintfBase</b> . . . . .	258
<b>Arc::Profile</b> . . . . .	258
<b>ArcCredential::PROXYCERTINFO_st</b> . . . . .	259
<b>ArcCredential::PROXYPOLICY_st</b> . . . . .	259
<b>Arc::Query</b> . . . . .	259
<b>Arc::Range&lt; T &gt;</b> . . . . .	261
<b>Arc::Register_Info_Type</b> . . . . .	261
<b>Arc::RegisteredService</b> ( <b>RegisteredService</b> (p. 262) - extension of <b>Service</b> (p. 285) performing self-registration ) . . . . .	262
<b>Arc::RegularExpression</b> (A regular expression class ) . . . . .	262
<b>ArcSec::Request</b> (Base class/Interface for request, includes a container for RequestItems and some operations ) . . . . .	263
<b>ArcSec::RequestAttribute</b> (Wrapper which includes <b>AttributeValue</b> (p.56) object which is generated according to date type of one specific node in Request.xml ) . . . . .	265
<b>ArcSec::RequestItem</b> (Interface for request item container, <subjects, actions, objects, ctxs> tuple ) . . . . .	266
<b>ArcSec::RequestTuple</b> . . . . .	266
<b>Arc::ResourceSlotType</b> . . . . .	266
<b>Arc::ResourcesType</b> . . . . .	266
<b>Arc::ResourceTargetType</b> . . . . .	267
<b>ArcSec::Response</b> (Container for the evaluation results ) . . . . .	267
<b>ArcSec::ResponseItem</b> (Evaluation result concerning one <b>RequestTuple</b> (p. 266) ) . . . . .	267
<b>ArcSec::ResponseList</b> . . . . .	267
<b>Arc::RSL</b> . . . . .	267
<b>Arc::RSLBoolean</b> . . . . .	268

<b>Arc::RSLConcat</b>	268
<b>Arc::RSLCondition</b>	268
<b>Arc::RSLList</b>	269
<b>Arc::RSLLiteral</b>	269
<b>Arc::RSLParser</b>	269
<b>Arc::RSLSequence</b>	270
<b>Arc::RSLValue</b>	270
<b>Arc::RSLVariable</b>	270
<b>Arc::Run</b>	270
<b>Arc::SAML2LoginClient</b>	274
<b>Arc::SAML2SSOHTTPClient</b>	275
<b>Arc::SAMLToken</b> (Class for manipulating SAML Token <b>Profile</b> (p. 258) )	277
<b>Arc::ScalableTime</b> < <b>T</b> >	279
<b>Arc::ScalableTime</b> < <b>int</b> >	280
<b>Arc::SecAttr</b> (This is an abstract interface to a security attribute )	280
<b>Arc::SecAttrFormat</b> (Export/import format )	281
<b>Arc::SecAttrValue</b> (This is an abstract interface to a security attribute )	282
<b>ArcSec::SecHandler</b> (Base class for simple security handling plugins )	283
<b>Arc::SecHandlerConfig</b>	283
<b>ArcSec::SecHandlerConfig</b>	284
<b>ArcSec::SecHandlerPluginArgument</b>	284
<b>ArcSec::Security</b> (Common stuff used by security related classes )	284
<b>Arc::Service</b> ( <b>Service</b> (p. 285) - last component in a <b>Message</b> (p. 210) Chain )	285
<b>Arc::ServicePluginArgument</b>	287
<b>Arc::SimpleCondition</b> (Helper function to create simple thread )	287
<b>Arc::SimpleCounter</b>	288
<b>Arc::SOAPMessage</b> ( <b>Message</b> (p. 210) restricted to SOAP payload )	289
<b>Arc::Software</b> (Used to represent software (names and version) and comparison )	290
<b>Arc::SoftwareRequirement</b> (Class used to express and resolve version requirements on software )	297
<b>ArcSec::Source</b> (Acquires and parses XML document from specified source )	305
<b>ArcSec::SourceFile</b> (Convenience class for obtaining XML document from file )	306
<b>ArcSec::SourceURL</b> (Convenience class for obtaining XML document from remote URL )	306
<b>ArcSec::StringAttribute</b>	307
<b>Arc::Submitter</b> (Base class for the Submitters )	308
<b>Arc::SubmitterLoader</b>	309
<b>Arc::SubmitterPluginArgument</b>	310
<b>Arc::TargetGenerator</b> (Target generation class )	310
<b>Arc::TargetRetriever</b> (TargetRetriever base class )	313
<b>Arc::TargetRetrieverLoader</b>	315
<b>Arc::TargetRetrieverPluginArgument</b>	316
<b>Test::TestMCC</b>	316
<b>Test::TestService</b>	317
<b>Arc::ThreadInitializer</b>	318
<b>Arc::ThreadRegistry</b>	318
<b>Arc::Time</b> (A class for storing and manipulating times )	318
<b>ArcSec::TimeAttribute</b>	321
<b>Arc::URL</b>	323
<b>Arc::URLLocation</b> (Class to hold a resolved <b>URL</b> (p. 323) location )	331
<b>Arc::URLMap</b>	333
<b>Arc::User</b>	333
<b>Arc::UserConfig</b> (User configuration class )	333
<b>Arc::UsernameToken</b> (Interface for manipulation of WS-Security according to Username Token <b>Profile</b> (p. 258) )	360

<b>Arc::UserSwitch</b> . . . . .	362
<b>Arc::VOMSTrustList</b> . . . . .	363
<b>Arc::WSAEndpointReference</b> (Interface for manipulation of WS-Addressing Endpoint Reference) . . . . .	364
<b>Arc::WSAHeader</b> (Interface for manipulation WS-Addressing information in SOAP header) . . . . .	366
<b>Arc::WSRF</b> (Base class for every <b>WSRF</b> (p. 369) message) . . . . .	369
<b>Arc::WSRFBaseFault</b> (Base class for <b>WSRF</b> (p. 369) fault messages) . . . . .	371
<b>Arc::WSRFResourceUnavailableFault</b> . . . . .	372
<b>Arc::WSRFResourceUnknownFault</b> . . . . .	373
<b>Arc::WSRP</b> (Base class for WS-ResourceProperties structures) . . . . .	373
<b>Arc::WSRPDeleteResourceProperties</b> . . . . .	375
<b>Arc::WSRPDeleteResourcePropertiesRequest</b> . . . . .	375
<b>Arc::WSRPDeleteResourcePropertiesRequestFailedFault</b> . . . . .	376
<b>Arc::WSRPDeleteResourcePropertiesResponse</b> . . . . .	376
<b>Arc::WSRPFault</b> (Base class for WS-ResourceProperties faults) . . . . .	376
<b>Arc::WSRPGetMultipleResourcePropertiesRequest</b> . . . . .	377
<b>Arc::WSRPGetMultipleResourcePropertiesResponse</b> . . . . .	378
<b>Arc::WSRPGetResourcePropertyDocumentRequest</b> . . . . .	378
<b>Arc::WSRPGetResourcePropertyDocumentResponse</b> . . . . .	378
<b>Arc::WSRPGetResourcePropertyRequest</b> . . . . .	379
<b>Arc::WSRPGetResourcePropertyResponse</b> . . . . .	379
<b>Arc::WSRPInsertResourceProperties</b> . . . . .	380
<b>Arc::WSRPInsertResourcePropertiesRequest</b> . . . . .	380
<b>Arc::WSRPInsertResourcePropertiesRequestFailedFault</b> . . . . .	380
<b>Arc::WSRPInsertResourcePropertiesResponse</b> . . . . .	381
<b>Arc::WSRPInvalidModificationFault</b> . . . . .	381
<b>Arc::WSRPInvalidResourcePropertyQNameFault</b> . . . . .	382
<b>Arc::WSRPModifyResourceProperties</b> . . . . .	382
<b>Arc::WSRPPutResourcePropertyDocumentRequest</b> . . . . .	383
<b>Arc::WSRPPutResourcePropertyDocumentResponse</b> . . . . .	383
<b>Arc::WSRPQueryResourcePropertiesRequest</b> . . . . .	383
<b>Arc::WSRPQueryResourcePropertiesResponse</b> . . . . .	384
<b>Arc::WSRPResourcePropertyChangeFailure</b> . . . . .	384
<b>Arc::WSRPSetResourcePropertiesRequest</b> . . . . .	385
<b>Arc::WSRPSetResourcePropertiesResponse</b> . . . . .	385
<b>Arc::WSRPSetResourcePropertyRequestFailedFault</b> . . . . .	386
<b>Arc::WSRPUnableToModifyResourcePropertyFault</b> . . . . .	386
<b>Arc::WSRPUnableToPutResourcePropertyDocumentFault</b> . . . . .	387
<b>Arc::WSRPUpdateResourceProperties</b> . . . . .	387
<b>Arc::WSRPUpdateResourcePropertiesRequest</b> . . . . .	387
<b>Arc::WSRPUpdateResourcePropertiesRequestFailedFault</b> . . . . .	388
<b>Arc::WSRPUpdateResourcePropertiesResponse</b> . . . . .	388
<b>ArcSec::X509NameAttribute</b> . . . . .	389
<b>Arc::X509Token</b> (Class for manipulating X.509 Token <b>Profile</b> (p. 258)) . . . . .	390
<b>Arc::XmlContainer</b> . . . . .	391
<b>Arc::XmlDatabase</b> . . . . .	392
<b>Arc::XMLNode</b> (Wrapper for LibXML library Tree interface) . . . . .	392
<b>Arc::XMLNodeContainer</b> . . . . .	403
<b>Arc::XMLSecNode</b> (Extends <b>XMLNode</b> (p. 392) class to support XML security operation) . . . . .	405
<b>Arc::XRSLParser</b> . . . . .	407



# Chapter 4

## File Index

### 4.1 File List

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

<b>AlgFactory.h</b>	??
<b>AnyURIAttribute.h</b>	??
<b>ArcConfig.h</b>	??
<b>ARCJSDLParser.h</b>	??
<b>ArcLocation.h</b>	??
<b>ArcRegex.h</b>	??
<b>AttributeFactory.h</b>	??
<b>AttributeProxy.h</b>	??
<b>AttributeValue.h</b>	??
<b>Base64.h</b>	??
<b>BooleanAttribute.h</b>	??
<b>Broker.h</b>	??
<b>ByteArray.h</b>	??
<b>CertUtil.h</b>	??
<b>Checksum.h</b>	??
<b>CISStringValue.h</b>	??
<b>ClassLoader.h</b>	??
<b>ClientInterface.h</b>	??
<b>ClientSAML2SSO.h</b>	??
<b>ClientX509Delegation.h</b>	??
<b>CombiningAlg.h</b>	??
<b>ConfusaCertHandler.h</b>	??
<b>ConfusaParserUtils.h</b>	??
<b>Counter.h</b>	??
<b>Credential.h</b>	??
<b>CredentialStore.h</b>	??
<b>DataBuffer.h</b>	??
<b>DataCallback.h</b>	??
<b>DataHandle.h</b>	??
<b>DataMover.h</b>	??
<b>DataPoint.h</b>	??
<b>DataPointDirect.h</b>	??
<b>DataPointIndex.h</b>	??

<b>DataSpeed.h</b>	??
<b>DataStatus.h</b>	??
<b>DateTime.h</b>	??
<b>DateTimeAttribute.h</b>	??
<b>DBInterface.h</b>	??
<b>DBranch.h</b>	??
<b>DelegationInterface.h</b>	??
<b>DenyOverridesAlg.h</b>	??
<b>EqualFunction.h</b>	??
<b>EvaluationCtx.h</b>	??
<b>Evaluator.h</b>	??
<b>EvaluatorLoader.h</b>	??
<b>ExecutionTarget.h</b>	??
<b>FileCache.h</b>	??
<b>FileCacheHash.h</b>	??
<b>FileInfo.h</b>	??
<b>FileLock.h</b>	??
<b>FileUtils.h</b>	??
<b>FinderLoader.h</b>	??
<b>FnFactory.h</b>	??
<b>Function.h</b>	??
<b>GenericAttribute.h</b>	??
<b>GlobusErrorUtils.h</b>	??
<b>GlobusWorkarounds.h</b>	??
<b>GSSCredential.h</b>	??
<b>GUID.h</b>	??
<b>HakaClient.h</b>	??
<b>InfoCache.h</b>	??
<b>InfoFilter.h</b>	??
<b>InfoRegister.h</b>	??
<b>InformationInterface.h</b>	??
<b>IniConfig.h</b>	??
<b>InRangeFunction.h</b>	??
<b>IntraProcessCounter.h</b>	??
<b>IString.h</b>	??
<b>JDLParser.h</b>	??
<b>Job.h</b>	??
<b>JobController.h</b>	??
<b>JobDescription.h</b>	??
<b>JobDescriptionParser.h</b>	??
<b>JobState.h</b>	??
<b>JobSupervisor.h</b>	??
<b>listfunc.h</b>	??
<b>Loader.h</b>	??
<b>Logger.h</b>	??
<b>MatchFunction.h</b>	??
<b>MCC.h</b>	??
<b>MCC_Status.h</b>	??
<b>MCCLoader.h</b>	??
<b>Message.h</b>	??
<b>MessageAttributes.h</b>	??
<b>MessageAuth.h</b>	??
<b>ModuleManager.h</b>	??
<b>MysqlWrapper.h</b>	??

<b>OAuthConsumer.h</b>	??
<b>OpenIdpClient.h</b>	??
<b>OpenSSL.h</b>	??
<b>OptionParser.h</b>	??
<b>OrderedAlg.h</b>	??
<b>PayloadRaw.h</b>	??
<b>PayloadSOAP.h</b>	??
<b>PayloadStream.h</b>	??
<b>PayloadWSRF.h</b>	??
<b>PDP.h</b>	??
<b>PermitOverridesAlg.h</b>	??
<b>Plexer.h</b>	??
<b>Plugin.h</b>	??
<b>Policy.h</b>	??
<b>PolicyParser.h</b>	??
<b>PolicyStore.h</b>	??
<b>Profile.h</b>	??
<b>Proxycertinfo.h</b>	??
<b>RegisteredService.h</b>	??
<b>Request.h</b>	??
<b>RequestAttribute.h</b>	??
<b>RequestItem.h</b>	??
<b>Response.h</b>	??
<b>Result.h</b>	??
<b>RSLParser.h</b>	??
<b>Run.h</b>	??
<b>SAML2LoginClient.h</b>	??
<b>saml_util.h</b>	??
<b>SAMLToken.h</b>	??
<b>SecAttr.h</b>	??
<b>SecAttrValue.h</b>	??
<b>SecHandler.h</b>	??
<b>Security.h</b>	??
<b>Service.h</b>	??
<b>SOAPEnvelope.h</b>	??
<b>SOAPMessage.h</b>	??
<b>Software.h</b>	??
<b>Source.h</b>	??
<b>StringAttribute.h</b>	??
<b>StringConv.h</b>	??
<b>Submitter.h</b>	??
<b>TargetGenerator.h</b>	??
<b>TargetRetriever.h</b>	??
<b>loader/TestMCC.h</b>	??
<b>message/TestMCC.h</b>	??
<b>TestService.h</b>	??
<b>Thread.h</b>	??
<b>URL.h</b> (Class to hold general URL's)	409
<b>URLMap.h</b>	??
<b>User.h</b>	??
<b>UserConfig.h</b>	??
<b>UsernameToken.h</b>	??
<b>Utils.h</b>	??
<b>VOMSAttribute.h</b>	??

---

<b>VOMSUtil.h</b>	??
<b>win32.h</b>	??
<b>WSA.h</b>	??
<b>WSResourceProperties.h</b>	??
<b>WSRF.h</b>	??
<b>WSRFBaseFault.h</b>	??
<b>X500NameAttribute.h</b>	??
<b>X509Token.h</b>	??
<b>XmlContainer.h</b>	??
<b>XmlDatabase.h</b>	??
<b>XMLNode.h</b>	??
<b>XMLSecNode.h</b>	??
<b>XmlSecUtils.h</b>	??
<b>XRSLParser.h</b>	??

# Chapter 5

## Namespace Documentation

### 5.1 Arc Namespace Reference

Some utility methods for using xml security library (<http://www.aleksey.com/xmlsec/>).

#### Data Structures

- class **ARCJSDLParser**
- class **Broker**
- class **BrokerLoader**
- class **BrokerPluginArgument**
- class **ClientInterface**
  - *Utility base class for MCC (p. 202).*
- class **ClientTCP**
  - *Class for setting up a MCC (p. 202) chain for TCP communication.*
- struct **HTTPClientInfo**
- class **ClientHTTP**
  - *Class for setting up a MCC (p. 202) chain for HTTP communication.*
- class **ClientSOAP**
- class **SecHandlerConfig**
- class **DNListHandlerConfig**
- class **ARCPolicyHandlerConfig**
- class **ClientHTTPwithSAML2SSO**
- class **ClientSOAPwithSAML2SSO**
- class **ClientX509Delegation**
- class **ConfusaCertHandler**
- class **ConfusaParserUtils**
- class **HakaClient**
- class **OpenIdpClient**
- class **OAuthConsumer**
- class **SAML2LoginClient**
- class **SAML2SSOHTTPClient**

- class **ApplicationEnvironment**  
*ApplicationEnvironment* (p. 50).
- class **ExecutionTarget**  
*ExecutionTarget* (p. 154).
- class **JDLParser**
- class **Job**  
*Job* (p. 184).
- class **JobController**  
*Base class for the JobControllers.*
- class **JobControllerLoader**
- class **JobControllerPluginArgument**
- class **Range**
- class **ScalableTime**
- class **ScalableTime< int >**
- class **JobIdentificationType**
- class **ExecutableType**
- class **NotificationType**
- class **ApplicationType**
- class **ResourceSlotType**
- class **DiskSpaceRequirementType**
- class **ResourceTargetType**
- class **ResourcesType**
- class **DataSourceType**
- class **DataTargetType**
- class **Data**Type
- class **FileType**
- class **DirectoryType**
- class **DataStagingType**
- class **JobMetaType**
- class **JobDescription**
- class **JobDescriptionParser**
- class **JobState**
- class **JobSupervisor**  
*% JobSupervisor* (p. 190) class
- class **RSLValue**
- class **RSLLiteral**
- class **RSLVariable**
- class **RSLConcat**
- class **RSLList**
- class **RSLSequence**
- class **RSL**
- class **RSLBoolean**
- class **RSLCondition**
- class **RSLParser**
- class **Software**

*Used to represent software (names and version) and comparison.*

- class **SoftwareRequirement**

*Class used to express and resolve version requirements on software.*

- class **Submitter**

*Base class for the Submitters.*

- class **SubmitterLoader**

- class **SubmitterPluginArgument**

- class **TargetGenerator**

*Target generation class*

- class **TargetRetriever**

*TargetRetriever base class*

- class **TargetRetrieverLoader**

- class **TargetRetrieverPluginArgument**

- class **XRSLParser**

- class **Config**

*Configuration element - represents (sub)tree of ARC configuration.*

- class **BaseConfig**

- class **ArcLocation**

*Determines ARC installation location.*

- class **RegularExpression**

*A regular expression class.*

- class **Base64**

- class **MemoryAllocationException**

- class **ByteArray**

- class **Counter**

*A class defining a common interface for counters.*

- class **CounterTicket**

*A class for "tickets" that correspond to counter reservations.*

- class **ExpirationReminder**

*A class intended for internal use within counters.*

- class **Period**

- class **Time**

*A class for storing and manipulating times.*

- class **Database**

*Interface for calling database client library.*

- class **Query**

- class **DItem**

- class **DBranch**
- class **DItemString**
- class **FileLock**  
*A general file locking class.*
  
- class **IniConfig**
- class **IntraProcessCounter**  
*A class for counters used by threads within a single process.*
  
- class **PrintFBase**
- class **PrintF**
- class **IString**
- struct **LoggerFormat**
- class **LogMessage**  
*A class for log messages.*
  
- class **LogDestination**  
*A base class for log destinations.*
  
- class **LogStream**  
*A class for logging to ostreams.*
  
- class **LogFile**  
*A class for logging to files.*
  
- class **Logger**  
*A logger class.*
  
- class **MySQLDatabase**
- class **MySQLQuery**
- class **OptionParser**
- class **Profile**
- class **Run**
- class **SimpleCondition**  
*Helper function to create simple thread.*
  
- class **SimpleCounter**
- class **ThreadRegistry**
- class **ThreadInitializer**
- class **URL**
- class **URLLocation**  
*Class to hold a resolved **URL** (p. 323) location.*
  
- class **PathIterator**  
*Class to iterate through elements of path.*
  
- class **User**
- class **UserSwitch**
- class **initializeCredentialsType**
- class **UserConfig**

*User configuration class*

- class **AutoPointer**

*Wrapper for pointer with automatic destruction.*

- class **CountedPointer**

*Wrapper for pointer with automatic destruction and mutiple references.*

- class **NS**

- class **XMLNode**

*Wrapper for LibXML library Tree interface.*

- class **XMLNodeContainer**

- class **CredentialError**

- class **Credential**

- class **VOMSTrustList**

- class **CredentialStore**

- class **Checksum**

*Defines interface for variuos checksum manipulations.*

- class **CRC32Sum**

*Implementation of CRC32 checksum.*

- class **MD5Sum**

*Implementation of MD5 checksum.*

- class **Adler32Sum**

*Implementation of Adler32 checksum.*

- class **ChecksumAny**

*Wrapper for **Checksum** (p. 67) class.*

- class **DataBuffer**

*Represents set of buffers.*

- class **DataCallback**

- class **DataHandle**

*This class is a wrapper around the **DataPoint** (p. 108) class.*

- class **DataMover**

- class **DataPoint**

*This base class is an abstraction of **URL** (p. 323).*

- class **DataPointLoader**

- class **DataPointPluginArgument**

- class **DataPointDirect**

*This is a kind of generalized file handle.*

- class **DataPointIndex**

*Complements **DataPoint** (p. 108) with attributes common for **Indexing Service** (p. 285) URLs.*

- class **DataSpeed**  
*Keeps track of average and instantaneous transfer speed.*
  
- class **DataStatus**
- struct **CacheParameters**
- class **FileCache**
- class **FileInfo**  
***FileInfo** (p. 164) stores information about files (metadata).*
  
- class **URLMap**
- class **XmlContainer**
- class **XmlDatabase**
- class **DelegationConsumer**
- class **DelegationProvider**
- class **DelegationConsumerSOAP**
- class **DelegationProviderSOAP**
- class **DelegationContainerSOAP**
- class **GlobusResult**
- class **GSSCredential**
- class **InfoCache**  
*Stores XML document in filesystem split into parts.*
  
- class **InfoCacheInterface**
- class **InfoFilter**  
*Filters information document according to identity of requestor.*
  
- class **InfoRegister**  
*Registration to ISIS interface.*
  
- class **InfoRegisters**  
*Handling multiple registrations to ISISes.*
  
- struct **Register\_Info\_Type**
- struct **ISIS\_description**
- class **InfoRegistrar**  
*Registration process associated with particular ISIS.*
  
- class **InfoRegisterContainer**
- class **InformationInterface**  
*Information System message processor.*
  
- class **InformationContainer**  
*Information System document container and processor.*
  
- class **InformationRequest**  
*Request for information in InfoSystem.*
  
- class **InformationResponse**  
*Informational response from InfoSystem.*

- class **RegisteredService**  
*RegisteredService* (p. 262) - extension of *Service* (p. 285) performing self-registration.
- class **FinderLoader**
- class **Loader**  
*Plugins loader.*
- class **LoadableModuleDescription**
- class **ModuleManager**  
*Manager of shared libraries.*
- class **Plugin**  
*Base class for loadable ARC components.*
- class **PluginArgument**  
*Base class for passing arguments to loadable ARC components.*
- struct **PluginDescriptor**  
*Description of ARC loadable component.*
- class **PluginDesc**  
*Description of plugin.*
- class **ModuleDesc**  
*Description of loadable module.*
- class **PluginsFactory**  
*Generic ARC plugins loader.*
- class **MCCInterface**  
*Interface for communication between **MCC** (p. 202), **Service** (p. 285) and **Plexer** (p. 247) objects.*
- class **MCC**  
***Message** (p. 210) Chain Component - base class for every **MCC** (p. 202) plugin.*
- class **MCCConfig**
- class **MCCPluginArgument**
- class **MCC\_Status**  
*A class for communication of **MCC** (p. 202) processing results.*
- class **MCCLoader**  
*Creator of **Message** (p. 210) Component Chains (**MCC** (p. 202)).*
- class **ChainContext**  
*Interface to chain specific functionality.*
- class **MessagePayload**  
*Base class for content of message passed through chain.*

- class **MessageContextElement**  
*Top class for elements contained in message context.*
- class **MessageContext**  
*Handler for content of message context.*
- class **MessageAuthContext**  
*Handler for content of message auth\* context.*
- class **Message**  
*Object being passed through chain of MCCs.*
- class **AttributeIterator**  
*A const iterator class for accessing multiple values of an attribute.*
- class **MessageAttributes**  
*A class for storage of attribute values.*
- class **MessageAuth**  
*Contains authenticity information, authorization tokens and decisions.*
- class **PayloadRawInterface**  
*Random Access Payload for **Message** (p. 210) objects.*
- struct **PayloadRawBuf**
- class **PayloadRaw**  
*Raw byte multi-buffer.*
- class **PayloadSOAP**  
*Payload of **Message** (p. 210) with SOAP content.*
- class **PayloadStreamInterface**  
*Stream-like Payload for **Message** (p. 210) object.*
- class **PayloadStream**  
*POSIX handle as Payload.*
- class **PlexerEntry**  
*A pair of label (regex) and pointer to **MCC** (p. 202).*
- class **Plexer**  
*The **Plexer** (p. 247) class, used for routing messages to services.*
- class **CIStrngValue**  
*This class implements case insensitive strings as security attributes.*
- class **SecAttrValue**  
*This is an abstract interface to a security attribute.*
- class **SecAttrFormat**

*Export/import format.*

- class **SecAttr**  
*This is an abstract interface to a security attribute.*
- class **MultiSecAttr**  
*Container of multiple **SecAttr** (p. 280) attributes.*
- class **Service**  
***Service** (p. 285) - last component in a **Message** (p. 210) Chain.*
- class **ServicePluginArgument**
- class **SOAPMessage**  
***Message** (p. 210) restricted to SOAP payload.*
- class **ClassLoader**
- class **ClassLoaderPluginArgument**
- class **WSAEndpointReference**  
*Interface for manipulation of WS-Addressing Endpoint Reference.*
- class **WSAHeader**  
*Interface for manipulation WS-Addressing information in SOAP header.*
- class **SAMLToken**  
*Class for manipulating SAML Token **Profile** (p. 258).*
- class **UsernameToken**  
*Interface for manipulation of WS-Security according to Username Token **Profile** (p. 258).*
- class **X509Token**  
*Class for manipulating X.509 Token **Profile** (p. 258).*
- class **PayloadWSRF**  
*This class combines **MessagePayload** (p. 218) with **WSRF** (p. 369).*
- class **WSRP**  
*Base class for WS-ResourceProperties structures.*
- class **WSRPFault**  
*Base class for WS-ResourceProperties faults.*
- class **WSRPInvalidResourcePropertyQNameFault**
- class **WSRPResourcePropertyChangeFailure**
- class **WSRPUnableToPutResourcePropertyDocumentFault**
- class **WSRPInvalidModificationFault**
- class **WSRPUnableToModifyResourcePropertyFault**
- class **WSRPSetResourcePropertyRequestFailedFault**
- class **WSRPInsertResourcePropertiesRequestFailedFault**
- class **WSRPUpdateResourcePropertiesRequestFailedFault**
- class **WSRPDeleteResourcePropertiesRequestFailedFault**

- class **WSRPGetResourcePropertyDocumentRequest**
- class **WSRPGetResourcePropertyDocumentResponse**
- class **WSRPGetResourcePropertyRequest**
- class **WSRPGetResourcePropertyResponse**
- class **WSRPGetMultipleResourcePropertiesRequest**
- class **WSRPGetMultipleResourcePropertiesResponse**
- class **WSRPPutResourcePropertyDocumentRequest**
- class **WSRPPutResourcePropertyDocumentResponse**
- class **WSRPModifyResourceProperties**
- class **WSRPInsertResourceProperties**
- class **WSRPUpdateResourceProperties**
- class **WSRPDeleteResourceProperties**
- class **WSRPSetResourcePropertiesRequest**
- class **WSRPSetResourcePropertiesResponse**
- class **WSRPInsertResourcePropertiesRequest**
- class **WSRPInsertResourcePropertiesResponse**
- class **WSRPUpdateResourcePropertiesRequest**
- class **WSRPUpdateResourcePropertiesResponse**
- class **WSRPDeleteResourcePropertiesRequest**
- class **WSRPDeleteResourcePropertiesResponse**
- class **WSRPQueryResourcePropertiesRequest**
- class **WSRPQueryResourcePropertiesResponse**
- class **WSRF**

*Base class for every WSRF (p. 369) message.*

- class **WSRFBaseFault**

*Base class for WSRF (p. 369) fault messages.*

- class **WSRFResourceUnknownFault**
- class **WSRFResourceUnavailableFault**
- class **XMLSecNode**

*Extends XMLNode (p. 392) class to support XML security operation.*

## Typedefs

- typedef **Plugin** `((* get_plugin_instance )(PluginArgument *arg)`
- typedef `std::multimap< std::string, std::string >` **AttrMap**
- typedef `AttrMap::const_iterator` **AttrConstIter**
- typedef `AttrMap::iterator` **AttrIter**

## Enumerations

- enum **TimeFormat**
- enum **LogLevel**
- enum **StatusKind** { ,  
**STATUS\_OK** = 1, **GENERIC\_ERROR** = 2, **PARSING\_ERROR** = 4, **PROTOCOL\_-RECOGNIZED\_ERROR** = 8,  
**UNKNOWN\_SERVICE\_ERROR** = 16, **BUSY\_ERROR** = 32, **SESSION\_CLOSE** = 64 }
- enum **WSAFault** { , **WSAFaultUnknown**, **WSAFaultInvalidAddressingHeader** }

## Functions

- `std::ostream & operator<<` (`std::ostream &`, `const Period &`)
- `std::ostream & operator<<` (`std::ostream &`, `const Time &`)
- `std::string TimeStamp` (`const TimeFormat &=Time::GetFormat()`)
- `std::string TimeStamp` (`Time`, `const TimeFormat &=Time::GetFormat()`)
- `int FileOpen` (`const std::string &path`, `int flags`, `mode_t mode=0600`)
- `int FileOpen` (`const std::string &path`, `int flags`, `uid_t uid`, `gid_t gid`, `mode_t mode=0600`)
- `bool FileCopy` (`const std::string &source_path`, `const std::string &destination_path`)
- `bool FileCopy` (`const std::string &source_path`, `int destination_handle`)
- `bool FileCopy` (`int source_handle`, `const std::string &destination_path`)
- `bool FileCopy` (`int source_handle`, `int destination_handle`)
- `Glib::Dir * DirOpen` (`const std::string &path`)
- `Glib::Dir * DirOpen` (`const std::string &path`, `uid_t uid`, `gid_t gid`)
- `bool FileStat` (`const std::string &path`, `struct stat *st`, `bool follow_symlinks`)
- `bool FileStat` (`const std::string &path`, `struct stat *st`, `uid_t uid`, `gid_t gid`, `bool follow_symlinks`)
- `bool DirCreate` (`const std::string &path`, `mode_t mode`, `bool with_parents=false`)
- `bool DirCreate` (`const std::string &path`, `uid_t uid`, `gid_t gid`, `mode_t mode`, `bool with_parents=false`)
- `bool DirDelete` (`const std::string &path`)
- `bool DirDelete` (`const std::string &path`, `uid_t uid`, `gid_t gid`)
- `void GUID` (`std::string &guid`)
- `std::string UUID` (`void`)
- `std::ostream & operator<<` (`std::ostream &os`, `LogLevel level`)
- `LogLevel string_to_level` (`const std::string &str`)
- `bool istring_to_level` (`const std::string &llStr`, `LogLevel &ll`)
- `bool string_to_level` (`const std::string &str`, `LogLevel &ll`)
- `std::string level_to_string` (`const LogLevel &level`)
- `LogLevel old_level_to_level` (`unsigned int old_level`)
- `template<typename T >`  
`T stringto` (`const std::string &s`)
- `template<typename T >`  
`bool stringto` (`const std::string &s`, `T &t`)
- `template<typename T >`  
`std::string tostring` (`T t`, `const int width=0`, `const int precision=0`)
- `std::string lower` (`const std::string &s`)
- `std::string upper` (`const std::string &s`)
- `void tokenize` (`const std::string &str`, `std::vector< std::string > &tokens`, `const std::string &delimiters=" "`)
- `std::string trim` (`const std::string &str`, `const char *sep=NULL`)
- `std::string strip` (`const std::string &str`)
- `std::string uri_unescape` (`const std::string &str`)
- `std::string convert_to_rdn` (`const std::string &dn`)
- `bool CreateThreadFunction` (`void(*func)(void *)`, `void *arg`, `SimpleCounter *count=NULL`)
- `std::list< URL > ReadURLList` (`const URL &urllist`)
- `std::string GetEnv` (`const std::string &var`)
- `std::string GetEnv` (`const std::string &var`, `bool &found`)
- `bool SetEnv` (`const std::string &var`, `const std::string &value`, `bool overwrite=true`)
- `void UnsetEnv` (`const std::string &var`)
- `std::string StrError` (`int errnum=errno`)
- `bool MatchXMLName` (`const XMLNode &node1`, `const XMLNode &node2`)

- bool **MatchXMLName** (const **XMLNode** &node, const char \*name)
- bool **MatchXMLName** (const **XMLNode** &node, const std::string &name)
- bool **MatchXMLNamespace** (const **XMLNode** &node1, const **XMLNode** &node2)
- bool **MatchXMLNamespace** (const **XMLNode** &node, const char \*uri)
- bool **MatchXMLNamespace** (const **XMLNode** &node, const std::string &uri)
- bool **createVOMSAC** (std::string &codedac, **Credential** &issuer\_cred, **Credential** &holder\_cred, std::vector< std::string > &fqan, std::vector< std::string > &targets, std::vector< std::string > &attributes, std::string &voname, std::string &uri, int lifetime)
- bool **addVOMSAC** (**ArcCredential::AC** \*\*&aclist, std::string &acorder, std::string &decodedac)
- bool **parseVOMSAC** (X509 \*holder, const std::string &ca\_cert\_dir, const std::string &ca\_cert\_file, const **VOMSTrustList** &vomscert\_trust\_dn, std::vector< std::string > &output, bool verify=true)
- bool **parseVOMSAC** (const **Credential** &holder\_cred, const std::string &ca\_cert\_dir, const std::string &ca\_cert\_file, const **VOMSTrustList** &vomscert\_trust\_dn, std::vector< std::string > &output, bool verify=true)
- char \* **VOMSDecode** (const char \*data, int size, int \*j)
- const std::string **get\_property** (const **Arc::Credential** &u, const std::string property)
- bool **OpenSSLInit** (void)
- void **HandleOpenSSLError** (void)
- void **HandleOpenSSLError** (int code)
- std::string **string** (**StatusKind** kind)
- const char \* **ContentFromPayload** (const **MessagePayload** &payload)
- void **WSAFaultAssign** (SOAPEnvelope &message, **WSAFault** fid)
- **WSAFault** **WSAFaultExtract** (SOAPEnvelope &message)
- int **passphrase\_callback** (char \*buf, int size, int rflag, void \*)
- bool **init\_xmlsec** (void)
- bool **final\_xmlsec** (void)
- std::string **get\_cert\_str** (const char \*certfile)
- xmlSecKey \* **get\_key\_from\_keystr** (const std::string &value)
- xmlSecKey \* **get\_key\_from\_keyfile** (const char \*keyfile)
- std::string **get\_key\_from\_certfile** (const char \*certfile)
- xmlSecKey \* **get\_key\_from\_certstr** (const std::string &value)
- xmlSecKeysMngrPtr **load\_key\_from\_keyfile** (xmlSecKeysMngrPtr \*keys\_manager, const char \*keyfile)
- xmlSecKeysMngrPtr **load\_key\_from\_certfile** (xmlSecKeysMngrPtr \*keys\_manager, const char \*certfile)
- xmlSecKeysMngrPtr **load\_key\_from\_certstr** (xmlSecKeysMngrPtr \*keys\_manager, const std::string &certstr)
- xmlSecKeysMngrPtr **load\_trusted\_cert\_file** (xmlSecKeysMngrPtr \*keys\_manager, const char \*cert\_file)
- xmlSecKeysMngrPtr **load\_trusted\_cert\_str** (xmlSecKeysMngrPtr \*keys\_manager, const std::string &cert\_str)
- xmlSecKeysMngrPtr **load\_trusted\_certs** (xmlSecKeysMngrPtr \*keys\_manager, const char \*cafile, const char \*capath)
- **XMLNode** **get\_node** (**XMLNode** &parent, const char \*name)

## Variables

- const Glib::TimeVal **ETERNAL**
- const Glib::TimeVal **HISTORIC**
- const size\_t **thread\_stacksize** = (16 \* 1024 \* 1024)
- **Logger** **CredentialLogger**
- const char \* **plugins\_table\_name**

### 5.1.1 Detailed Description

Some utility methods for using xml security library (<http://www.aleksey.com/xmlsec/>).  
**ARCJSDLParser** (p.50) The **ARCJSDLParser** (p.50) class, derived from the **JobDescriptionParser** (p.189) class, is primarily a job description parser for the consolidated job description language (ARCJSDL), derived from JSDL, described in the following document [http://svn.nordugrid.org/trac/nordugrid/browser/arcl/trunk/doc/tech\\_doc/client/job\\_description.odt](http://svn.nordugrid.org/trac/nordugrid/browser/arcl/trunk/doc/tech_doc/client/job_description.odt). However it is also capable of parsing regular JSDL (GFD 136), the POSIX-JSDL extension (GFD 136) and the JSDL HPC **Profile** (p. 258) Application Extension (GFD 111 and GFD 114). When parsing ARCJSDL takes precedence over other non-ARCJSDL, so if a non-ARCJSDL element specifies the same attribute as ARCJSDL, the ARCJSDL element will be saved. The output generated by the `ARCJSDLParser::UnParse` method will follow that of the ARCJSDL document, see reference above.

**JDLParser** (p. 184) The **JDLParser** (p. 184) class, derived from the **JobDescriptionParser** (p. 189) class, is a job description parser for the **Job** (p. 184) Description Language (JDL) specified in CREAM **Job** (p. 184) Description Language Attributes Specification for the EGEE middleware (EGEE-JRA1-TEC-592336) and **Job** (p. 184) Description Language Attributes Specification for the gLite middleware (EGEE-JRA1-TEC-590869-JDL-Attributes-v0-8).

**JobDescription** (p.188) The **JobDescription** (p.188) class is the internal representation of a job description in the ARC-lib. It is structured into a number of other classes/objects which should strictly follow the description given in the job description document [http://svn.nordugrid.org/trac/nordugrid/browser/arcl/trunk/doc/tech\\_doc/client/job\\_description.odt](http://svn.nordugrid.org/trac/nordugrid/browser/arcl/trunk/doc/tech_doc/client/job_description.odt).

The class consist of a parsing method `JobDescription::Parse` which tries to parse the passed source using a number of different parsers. The parser method is complemented by the `JobDescription::UnParse` method, a method to generate a job description document in one of the supported formats. Additionally the internal representation is contained in public members which makes it directly accessible and modifiable from outside the scope of the class.

**JobDescriptionParser** (p. 189) The **JobDescriptionParser** (p. 189) class is abstract which provide a interface for job description parsers. A job description parser should inherit this class and overwrite the `JobDescriptionParser::Parse` and `JobDescriptionParser::UnParse` methods.

**XRSLParser** (p. 407) The **XRSLParser** (p. 407) class, derived from the **JobDescriptionParser** (p. 189) class, is a job description parser for the Extended Resource Specification Language (XRSL) specified in the NORDUGRID-MANUAL-4 document.

**Credential** (p. 90) class covers the functionality about general processing about certificate/key files, including: 1. certificate/key parsing, information extracting (such as subject name, issuer name, lifetime, etc.), chain verifying, extension processing about proxy certinfo, extension processing about other general certificate extension (such as voms attributes, it should be the extension-specific code itself to create, parse and verify the extension, not the **Credential** (p. 90) class. For voms, it is some code about writing and parsing voms-implementing Attribute Certificate/ RFC3281, the voms-attribute is then be looked as a binary part and embeded into extension of X509 certificate/proxy certificate); 2. certificate request, extension emeding and certificate signing, for both proxy certificate and EEC (end entity certificate) certificate The Credential class support PEM, DER PKCS12 credential.

Some implicit idea in the ClassLoader/ModuleManager stuff: `share_lib_name` (e.g. `mccsoap`) should be global identical `plugin_name` (e.g. `__arc_attrfactory_modules__`) should be global identical `desc->name` (e.g. `attr.factory`) should also be global identical

## 5.1.2 Typedef Documentation

### 5.1.2.1 typedef AttrMap::const\_iterator Arc::AttrConstIter

A typedef of a const\_iterator for AttrMap.

This typedef is used as a shorthand for a const\_iterator for AttrMap. It is used extensively within the **MessageAttributes** (p. 212) class as well as the AttributesIterator class, but is not visible externally.

### 5.1.2.2 typedef AttrMap::iterator Arc::AttrIter

A typedef of an (non-const) iterator for AttrMap.

This typedef is used as a shorthand for a (non-const) iterator for AttrMap. It is used in one method within the **MessageAttributes** (p. 212) class, but is not visible externally.

### 5.1.2.3 typedef std::multimap<std::string,std::string> Arc::AttrMap

A typedef of a multimap for storage of message attributes.

This typedef is used as a shorthand for a multimap that uses strings for keys as well as values. It is used within the MessageAttributes class for internal storage of message attributes, but is not visible externally.

### 5.1.2.4 typedef Plugin>(\* Arc::get\_plugin\_instance)(PluginArgument \*arg)

Constructor function of ARC loadable component.

This function is called with plugin-specific argument and should produce and return valid instance of plugin. If plugin can't be produced by any reason (for example because passed argument is not applicable) then NULL is returned. No exceptions should be raised.

## 5.1.3 Enumeration Type Documentation

### 5.1.3.1 enum Arc::LogLevel

Logging levels.

Logging levels for tagging and filtering log messages. FATAL level designates very severe error events that will presumably lead the application to abort. ERROR level designates error events that might still allow the application to continue running. WARNING level designates potentially harmful situations. INFO level designates informational messages that highlight the progress of the application at coarse-grained level. VERBOSE level designates fine-grained informational events that will give additional information about the application. DEBUG level designates finer-grained informational events which should only be used for debugging purposes.

### 5.1.3.2 enum Arc::StatusKind

Status kinds (types).

This enum defines a set of possible status kinds.

#### Enumerator:

*STATUS\_OK* Default status - undefined error.

**GENERIC\_ERROR** No error.

**PARSING\_ERROR** Error does not fit any class.

**PROTOCOL\_RECOGNIZED\_ERROR** Error detected while parsing request/response.

**UNKNOWN\_SERVICE\_ERROR Message** (p. 210) does not fit into expected protocol.

**BUSY\_ERROR** There is no destination configured for this message.

**SESSION\_CLOSE Message** (p. 210) can't be processed now.

### 5.1.3.3 enum Arc::WSAFault

WS-Addressing possible faults.

#### Enumerator:

**WSAFaultUnknown** This is not a fault

**WSAFaultInvalidAddressingHeader** This is not a WS-Addressing fault

## 5.1.4 Function Documentation

### 5.1.4.1 bool Arc::addVOMSAC ( ArcCredential::AC \*\*& *aclist*, std::string & *acorder*, std::string & *decodedac* )

Add decoded AC string into a list of AC objects

#### Parameters

*aclist* The list of AC objects (output)

*acorder* The order of AC objects (output)

*decodedac* The AC string that is decoded from the string returned from voms server (input)

### 5.1.4.2 const char\* Arc::ContentFromPayload ( const MessagePayload & *payload* )

Returns pointer to main memory chunk of **Message** (p. 210) payload.

If no buffer is present or if payload is not of **PayloadRawInterface** (p. 232) type NULL is returned.

### 5.1.4.3 bool Arc::CreateThreadFunction ( void(\*)(void \*) *func*, void \* *arg*, SimpleCounter \* *count = NULL* )

This macro behaves like function which makes thread of class' method.

It accepts class instance and full name of method - like class::method. 'method' should not be static member of the class. Result is true if creation of thread succeeded. Specified instance must be valid during whole lifetime of thread. So probably it is safer to destroy 'instance' in 'method' just before exiting. Helper function to create simple thread. It takes care of all peculiarities of Glib::Thread API. As result it runs function 'func' with argument 'arg' in a separate thread. If count parameter not NULL then corresponding object will be incremented before function returns and then decremented then thread finished. Returns true on success.

**5.1.4.4** `bool Arc::createVOMSAC ( std::string & codedac, Credential & issuer_cred, Credential & holder_cred, std::vector< std::string > & fqn, std::vector< std::string > & targets, std::vector< std::string > & attributes, std::string & voname, std::string & uri, int lifetime )`

Create AC(Attribute Certificate) with voms specific format.

#### Parameters

*codedac* The coded AC as output of this method

*issuer\_cred* The issuer credential which is used to sign the AC

*holder\_cred* The holder credential, the holder certificate is the one which carries AC The rest arguments are the same as the above method

**5.1.4.5** `int Arc::FileOpen ( const std::string & path, int flags, mode_t mode = 0600 )`

Utility functions for handling files and directories.

Open a file and return a file handle

**5.1.4.6** `bool Arc::final_xmlsec ( void )`

Finalize the xml security library

**5.1.4.7** `std::string Arc::get_cert_str ( const char * certfile )`

Get certificate in string format from certificate file

**5.1.4.8** `std::string Arc::get_key_from_certfile ( const char * certfile )`

Get public key in string format from certificate file

**5.1.4.9** `xmlSecKey* Arc::get_key_from_certstr ( const std::string & value )`

Get public key in xmlSecKey structure from certificate string (the string under "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----")

**5.1.4.10** `xmlSecKey* Arc::get_key_from_keyfile ( const char * keyfile )`

Get key in xmlSecKey structure from key file

**5.1.4.11** `xmlSecKey* Arc::get_key_from_keystr ( const std::string & value )`

Get key in xmlSecKey structure from key in string format

**5.1.4.12** `XMLNode Arc::get_node ( XMLNode & parent, const char * name )`

Generate a new child **XMLNode** (p. 392) with specified name

**5.1.4.13** `const std::string Arc::get_property ( const Arc::Credential & u, const std::string property )`

Extract the needed field from the certificate

**5.1.4.14** `void Arc::GUID ( std::string & guid )`

Utilities for generating unique identifiers in the form 12345678-90ab-cdef-1234-567890abcdef.

Generates a unique identifier using information such as IP address, current time etc.

**5.1.4.15** `bool Arc::init_xmlsec ( void )`

Initialize the xml security library, it should be called before the xml security functionality is used.

**5.1.4.16** `bool Arc::istring_to_level ( const std::string & lStr, LogLevel & ll )`

Case-insensitive parsing of a string to a LogLevel with error response.

The method will try to parse (case-insensitive) the argument string to a corresponding LogLevel. If the method succeeds, true will be returned and the argument *ll* will be set to the parsed LogLevel. If the parsing fails `false` will be returned. The parsing succeeds if *lStr* match (case-insensitively) one of the names of the LogLevel members.

**Parameters**

*lStr* a string which should be parsed to a `Arc::LogLevel` (p. 36).

*ll* a `Arc::LogLevel` (p. 36) reference which will be set to the matching `Arc::LogLevel` (p. 36) upon successful parsing.

**Returns**

`true` in case of successful parsing, otherwise `false`.

**See also**

`LogLevel` (p. 36)

**5.1.4.17** `xmlSecKeysMngrPtr Arc::load_key_from_certfile ( xmlSecKeysMngrPtr * keys_manager, const char * certfile )`

Load public key from a certificate file into key manager

**5.1.4.18** `xmlSecKeysMngrPtr Arc::load_key_from_certstr ( xmlSecKeysMngrPtr * keys_manager, const std::string & certstr )`

Load public key from a certificate string into key manager

**5.1.4.19** `xmlSecKeysMngrPtr Arc::load_key_from_keyfile ( xmlSecKeysMngrPtr * keys_manager, const char * keyfile )`

Load private or public key from a key file into key manager

**5.1.4.20** `xmlSecKeysMngrPtr Arc::load_trusted_cert_file ( xmlSecKeysMngrPtr * keys_manager,  
const char * cert_file )`

Load trusted certificate from certificate file into key manager

**5.1.4.21** `xmlSecKeysMngrPtr Arc::load_trusted_cert_str ( xmlSecKeysMngrPtr * keys_manager,  
const std::string & cert_str )`

Load trusted certificate from certificate string into key manager

**5.1.4.22** `xmlSecKeysMngrPtr Arc::load_trusted_certs ( xmlSecKeysMngrPtr * keys_manager,  
const char * cafile, const char * capath )`

Load trusted certificates from a file or directory into key manager

**5.1.4.23** `bool Arc::MatchXMLName ( const XMLNode & node1, const XMLNode & node2 )`

Returns true if underlying XML elements have same names

**5.1.4.24** `bool Arc::MatchXMLName ( const XMLNode & node, const char * name )`

Returns true if 'name' matches name of 'node'. If name contains prefix it's checked too

**5.1.4.25** `bool Arc::MatchXMLName ( const XMLNode & node, const std::string & name )`

Returns true if 'name' matches name of 'node'. If name contains prefix it's checked too

**5.1.4.26** `bool Arc::MatchXMLNamespace ( const XMLNode & node, const std::string & uri )`

Returns true if 'namespace' matches 'node's namespace.

**5.1.4.27** `bool Arc::MatchXMLNamespace ( const XMLNode & node1, const XMLNode & node2  
)`

Returns true if underlying XML elements belong to same namespaces

**5.1.4.28** `bool Arc::MatchXMLNamespace ( const XMLNode & node, const char * uri )`

Returns true if 'namespace' matches 'node's namespace.

**5.1.4.29** `bool Arc::OpenSSLInit ( void )`

This module contains various convenience utilities for using OpenSSL.

Application may be linked to this module instead of OpenSSL libraries directly. This function initializes OpenSSL library. It may be called multiple times and makes sure everything is done properly and OpenSSL may be used in multi-threaded environment. Because this function makes use of `ArcLocation` (p. 51) it is advisable to call it after `ArcLocation::Init()` (p. 51).

**5.1.4.30** `std::ostream& Arc::operator<< ( std::ostream & , const Period & )`

Prints a Period-object to the given ostream -- typically cout.

**5.1.4.31** `std::ostream& Arc::operator<< ( std::ostream & , const Time & )`

Prints a Time-object to the given ostream -- typically cout.

**5.1.4.32** `std::ostream& Arc::operator<< ( std::ostream & os, LogLevel level )`

Printing of LogLevel values to ostreams.

Output operator so that LogLevel values can be printed in a nicer way.

**5.1.4.33** `bool Arc::parseVOMSAC ( X509 * holder, const std::string & ca_cert_dir, const std::string & ca_cert_file, const VOMSTrustList & vomscert_trust_dn, std::vector< std::string > & output, bool verify = true )`

Parse the certificate, and output the attributes.

**Parameters**

**holder** The proxy certificate which includes the voms specific formatted AC.

**ca\_cert\_dir** The trusted certificates which are used to verify the certificate which is used to sign the AC

**ca\_cert\_file** The same as ca\_cert\_dir except it is a file instead of a directory. Only one of them need to be set

**vomsdir** The directory which include \*.lsc file for each vo. For instance, a vo called "knowarc.eu" should have file \$prefix/vomsdir/knowarc/voms.knowarc.eu.lsc which contains on the first line the DN of the VOMS server, and on the second line the corresponding CA DN: /O=Grid/O=NorduGrid/OU=KnowARC/CN=voms.knowarc.eu/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority See more in : <https://twiki.cern.ch/twiki/bin/view/LCG/VomsFAQforServiceManagers>

**output** The parsed attributes (Role and Generic Attribute) . Each attribute is stored in element of a vector as a string. It is up to the consumer to understand the meaning of the attribute. There are two types of attributes stored in VOMS AC: AC\_IETFATTR, AC\_FULL\_ATTRIBUTES. The AC\_IETFATTR will be like /Role=Employee/Group=Tester/Capability=NULL The AC\_FULL\_ATTRIBUTES will be like knowarc:Degree=PhD (qualifier::name=value) In order to make the output attribute values be identical, the voms server information is added as prefix of the original attributes in AC. for AC\_FULL\_ATTRIBUTES, the voname + hostname is added: /voname=knowarc.eu/hostname=arthur.hep.lu.se:15001//knowarc.eu/coredev:attribute1=1 for AC\_IETFATTR, the 'VO' (voname) is added: /VO=knowarc.eu/Group=coredev/Role=NULL/Capability=NULL /VO=knowarc.eu/Group=testers/Role=NULL/Capability=NULL

some other redundant attributes is provided: voname=knowarc.eu/hostname=arthur.hep.lu.se:15001

**Parameters**

**verify** true: Verify the voms certificate is trusted based on the ca\_cert\_dir/ca\_cert\_file which specifies the CA certificates, and the vomscert\_trust\_dn which specifies the trusted DN chain from voms server certificate to CA certificate.

false: Not verify, which means the issuer of AC (voms server certificate is supposed to be trusted by default). In this case the parameters 'ca\_cert\_dir', 'ca\_cert\_file' and 'vomscert\_trust\_dn' will not effect, and should be set as empty. This case is specifically used by 'arcproxy --info' to list all of the attributes in AC, and not to need to verify if the AC's issuer is trusted.

**5.1.4.34** `bool Arc::parseVOMSAC ( const Credential & holder_cred, const std::string & ca_cert_dir, const std::string & ca_cert_file, const VOMSTrustList & vomscert_trust_dn, std::vector< std::string > & output, bool verify = true )`

Parse the certificate. The same as the above one

**5.1.4.35** `int Arc::passphrase_callback ( char * buf, int size, int rwflag, void * )`

callback method for inputing passphrase of key file

**5.1.4.36** `std::string Arc::string ( StatusKind kind )`

Conversion to string.

Conversion from StatusKind to string.

#### Parameters

*kind* The StatusKind to convert.

**5.1.4.37** `std::string Arc::TimeStamp ( Time, const TimeFormat & = Time::GetFormat () )`

Returns a time-stamp of some specified time in some format.

**5.1.4.38** `std::string Arc::TimeStamp ( const TimeFormat & = Time::GetFormat () )`

Returns a time-stamp of the current time in some format.

**5.1.4.39** `char* Arc::VOMSDecode ( const char * data, int size, int * j )`

Decode the data which is encoded by voms server. Since voms code uses some specific coding method (not base64 encoding), we simply copy the method from voms code to here

**5.1.4.40** `void Arc::WSAFaultAssign ( SOAPEnvelope & message, WSAFault fid )`

Makes WS-Addressing fault.

It fills SOAP Fault message with WS-Addressing fault related information.

**5.1.4.41** `WSAFault Arc::WSAFaultExtract ( SOAPEnvelope & message )`

Gets WS-addressing fault.

Analyzes SOAP Fault message and returns WS-Addressing fault it represents.

## 5.1.5 Variable Documentation

### 5.1.5.1 Logger Arc::CredentialLogger

**Logger** (p. 195) to be used by all modules of credentials library

### 5.1.5.2 const char\* Arc::plugins\_table\_name

Name of symbol referring to table of plugins.

This C null terminated string specifies name of symbol which shared library should export to give an access to an array of **PluginDescriptor** (p. 252) elements. The array is terminated by element with all components set to NULL.

### 5.1.5.3 const size\_t Arc::thread\_stacksize = (16 \* 1024 \* 1024)

This module provides convenient helpers for Glibmm interface for thread management.

So far it takes care of automatic initialization of threading environment and creation of simple detached threads. Always use it instead of glibmm/thread.h and keep among first includes. It safe to use it multiple times and to include it both from source files and other include files. Defines size of stack assigned to every new thread.

## 5.2 ArcCredential Namespace Reference

### Data Structures

- struct **cert\_verify\_context**
- struct **PROXYPOLICY\_st**
- struct **PROXYCERTINFO\_st**
- struct **ACDIGEST**
- struct **ACIS**
- struct **ACFORM**
- struct **ACACI**
- struct **ACHOLDER**
- struct **ACVAL**
- struct **ACIETFATTR**
- struct **ACTARGET**
- struct **ACTARGETS**
- struct **ACATTR**
- struct **ACINFO**
- struct **ACC**
- struct **ACSEQ**
- struct **ACCERTS**
- struct **ACATTRIBUTE**
- struct **ACATTHOLDER**
- struct **ACFULLATTRIBUTES**

## Enumerations

- enum `certType` {
  - `CERT_TYPE_EEC`, `CERT_TYPE_CA`, `CERT_TYPE_GSI_3_IMPERSONATION_PROXY`, `CERT_TYPE_GSI_3_INDEPENDENT_PROXY`,
  - `CERT_TYPE_GSI_3_LIMITED_PROXY`, `CERT_TYPE_GSI_3_RESTRICTED_PROXY`, `CERT_TYPE_GSI_2_PROXY`, `CERT_TYPE_GSI_2_LIMITED_PROXY`,
  - `CERT_TYPE_RFC_IMPERSONATION_PROXY`, `CERT_TYPE_RFC_INDEPENDENT_PROXY`, `CERT_TYPE_RFC_LIMITED_PROXY`, `CERT_TYPE_RFC_RESTRICTED_PROXY`,
  - `CERT_TYPE_RFC_ANYLANGUAGE_PROXY` }

### 5.2.1 Detailed Description

Functions and constants for maintaining proxy certificates The code is derived from globus gsi, voms, and openssl-0.9.8e. The existing code for maintaining proxy certificates in OpenSSL only covers standard proxies and does not cover old Globus proxies, so here the Globus code is introduced.

Borrow the code about Attribute Certificate from VOMS The `VOMSAttribute.h` (p.??) and `VOMSAttribute.cpp` are integration about code written by VOMS project, so here the original license follows.

### 5.2.2 Enumeration Type Documentation

#### 5.2.2.1 enum `ArcCredential::certType`

##### Enumerator:

- `CERT_TYPE_EEC` A end entity certificate
- `CERT_TYPE_CA` A CA certificate
- `CERT_TYPE_GSI_3_IMPERSONATION_PROXY` A X.509 Proxy Certificate Profile (pre-RFC) compliant impersonation proxy
- `CERT_TYPE_GSI_3_INDEPENDENT_PROXY` A X.509 Proxy Certificate Profile (pre-RFC) compliant independent proxy
- `CERT_TYPE_GSI_3_LIMITED_PROXY` A X.509 Proxy Certificate Profile (pre-RFC) compliant limited proxy
- `CERT_TYPE_GSI_3_RESTRICTED_PROXY` A X.509 Proxy Certificate Profile (pre-RFC) compliant restricted proxy
- `CERT_TYPE_GSI_2_PROXY` A legacy Globus impersonation proxy
- `CERT_TYPE_GSI_2_LIMITED_PROXY` A legacy Globus limited impersonation proxy
- `CERT_TYPE_RFC_IMPERSONATION_PROXY` A X.509 Proxy Certificate Profile RFC compliant impersonation proxy; RFC inheritAll proxy
- `CERT_TYPE_RFC_INDEPENDENT_PROXY` A X.509 Proxy Certificate Profile RFC compliant independent proxy; RFC independent proxy
- `CERT_TYPE_RFC_LIMITED_PROXY` A X.509 Proxy Certificate Profile RFC compliant limited proxy
- `CERT_TYPE_RFC_RESTRICTED_PROXY` A X.509 Proxy Certificate Profile RFC compliant restricted proxy
- `CERT_TYPE_RFC_ANYLANGUAGE_PROXY` RFC anyLanguage proxy

## Chapter 6

# Data Structure Documentation

### 6.1 ArcCredential::ACACI Struct Reference

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

- VOMSAttribute.h

### 6.2 ArcCredential::ACATTHOLDER Struct Reference

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

- VOMSAttribute.h

### 6.3 ArcCredential::ACATTR Struct Reference

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

- VOMSAttribute.h

### 6.4 ArcCredential::ACATTRIBUTE Struct Reference

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

- VOMSAttribute.h

### 6.5 ArcCredential::ACC Struct Reference

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

- VOMSAttribute.h

## 6.6 ArcCredential::ACCERTS Struct Reference

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

- VOMSAttribute.h

## 6.7 ArcCredential::ACDIGEST Struct Reference

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

- VOMSAttribute.h

## 6.8 ArcCredential::ACFORM Struct Reference

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

- VOMSAttribute.h

## 6.9 ArcCredential::ACFULLATTRIBUTES Struct Reference

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

- VOMSAttribute.h

## 6.10 ArcCredential::ACHOLDER Struct Reference

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

- VOMSAttribute.h

## 6.11 ArcCredential::ACIETFATTR Struct Reference

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

- VOMSAttribute.h

## 6.12 ArcCredential::ACINFO Struct Reference

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

- VOMSAttribute.h

## 6.13 ArcCredential::ACIS Struct Reference

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

- VOMSAttribute.h

## 6.14 ArcCredential::ACSEQ Struct Reference

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

- VOMSAttribute.h

## 6.15 ArcCredential::ACTARGET Struct Reference

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

- VOMSAttribute.h

## 6.16 ArcCredential::ACTARGETS Struct Reference

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

- VOMSAttribute.h

## 6.17 ArcCredential::ACVAL Struct Reference

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

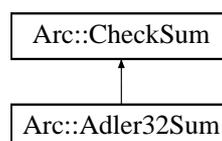
- VOMSAttribute.h

## 6.18 Arc::Adler32Sum Class Reference

Implementation of Adler32 checksum.

```
#include <Checksum.h>
```

Inheritance diagram for Arc::Adler32Sum:



### 6.18.1 Detailed Description

Implementation of Adler32 checksum.

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

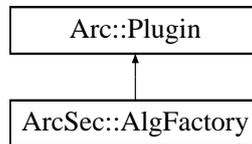
- CheckSum.h

## 6.19 ArcSec::AlgFactory Class Reference

Interface for algorithm factory class.

```
#include <AlgFactory.h>
```

Inheritance diagram for ArcSec::AlgFactory:



### Public Member Functions

- virtual **CombiningAlg** \* **createAlg** (const std::string &type)=0

### 6.19.1 Detailed Description

Interface for algorithm factory class. **AlgFactory** (p. 48) is in charge of creating **CombiningAlg** (p. 75) according to the algorithm type given as argument of method **createAlg**. This class can be inherited for implementing a factory class which can create some specific combining algorithm objects.

### 6.19.2 Member Function Documentation

#### 6.19.2.1 virtual **CombiningAlg**\* ArcSec::AlgFactory::createAlg ( const std::string & type ) [pure virtual]

creat algorithm object based on the type algorithm type

#### Parameters

*type* The type of combining algorithm

#### Returns

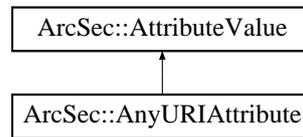
The object of **CombiningAlg** (p. 75)

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

- AlgFactory.h

## 6.20 ArcSec::AnyURIAttribute Class Reference

Inheritance diagram for ArcSec::AnyURIAttribute:



### Public Member Functions

- virtual bool **equal** (**AttributeValue** \*other, bool check\_id=true)
- virtual std::string **encode** ()
- std::string **getId** ()
- virtual std::string **getType** ()

### 6.20.1 Member Function Documentation

#### 6.20.1.1 virtual std::string ArcSec::AnyURIAttribute::encode ( ) [inline, virtual]

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 57).

#### 6.20.1.2 virtual bool ArcSec::AnyURIAttribute::equal ( AttributeValue \* value, bool check\_id = true ) [virtual]

Evaluate whether "this" equals to the parameter value

Implements **ArcSec::AttributeValue** (p. 58).

#### 6.20.1.3 std::string ArcSec::AnyURIAttribute::getId ( ) [inline, virtual]

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 58).

#### 6.20.1.4 virtual std::string ArcSec::AnyURIAttribute::getType ( ) [inline, virtual]

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 58).

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

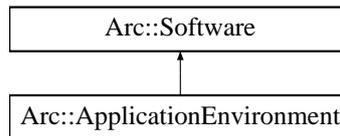
- AnyURIAttribute.h

## 6.21 Arc::ApplicationEnvironment Class Reference

**ApplicationEnvironment** (p. 50).

```
#include <ExecutionTarget.h>
```

Inheritance diagram for Arc::ApplicationEnvironment:



### 6.21.1 Detailed Description

**ApplicationEnvironment** (p. 50). The ApplicationEnvironment is closely related to the definition given in GLUE2. By extending the **Software** (p. 290) class the two GLUE2 attributes `AppName` and `AppVersion` are mapped to two private members. However these can be obtained through the inherited member methods `getName` and `getVersion`.

GLUE2 description: A description of installed application software or software environment characteristics available within one or more Execution Environments.

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

- ExecutionTarget.h

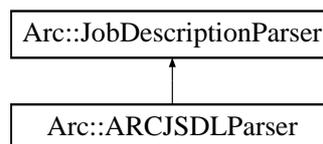
## 6.22 Arc::ApplicationType Class Reference

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

- JobDescription.h

## 6.23 Arc::ARCJSDLParser Class Reference

Inheritance diagram for Arc::ARCJSDLParser:



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

- ARCJSDLParser.h

## 6.24 Arc::ArcLocation Class Reference

Determines ARC installation location.

```
#include <ArcLocation.h>
```

### Static Public Member Functions

- static void **Init** (std::string path)
- static const std::string & **Get** ()
- static std::list< std::string > **GetPlugins** ()

#### 6.24.1 Detailed Description

Determines ARC installation location.

#### 6.24.2 Member Function Documentation

##### 6.24.2.1 static std::list<std::string> Arc::ArcLocation::GetPlugins ( ) [static]

Returns ARC plugins directory location.

Main source is value of variable ARC\_PLUGIN\_PATH, otherwise path is derived from installation location.

##### 6.24.2.2 static void Arc::ArcLocation::Init ( std::string path ) [static]

Initializes location information.

Main source is value of variable ARC\_LOCATION, otherwise path to executable provided in is used. If nothing works then warning message is sent to logger and initial installation prefix is used.

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

- ArcLocation.h

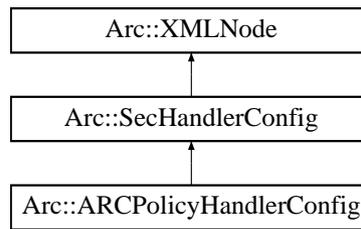
## 6.25 ArcSec::ArcPeriod Struct Reference

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

- DateTimeAttribute.h

## 6.26 Arc::ARCPolicyHandlerConfig Class Reference

Inheritance diagram for Arc::ARCPolicyHandlerConfig:



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

- ClientInterface.h

## 6.27 ArcSec::Attr Struct Reference

**Attr** (p. 52) contains a tuple of attribute type and value.

```
#include <Request.h>
```

### 6.27.1 Detailed Description

**Attr** (p. 52) contains a tuple of attribute type and value.

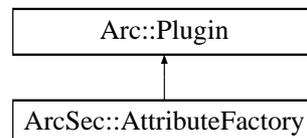
The documentation for this struct was generated from the following file:

- Request.h

## 6.28 ArcSec::AttributeFactory Class Reference

```
#include <AttributeFactory.h>
```

Inheritance diagram for ArcSec::AttributeFactory:



### 6.28.1 Detailed Description

Base attribute factory class

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

- AttributeFactory.h

## 6.29 Arc::AttributeIterator Class Reference

A const iterator class for accessing multiple values of an attribute.

```
#include <MessageAttributes.h>
```

### Public Member Functions

- **AttributeIterator** ()
- const std::string & **operator\*** () const
- const std::string \* **operator->** () const
- const std::string & **key** (void) const
- const **AttributeIterator** & **operator++** ()
- **AttributeIterator** **operator++** (int)
- bool **hasMore** () const

### Protected Member Functions

- **AttributeIterator** (AttrConstIter begin, AttrConstIter end)

### Protected Attributes

- AttrConstIter **current\_**
- AttrConstIter **end\_**

### Friends

- class **MessageAttributes**

#### 6.29.1 Detailed Description

A const iterator class for accessing multiple values of an attribute. This is an iterator class that is used when accessing multiple values of an attribute. The `getAll()` method of the **MessageAttributes** (p. 212) class returns an **AttributeIterator** (p. 53) object that can be used to access the values of the attribute.

Typical usage is:

```
MessageAttributes attributes;
...
for (AttributeIterator iterator=attributes.getAll("Foo:Bar");
     iterator.hasMore(); ++iterator)
    std::cout << *iterator << std::endl;
```

#### 6.29.2 Constructor & Destructor Documentation

##### 6.29.2.1 Arc::AttributeIterator::AttributeIterator ( )

Default constructor.

The default constructor. Does nothing since all attributes are instances of well-behaving STL classes.

### 6.29.2.2 `Arc::AttributeIterator::AttributeIterator ( AttrConstIter begin, AttrConstIter end )` [protected]

Protected constructor used by the `MessageAttributes` (p. 212) class.

This constructor is used to create an iterator for iteration over all values of an attribute. It is not supposed to be visible externally, but is only used from within the `getAll()` method of `MessageAttributes` (p. 212) class.

#### Parameters

*begin* A `const_iterator` pointing to the first matching key-value pair in the internal multimap of the `MessageAttributes` (p. 212) class.

*end* A `const_iterator` pointing to the first key-value pair in the internal multimap of the `MessageAttributes` (p. 212) class where the key is larger than the key searched for.

## 6.29.3 Member Function Documentation

### 6.29.3.1 `bool Arc::AttributeIterator::hasMore ( ) const`

Predicate method for iteration termination.

This method determines whether there are more values for the iterator to refer to.

#### Returns

Returns true if there are more values, otherwise false.

### 6.29.3.2 `const std::string& Arc::AttributeIterator::key ( void ) const`

The key of attribute.

This method returns reference to key of attribute to which iterator refers.

### 6.29.3.3 `const std::string& Arc::AttributeIterator::operator* ( ) const`

The dereference operator.

This operator is used to access the current value referred to by the iterator.

#### Returns

A (constant reference to a) string representation of the current value.

### 6.29.3.4 `const AttributeIterator& Arc::AttributeIterator::operator++ ( )`

The prefix advance operator.

Advances the iterator to the next value. Works intuitively.

#### Returns

A const reference to this iterator.

### 6.29.3.5 AttributeIterator Arc::AttributeIterator::operator++ ( int )

The postfix advance operator.

Advances the iterator to the next value. Works intuitively.

#### Returns

An iterator referring to the value referred to by this iterator before the advance.

### 6.29.3.6 const std::string\* Arc::AttributeIterator::operator-> ( ) const

The arrow operator.

Used to call methods for value objects (strings) conveniently.

## 6.29.4 Friends And Related Function Documentation

### 6.29.4.1 friend class MessageAttributes [friend]

The **MessageAttributes** (p. 212) class is a friend.

The constructor that creates an **AttributeIterator** (p. 53) that is connected to the internal multimap of the **MessageAttributes** (p. 212) class should not be exposed to the outside, but it still needs to be accessible from the `getAll()` method of the **MessageAttributes** (p. 212) class. Therefore, that class is a friend.

## 6.29.5 Field Documentation

### 6.29.5.1 AttrConstIter Arc::AttributeIterator::current\_ [protected]

A `const_iterator` pointing to the current key-value pair.

This iterator is the internal representation of the current value. It points to the corresponding key-value pair in the internal multimap of the **MessageAttributes** (p. 212) class.

### 6.29.5.2 AttrConstIter Arc::AttributeIterator::end\_ [protected]

A `const_iterator` pointing beyond the last key-value pair.

A `const_iterator` pointing to the first key-value pair in the internal multimap of the **MessageAttributes** (p. 212) class where the key is larger than the key searched for.

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

- MessageAttributes.h

## 6.30 ArcSec::AttributeProxy Class Reference

Interface for creating the **AttributeValue** (p. 56) object, it will be used by **AttributeFactory** (p. 52).

```
#include <AttributeProxy.h>
```

## Public Member Functions

- virtual **AttributeValue** \* **getAttribute** (const **Arc::XMLNode** &node)=0

### 6.30.1 Detailed Description

Interface for creating the **AttributeValue** (p. 56) object, it will be used by **AttributeFactory** (p. 52). The **AttributeProxy** (p. 55) object will be insert into AttributeFactory; and the **getAttribute(node)** method will be called inside AttributeFactory.createvalue(node), in order to create a specific **AttributeValue** (p. 56)

### 6.30.2 Member Function Documentation

#### 6.30.2.1 virtual **AttributeValue**\* **ArcSec::AttributeProxy::getAttribute** ( const **Arc::XMLNode** & *node* ) [pure virtual]

Create a **AttributeValue** (p. 56) object according to the information inside the **XMLNode** as parameter.

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

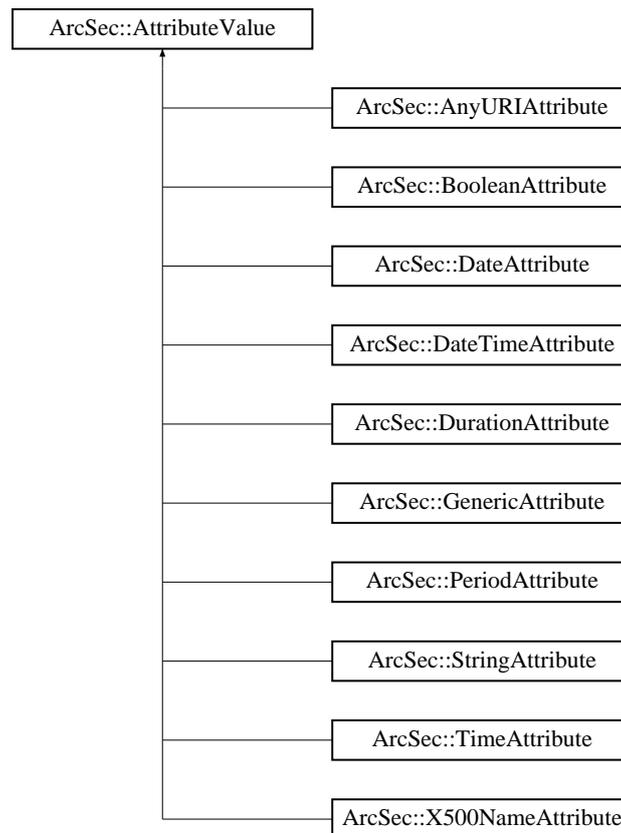
- AttributeProxy.h

## 6.31 **ArcSec::AttributeValue** Class Reference

Interface for containing different type of <Attribute> node for both policy and request.

```
#include <AttributeValue.h>
```

Inheritance diagram for **ArcSec::AttributeValue**:



## Public Member Functions

- virtual bool **equal** (AttributeValue \*value, bool check\_id=true)=0
- virtual std::string **encode** ()=0
- virtual std::string **getType** ()=0
- virtual std::string **getId** ()=0

### 6.31.1 Detailed Description

Interface for containing different type of <Attribute> node for both policy and request. <Attribute> contains different "Type" definition; Each type of <Attribute> needs different approach to compare the value. Any specific class which is for processing specific "Type" should inherit this class. The "Type" supported so far is: **StringAttribute** (p. 307), **DateAttribute** (p. 134), **TimeAttribute** (p. 321), **DurationAttribute** (p. 147), **PeriodAttribute** (p. 245), **AnyURIAttribute** (p. 49), **X500NameAttribute** (p. 389)

### 6.31.2 Member Function Documentation

#### 6.31.2.1 virtual std::string ArcSec::AttributeValue::encode ( ) [pure virtual]

encode the value in a string format

Implemented in **ArcSec::AnyURIAttribute** (p. 49), **ArcSec::BooleanAttribute** (p. 62), **ArcSec::DateTimeAttribute** (p. 136), **ArcSec::TimeAttribute** (p. 322), **ArcSec::DateAttribute** (p. 134),

[ArcSec::DurationAttribute](#) (p. 147), [ArcSec::PeriodAttribute](#) (p. 245), [ArcSec::GenericAttribute](#) (p. 167), [ArcSec::StringAttribute](#) (p. 307), and [ArcSec::X500NameAttribute](#) (p. 389).

**6.31.2.2** `virtual bool ArcSec::AttributeValue::equal ( AttributeValue * value, bool check_id = true ) [pure virtual]`

Evaluate whether "this" equals to the parameter value

Implemented in [ArcSec::AnyURIAttribute](#) (p. 49), [ArcSec::BooleanAttribute](#) (p. 62), [ArcSec::DateTimeAttribute](#) (p. 136), [ArcSec::TimeAttribute](#) (p. 322), [ArcSec::DateAttribute](#) (p. 135), [ArcSec::DurationAttribute](#) (p. 147), [ArcSec::PeriodAttribute](#) (p. 245), [ArcSec::GenericAttribute](#) (p. 168), [ArcSec::StringAttribute](#) (p. 307), and [ArcSec::X500NameAttribute](#) (p. 389).

**6.31.2.3** `virtual std::string ArcSec::AttributeValue::getId ( ) [pure virtual]`

Get the AttributeId of the <Attribute>

Implemented in [ArcSec::AnyURIAttribute](#) (p. 49), [ArcSec::BooleanAttribute](#) (p. 62), [ArcSec::DateTimeAttribute](#) (p. 136), [ArcSec::TimeAttribute](#) (p. 322), [ArcSec::DateAttribute](#) (p. 135), [ArcSec::DurationAttribute](#) (p. 147), [ArcSec::PeriodAttribute](#) (p. 246), [ArcSec::GenericAttribute](#) (p. 168), [ArcSec::StringAttribute](#) (p. 308), and [ArcSec::X500NameAttribute](#) (p. 389).

**6.31.2.4** `virtual std::string ArcSec::AttributeValue::getType ( ) [pure virtual]`

Get the DataType of the <Attribute>

Implemented in [ArcSec::AnyURIAttribute](#) (p. 49), [ArcSec::BooleanAttribute](#) (p. 62), [ArcSec::DateTimeAttribute](#) (p. 136), [ArcSec::TimeAttribute](#) (p. 322), [ArcSec::DateAttribute](#) (p. 135), [ArcSec::DurationAttribute](#) (p. 147), [ArcSec::PeriodAttribute](#) (p. 246), [ArcSec::GenericAttribute](#) (p. 168), [ArcSec::StringAttribute](#) (p. 308), and [ArcSec::X500NameAttribute](#) (p. 389).

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

- AttributeValue.h

## 6.32 ArcSec::Attrs Class Reference

[Attrs](#) (p. 58) is a container for one or more [Attr](#) (p. 52).

```
#include <Request.h>
```

### 6.32.1 Detailed Description

[Attrs](#) (p. 58) is a container for one or more [Attr](#) (p. 52). [Attrs](#) (p. 58) includes includes methods for inserting, getting items, and counting size as well

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

- Request.h

## 6.33 ArcSec::AuthzRequest Struct Reference

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

- PDP.h

## 6.34 ArcSec::AuthzRequestSection Struct Reference

```
#include <PDP.h>
```

### 6.34.1 Detailed Description

These structure are based on the request schema for **PDP** (p. 242), so far it can apply to the ArcPDP's request schema, see src/hed/pdc/Request.xsd and src/hed/pdc/Request.xml. It could also apply to the XACMLPDP's request schema, since the difference is minor.

Another approach is, the service composes/marshalls the xml structure directly, then the service should use difference code to compose for ArcPDP's request schema and XACMLPDP's schema, which is not so good.

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

- PDP.h

## 6.35 Arc::AutoPointer< T > Class Template Reference

Wrapper for pointer with automatic destruction.

```
#include <Utils.h>
```

### Public Member Functions

- **AutoPointer** (void)
- **AutoPointer** (T \*o)
- **~AutoPointer** (void)
- **T & operator\*** (void) const
- **T \* operator->** (void) const
- **operator bool** (void) const
- **bool operator!** (void) const
- **operator T \*** (void) const

### 6.35.1 Detailed Description

```
template<typename T> class Arc::AutoPointer< T >
```

Wrapper for pointer with automatic destruction. If ordinary pointer is wrapped in instance of this class it will be automatically destroyed when instance is destroyed. This is useful for maintaing pointers in scope of one function. Only pointers returned by new() are supported.

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

- Utils.h

## 6.36 Arc::Base64 Class Reference

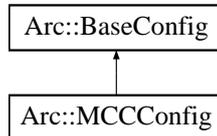
The documentation for this class was generated from the following file:

- Base64.h

## 6.37 Arc::BaseConfig Class Reference

```
#include <ArcConfig.h>
```

Inheritance diagram for Arc::BaseConfig:



### Public Member Functions

- void **AddPluginsPath** (const std::string &path)
- void **AddPrivateKey** (const std::string &path)
- void **AddCertificate** (const std::string &path)
- void **AddProxy** (const std::string &path)
- void **AddCAFile** (const std::string &path)
- void **AddCADir** (const std::string &path)
- void **AddOverlay** (XMLNode cfg)
- void **GetOverlay** (std::string fname)
- virtual XMLNode **MakeConfig** (XMLNode cfg) const

### 6.37.1 Detailed Description

Configuration for client interface. It contains information which can't be expressed in class constructor arguments. Most probably common things like software installation location, identity of user, etc.

### 6.37.2 Member Function Documentation

#### 6.37.2.1 void Arc::BaseConfig::AddCADir ( const std::string & *path* )

Add CA directory

#### 6.37.2.2 void Arc::BaseConfig::AddCAFile ( const std::string & *path* )

Add CA file

**6.37.2.3 void Arc::BaseConfig::AddCertificate ( const std::string & path )**

Add certificate

**6.37.2.4 void Arc::BaseConfig::AddOverlay ( XMLNode cfg )**

Add configuration overlay

**6.37.2.5 void Arc::BaseConfig::AddPluginsPath ( const std::string & path )**

Adds non-standard location of plugins

**6.37.2.6 void Arc::BaseConfig::AddPrivateKey ( const std::string & path )**

Add private key

**6.37.2.7 void Arc::BaseConfig::AddProxy ( const std::string & path )**

Add credentials proxy

**6.37.2.8 void Arc::BaseConfig::GetOverlay ( std::string fname )**

Read overlay from file

**6.37.2.9 virtual XMLNode Arc::BaseConfig::MakeConfig ( XMLNode cfg ) const [virtual]**

Adds configuration part corresponding to stored information into common configuration tree supplied in 'cfg' argument. Returns reference to XML node representing configuration of **ModuleManager** (p. 218)

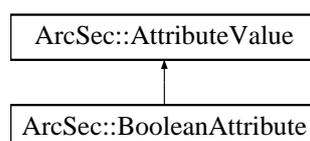
Reimplemented in **Arc::MCCConfig** (p. 207).

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

- ArcConfig.h

## 6.38 ArcSec::BooleanAttribute Class Reference

Inheritance diagram for ArcSec::BooleanAttribute:



## Public Member Functions

- virtual bool **equal** (**AttributeValue** \*o, bool check\_id=true)
- virtual std::string **encode** ()
- std::string **getId** ()
- std::string **getType** ()

### 6.38.1 Member Function Documentation

#### 6.38.1.1 virtual std::string ArcSec::BooleanAttribute::encode ( ) [**inline**, **virtual**]

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 57).

#### 6.38.1.2 virtual bool ArcSec::BooleanAttribute::equal ( **AttributeValue** \* *value*, bool *check\_id* = **true** ) [**virtual**]

Evaluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 58).

#### 6.38.1.3 std::string ArcSec::BooleanAttribute::getId ( ) [**inline**, **virtual**]

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 58).

#### 6.38.1.4 std::string ArcSec::BooleanAttribute::getType ( ) [**inline**, **virtual**]

Get the DataType of the <Attribute>

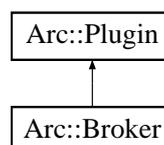
Implements **ArcSec::AttributeValue** (p. 58).

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

- BooleanAttribute.h

## 6.39 Arc::Broker Class Reference

Inheritance diagram for Arc::Broker:



## Public Member Functions

- const **ExecutionTarget** \* **GetBestTarget** ()
- void **PreFilterTargets** (std::list< **ExecutionTarget** > &targets, const **JobDescription** &job)
- void **RegisterJobsubmission** ()

## Protected Member Functions

- virtual void **SortTargets** ()=0

## Protected Attributes

- std::list< **ExecutionTarget** \* > **PossibleTargets**
- bool **TargetSortingDone**

### 6.39.1 Member Function Documentation

#### 6.39.1.1 const **ExecutionTarget**\* **Arc::Broker::GetBestTarget** ( )

Returns next target from the list of **ExecutionTarget** (p. 154) objects.

When first called this method will sort its list of **ExecutionTarget** (p. 154) objects, which have been filled by the **PreFilterTargets** method, and then the first target in the list will be returned.

If this is not the first call then the next target in the list is simply returned.

If there are no targets in the list or the end of the target list have been reached the NULL pointer is returned.

#### Returns

The pointer to the next **ExecutionTarget** (p. 154) in the list is returned.

#### 6.39.1.2 void **Arc::Broker::PreFilterTargets** ( std::list< **ExecutionTarget** > & *targets*, const **JobDescription** & *job* )

**ExecutionTarget** (p. 154) filtering, view-point: enough memory, disk space, CPUs, etc.

The "bad" targets will be ignored and only the good targets will be added to to the list of **ExecutionTarget** (p. 154) objects which be used for brokering.

#### Parameters

- targets* A list of **ExecutionTarget** (p. 154) objects to be considered for addition to the **Broker** (p. 62).
- jd* **JobDescription** (p. 188) object of the actual job.

#### 6.39.1.3 virtual void **Arc::Broker::SortTargets** ( ) [protected, pure virtual]

Custom Brokers should implement this method.

The task is to sort the **PossibleTargets** list by "custom" way, for example: **FastestQueueBroker**, **ExecutionTarget** (p. 154) which has the shortest queue length will be at the beginning of the **PossibleTargets** list

## 6.39.2 Field Documentation

### 6.39.2.1 `std::list<ExecutionTarget*> Arc::Broker::PossibleTargets` [protected]

This content the Prefilteres ExecutionTargets.

If an Execution Tartget has enough memory, CPU, diskspace, etc. for the actual job requirement than it will be added to the PossibleTargets list

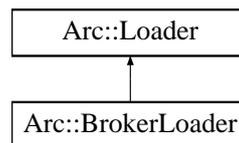
The documentation for this class was generated from the following file:

- Broker.h

## 6.40 Arc::BrokerLoader Class Reference

```
#include <Broker.h>
```

Inheritance diagram for Arc::BrokerLoader:



### Public Member Functions

- **BrokerLoader** ()
- **~BrokerLoader** ()
- **Broker \* load** (const std::string &name, const **UserConfig** &usercfg)
- const std::list< **Broker \* > & GetBrokers** () const

### 6.40.1 Detailed Description

Class responsible for loading **Broker** (p. 62) plugins The **Broker** (p. 62) objects returned by a **BrokerLoader** (p. 64) must not be used after the **BrokerLoader** (p. 64) goes out of scope.

### 6.40.2 Constructor & Destructor Documentation

#### 6.40.2.1 `Arc::BrokerLoader::BrokerLoader ( )`

Constructor Creates a new **BrokerLoader** (p. 64).

#### 6.40.2.2 `Arc::BrokerLoader::~~BrokerLoader ( )`

Destructor Calling the destructor destroys all Brokers loaded by the **BrokerLoader** (p. 64) instance.

### 6.40.3 Member Function Documentation

#### 6.40.3.1 `const std::list<Broker*> & Arc::BrokerLoader::GetBrokers ( ) const [inline]`

Retrieve the list of loaded Brokers.

##### Returns

A reference to the list of Brokers.

#### 6.40.3.2 `Broker* Arc::BrokerLoader::load ( const std::string & name, const UserConfig & usercfg )`

Load a new **Broker** (p. 62)

##### Parameters

*name* The name of the **Broker** (p. 62) to load.

*usercfg* The **UserConfig** (p. 333) object for the new **Broker** (p. 62).

##### Returns

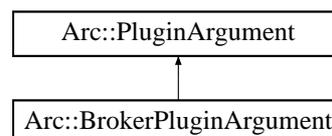
A pointer to the new **Broker** (p. 62) (NULL on error).

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

- Broker.h

## 6.41 Arc::BrokerPluginArgument Class Reference

Inheritance diagram for Arc::BrokerPluginArgument:



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

- Broker.h

## 6.42 Arc::ByteArray Class Reference

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

- ByteArray.h

## 6.43 Arc::CacheParameters Struct Reference

```
#include <FileCache.h>
```

### 6.43.1 Detailed Description

Contains data on the parameters of a cache.

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

- FileCache.h

## 6.44 ArcCredential::cert\_verify\_context Struct Reference

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

- CertUtil.h

## 6.45 Arc::ChainContext Class Reference

Interface to chain specific functionality.

```
#include <MCCLoader.h>
```

### Public Member Functions

- `operator PluginsFactory * ()`

### 6.45.1 Detailed Description

Interface to chain specific functionality. Object of this class is associated with every **MCCLoader** (p. 208) object. It is accessible for **MCC** (p. 202) and **Service** (p. 285) components and provides an interface to manipulate chains stored in **Loader** (p. 191). This makes it possible to modify chains dynamically - like deploying new services on demand.

### 6.45.2 Member Function Documentation

#### 6.45.2.1 Arc::ChainContext::operator PluginsFactory \* ( ) [inline]

Returns associated **PluginsFactory** (p. 252) object

References Arc::Loader::factory\_.

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

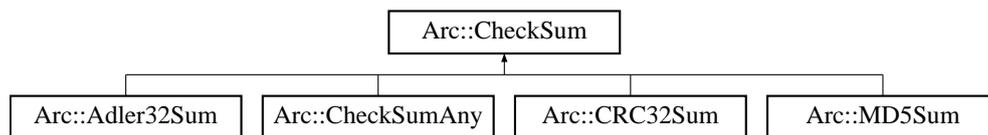
- MCCLoader.h

## 6.46 Arc::Checksum Class Reference

Defines interface for variuos checksum manipulations.

```
#include <Checksum.h>
```

Inheritance diagram for Arc::Checksum:



### 6.46.1 Detailed Description

Defines interface for variuos checksum manipulations. This class is used during data transfers through **DataBuffer** (p. 100) class

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

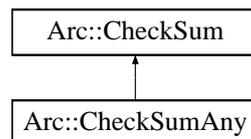
- CheckSum.h

## 6.47 Arc::ChecksumAny Class Reference

Wraper for **Checksum** (p. 67) class.

```
#include <Checksum.h>
```

Inheritance diagram for Arc::ChecksumAny:



### 6.47.1 Detailed Description

Wraper for **Checksum** (p. 67) class. To be used for manipulation of any supported checksum type in a transparent way.

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

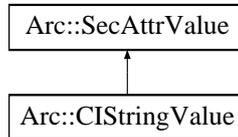
- CheckSum.h

## 6.48 Arc::CStringValue Class Reference

This class implements case insensitive strings as security attributes.

```
#include <CIStringValue.h>
```

Inheritance diagram for Arc::CIStringValue:



## Public Member Functions

- `CIStringValue ()`
- `CIStringValue (const char *ss)`
- `CIStringValue (const std::string &ss)`
- virtual `operator bool ()`

## Protected Member Functions

- virtual `bool equal (SecAttrValue &b)`

### 6.48.1 Detailed Description

This class implements case insensitive strings as security attributes. This is an example of how to inherit `SecAttrValue` (p. 282). The class is meant to implement security attributes that are case insensitive strings.

### 6.48.2 Constructor & Destructor Documentation

#### 6.48.2.1 Arc::CIStringValue::CIStringValue ( )

Default constructor

#### 6.48.2.2 Arc::CIStringValue::CIStringValue ( const char \* ss )

This is a constructor that takes a string literal.

#### 6.48.2.3 Arc::CIStringValue::CIStringValue ( const std::string & ss )

This is a constructor that takes a string object.

### 6.48.3 Member Function Documentation

#### 6.48.3.1 virtual bool Arc::CIStringValue::equal ( SecAttrValue & b ) [protected, virtual]

This function returns true if two strings are the same apart from letter case

Reimplemented from `Arc::SecAttrValue` (p. 282).

### 6.48.3.2 virtual Arc::CStringValue::operator bool ( ) [virtual]

This function returns false if the string is empty or uninitialized

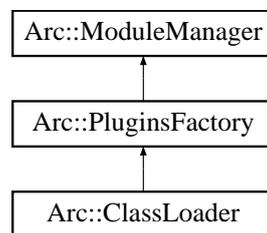
Reimplemented from `Arc::SecAttrValue` (p. 282).

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

- CStringValue.h

## 6.49 Arc::ClassLoader Class Reference

Inheritance diagram for Arc::ClassLoader:

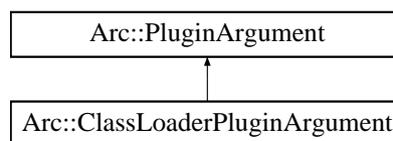


The documentation for this class was generated from the following file:

- ClassLoader.h

## 6.50 Arc::ClassLoaderPluginArgument Class Reference

Inheritance diagram for Arc::ClassLoaderPluginArgument:



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

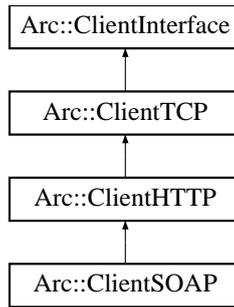
- ClassLoader.h

## 6.51 Arc::ClientHTTP Class Reference

Class for setting up a `MCC` (p. 202) chain for HTTP communication.

```
#include <ClientInterface.h>
```

Inheritance diagram for Arc::ClientHTTP:



### 6.51.1 Detailed Description

Class for setting up a **MCC** (p. 202) chain for HTTP communication. The **ClientHTTP** (p. 69) class inherits from the **ClientTCP** (p. 73) class and adds an HTTP **MCC** (p. 202) to the chain.

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

- ClientInterface.h

## 6.52 Arc::ClientHTTPwithSAML2SSO Class Reference

### Public Member Functions

- **ClientHTTPwithSAML2SSO** ()
- **MCC\_Status process** (const std::string &method, **PayloadRawInterface** \*request, **HTTPClientInfo** \*info, **PayloadRawInterface** \*\*response, const std::string &idp\_name, const std::string &username, const std::string &password, const bool reuse\_authn=false)

### 6.52.1 Constructor & Destructor Documentation

#### 6.52.1.1 Arc::ClientHTTPwithSAML2SSO::ClientHTTPwithSAML2SSO ( ) [inline]

Constructor creates **MCC** (p. 202) chain and connects to server.

### 6.52.2 Member Function Documentation

#### 6.52.2.1 MCC\_Status Arc::ClientHTTPwithSAML2SSO::process ( const std::string & *method*, **PayloadRawInterface** \* *request*, **HTTPClientInfo** \* *info*, **PayloadRawInterface** \*\* *response*, const std::string & *idp\_name*, const std::string & *username*, const std::string & *password*, const bool *reuse\_authn* = *false* )

Send HTTP request and receive response.

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

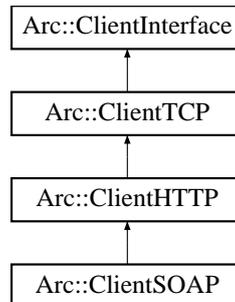
- ClientSAML2SSO.h

## 6.53 Arc::ClientInterface Class Reference

Utility base class for **MCC** (p. 202).

```
#include <ClientInterface.h>
```

Inheritance diagram for Arc::ClientInterface:



### 6.53.1 Detailed Description

Utility base class for **MCC** (p. 202). The **ClientInterface** (p. 71) class is a utility base class used for configuring a client side **Message** (p. 210) Chain Component (**MCC** (p. 202)) chain and loading it into memory. It has several specializations of increasing complexity of the **MCC** (p. 202) chains.

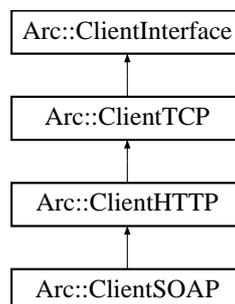
The documentation for this class was generated from the following file:

- ClientInterface.h

## 6.54 Arc::ClientSOAP Class Reference

```
#include <ClientInterface.h>
```

Inheritance diagram for Arc::ClientSOAP:



### Public Member Functions

- **ClientSOAP** ()
- **MCC\_Status process** (**PayloadSOAP** \*request, **PayloadSOAP** \*\*response)

- **MCC\_Status process** (const std::string &action, **PayloadSOAP** \*request, **PayloadSOAP** \*\*response)
- **MCC \* GetEntry** ()
- void **AddSecHandler** (XMLNode handlercfg, const std::string &libanme="", const std::string &libpath="")
- virtual bool **Load** ()

### 6.54.1 Detailed Description

Class with easy interface for sending/receiving SOAP messages over HTTP(S/G). It takes care of configuring MCC (p. 202) chain and making an entry point.

### 6.54.2 Constructor & Destructor Documentation

#### 6.54.2.1 Arc::ClientSOAP::ClientSOAP ( ) [inline]

Constructor creates MCC (p. 202) chain and connects to server.

### 6.54.3 Member Function Documentation

#### 6.54.3.1 void Arc::ClientSOAP::AddSecHandler ( XMLNode handlercfg, const std::string &libanme = "", const std::string &libpath = "" )

Adds security handler to configuration of SOAP MCC (p. 202)

Reimplemented from **Arc::ClientHTTP** (p. 69).

#### 6.54.3.2 MCC\* Arc::ClientSOAP::GetEntry ( ) [inline]

Returns entry point to SOAP MCC (p. 202) in configured chain. To initialize entry point **Load**() (p. 72) method must be called.

Reimplemented from **Arc::ClientHTTP** (p. 69).

#### 6.54.3.3 virtual bool Arc::ClientSOAP::Load ( ) [virtual]

Instantiates pluggable elements according to generated configuration

Reimplemented from **Arc::ClientHTTP** (p. 69).

#### 6.54.3.4 MCC\_Status Arc::ClientSOAP::process ( PayloadSOAP \* request, PayloadSOAP \*\* response )

Send SOAP request and receive response.

#### 6.54.3.5 MCC\_Status Arc::ClientSOAP::process ( const std::string & action, PayloadSOAP \* request, PayloadSOAP \*\* response )

Send SOAP request with specified SOAP action and receive response.

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

- ClientInterface.h

## 6.55 Arc::ClientSOAPwithSAML2SSO Class Reference

### Public Member Functions

- **ClientSOAPwithSAML2SSO** ()
- **MCC\_Status process** (**PayloadSOAP** \*request, **PayloadSOAP** \*\*response, const std::string &idp\_name, const std::string &username, const std::string &password, const bool reuse\_authn=false)
- **MCC\_Status process** (const std::string &action, **PayloadSOAP** \*request, **PayloadSOAP** \*\*response, const std::string &idp\_name, const std::string &username, const std::string &password, const bool reuse\_authn=false)

### 6.55.1 Constructor & Destructor Documentation

#### 6.55.1.1 Arc::ClientSOAPwithSAML2SSO::ClientSOAPwithSAML2SSO ( ) [**inline**]

Constructor creates MCC (p. 202) chain and connects to server.

### 6.55.2 Member Function Documentation

#### 6.55.2.1 MCC\_Status Arc::ClientSOAPwithSAML2SSO::process ( **PayloadSOAP** \* request, **PayloadSOAP** \*\* response, const std::string & idp\_name, const std::string & username, const std::string & password, const bool reuse\_authn = *false* )

Send SOAP request and receive response.

#### 6.55.2.2 MCC\_Status Arc::ClientSOAPwithSAML2SSO::process ( const std::string & action, **PayloadSOAP** \* request, **PayloadSOAP** \*\* response, const std::string & idp\_name, const std::string & username, const std::string & password, const bool reuse\_authn = *false* )

Send SOAP request with specified SOAP action and receive response.

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

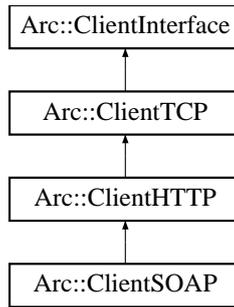
- ClientSAML2SSO.h

## 6.56 Arc::ClientTCP Class Reference

Class for setting up a MCC (p. 202) chain for TCP communication.

```
#include <ClientInterface.h>
```

Inheritance diagram for Arc::ClientTCP:



### 6.56.1 Detailed Description

Class for setting up a **MCC** (p.202) chain for TCP communication. The **ClientTCP** (p.73) class is a specialization of the **ClientInterface** (p.71) which sets up a client **MCC** (p.202) chain for TCP communication, and optionally with a security layer on top which can be either TLS, GSI or SSL3.

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

- ClientInterface.h

## 6.57 Arc::ClientX509Delegation Class Reference

### Public Member Functions

- **ClientX509Delegation** ()
- bool **createDelegation** (DelegationType deleg, std::string &delegation\_id)
- bool **acquireDelegation** (DelegationType deleg, std::string &delegation\_cred, std::string &delegation\_id, const std::string cred\_identity="", const std::string cred\_delegator\_ip="", const std::string username="", const std::string password="")

### 6.57.1 Constructor & Destructor Documentation

#### 6.57.1.1 Arc::ClientX509Delegation::ClientX509Delegation ( ) [inline]

Constructor creates **MCC** (p.202) chain and connects to server.

### 6.57.2 Member Function Documentation

#### 6.57.2.1 bool Arc::ClientX509Delegation::acquireDelegation ( DelegationType *deleg*, std::string & *delegation\_cred*, std::string & *delegation\_id*, const std::string *cred\_identity* = "", const std::string *cred\_delegator\_ip* = "", const std::string *username* = "", const std::string *password* = "" )

Acquire delegation credential from delegation service. This method should be called by intermediate service ('n+1' service as explained on above) in order to use this delegation credential on behalf of the EEC's holder.

#### Parameters

*deleg* Delegation type

*delegation\_id* delegation ID which is used to look up the credential by delegation service

*cred\_identity* the identity (in case of x509 credential, it is the DN of EEC credential).

*cred\_delegator\_ip* the IP address of the credential delegator. Regard of delegation, an intermediate service should accomplish three tasks: 1. Acquire 'n' level delegation credential (which is delegated by 'n-1' level delegator) from delegation service; 1. Create 'n+1' level delegation credential to delegation service; 2. Use 'n' level delegation credential to act on behalf of the EEC's holder. In case of absense of *delegation\_id*, the 'n-1' level delegator's IP address and credential's identity are supposed to be used for look up the delegation credential from delegation service.

### 6.57.2.2 bool Arc::ClientX509Delegation::createDelegation ( DelegationType deleg, std::string & delegation\_id )

Create the delegation credential according to the different remote delegation service. This method should be called by holder of EEC(end entity credential) which would delegate its EEC credential, or by holder of delegated credential(normally, the holder is intermediate service) which would further delegate the credential (on behalf of the original EEC's holder) (for instance, the 'n' intermediate service creates a delegation credential, then the 'n+1' intermediate service acquires this delegation credential from the delegation service and also acts on behalf of the EEC's holder by using this delegation credential).

#### Parameters

*deleg* Delegation type

*delegation\_id* For gridsite delegation service, the *delegation\_id* is supposed to be created by client side, and sent to service side; for ARC delegation service, the *delegation\_id* is supposed to be created by service side, and returned back. So for gridsite delegation service, this parameter is treated as input, while for ARC delegation service, it is treated as output.

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

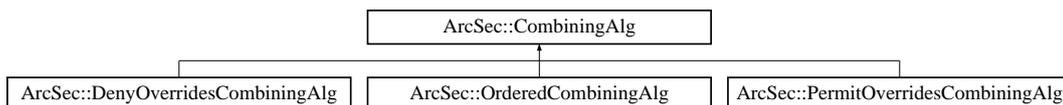
- ClientX509Delegation.h

## 6.58 ArcSec::CombiningAlg Class Reference

Interface for combining algrithm.

```
#include <CombiningAlg.h>
```

Inheritance diagram for ArcSec::CombiningAlg:



### Public Member Functions

- virtual Result **combine** (EvaluationCtx \*ctx, std::list< Policy \* > policies)=0
- virtual const std::string & **getalgId** (void) const =0

### 6.58.1 Detailed Description

Interface for combining algorithm. This class is used to implement a specific combining algorithm for combining policies.

### 6.58.2 Member Function Documentation

#### 6.58.2.1 virtual Result ArcSec::CombiningAlg::combine ( EvaluationCtx \* *ctx*, std::list< Policy \* > *policies* ) [pure virtual]

Evaluate request against policy, and if there are more than one policies, combine the evaluation results according to the combing algorithm implemented inside in the method combine(ctx, policies) itself.

#### Parameters

*ctx* The information about request is included

*policies* The "match" and "eval" method inside each policy will be called, and then those results from each policy will be combined according to the combining algorithm inside CombiningAlg class.

Implemented in **ArcSec::DenyOverridesCombiningAlg** (p. 145), and **ArcSec::PermitOverridesCombiningAlg** (p. 246).

#### 6.58.2.2 virtual const std::string& ArcSec::CombiningAlg::getalgId ( void ) const [pure virtual]

Get the identifier of the combining algorithm class

#### Returns

The identity of the algorithm

Implemented in **ArcSec::DenyOverridesCombiningAlg** (p. 145), and **ArcSec::PermitOverridesCombiningAlg** (p. 247).

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

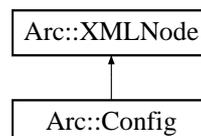
- CombiningAlg.h

## 6.59 Arc::Config Class Reference

Configuration element - represents (sub)tree of ARC configuration.

```
#include <ArcConfig.h>
```

Inheritance diagram for Arc::Config:



## Public Member Functions

- **Config** ()
- **Config** (const char \*filename)
- **Config** (const std::string &xml\_str)
- **Config** (XMLNode xml)
- **Config** (long cfg\_ptr\_addr)
- **Config** (const **Config** &cfg)
- void **print** (void)
- void **parse** (const char \*filename)
- const std::string & **getFileName** (void) const
- void **setFileName** (const std::string &filename)
- void **save** (const char \*filename)

### 6.59.1 Detailed Description

Configuration element - represents (sub)tree of ARC configuration. This class is intended to be used to pass configuration details to various parts of HED and external modules. Currently it's just a wrapper over XML tree. But than may change in a future, although interface should be preserved. Currently it is capable of loading XML configuration document from file. In future it will be capable of loading more user-readable format and process it into tree-like structure convenient for machine processing (XML-like). So far there are no schema and/or namespaces assigned.

### 6.59.2 Constructor & Destructor Documentation

#### 6.59.2.1 Arc::Config::Config ( ) [inline]

Creates empty XML tree

#### 6.59.2.2 Arc::Config::Config ( const char \* filename )

Loads configuration document from file 'filename'

#### 6.59.2.3 Arc::Config::Config ( const std::string & xml\_str ) [inline]

Parse configuration document from memory

#### 6.59.2.4 Arc::Config::Config ( XMLNode xml ) [inline]

Acquire existing XML (sub)tree. Content is not copied. Make sure XML tree is not destroyed while in use by this object.

#### 6.59.2.5 Arc::Config::Config ( long cfg\_ptr\_addr )

Copy constructor used by language bindings

### 6.59.2.6 Arc::Config::Config ( const Config & *cfg* )

Copy constructor used by language bindings

## 6.59.3 Member Function Documentation

### 6.59.3.1 const std::string& Arc::Config::getFileName ( void ) const [inline]

Gives back file name of config file or empty string if it was generated from the **XMLNode** (p. 392) subtree

### 6.59.3.2 void Arc::Config::parse ( const char \* *filename* )

Parse configuration document from file 'filename'

### 6.59.3.3 void Arc::Config::print ( void )

Print structure of document. For debugging purposes. Printed content is not an XML document.

### 6.59.3.4 void Arc::Config::save ( const char \* *filename* )

Save to file

### 6.59.3.5 void Arc::Config::setFileName ( const std::string & *filename* ) [inline]

Set the file name of config file

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

- ArcConfig.h

## 6.60 Arc::ConfusaCertHandler Class Reference

```
#include <ConfusaCertHandler.h>
```

### Public Member Functions

- **ConfusaCertHandler** (int keysize, const std::string dn)
- std::string **getCertRequestB64** ()
- bool **createCertRequest** (std::string password="", std::string storedir="./")

### 6.60.1 Detailed Description

Wrapper around **Credential** (p. 90) handling the Confusa specifics.

## 6.60.2 Constructor & Destructor Documentation

### 6.60.2.1 Arc::ConfusaCertHandler::ConfusaCertHandler ( int *keysize*, const std::string *dn* )

Create a new **ConfusaCertHandler** (p. 78) for DN *dn* and given *keysize*. Basically Confusa cert handler wraps around **Credential** (p. 90)

## 6.60.3 Member Function Documentation

### 6.60.3.1 bool Arc::ConfusaCertHandler::createCertRequest ( std::string *password* = "", std::string *storedir* = ". /" )

Create a new end entity certificate, with a private key encrypted with password *password*. Private key and certificate will be stored in directory *storedir*.

### 6.60.3.2 std::string Arc::ConfusaCertHandler::getCertRequestB64 ( )

Get the certificate request managed by this confusa cert handler in base 64 encoding

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

- ConfusaCertHandler.h

## 6.61 Arc::ConfusaParserUtils Class Reference

```
#include <ConfusaParserUtils.h>
```

### Static Public Member Functions

- static std::string **urlencode** (const std::string *url*)
- static std::string **urlencode\_params** (const std::string *url*)
- static xmlDocPtr **get\_doc** (const std::string *xml\_file*)
- static void **destroy\_doc** (xmlDocPtr *doc*)
- static std::string **extract\_body\_information** (const std::string *html\_string*)
- static std::string **handle\_redirect\_step** (Arc::MCCConfig *cfg*, const std::string *remote\_url*, std::string *\*cookies*=NULL, std::multimap< std::string, std::string > *\*httpAttributes*=NULL)
- static std::string **evaluate\_path** (xmlDocPtr *doc*, const std::string *xpathExpr*, std::list< std::string > *\*contentList*=NULL)

### 6.61.1 Detailed Description

Methods often needed in evaluation web pages from the Confusa WebSSO workflow

### 6.61.2 Member Function Documentation

#### 6.61.2.1 static void Arc::ConfusaParserUtils::destroy\_doc ( xmlDocPtr *doc* ) [static]

Destroy a libxml2 doc representation

**6.61.2.2** `static std::string Arc::ConfusaParserUtils::evaluate_path ( xmlDocPtr doc, const std::string xpathExpr, std::list< std::string > * contentList = NULL ) [static]`

Evaluate the given xpathExpr on the document ptr. Return a string with the FIRST result if contentList is NULL. Return a string with the first result and all results, including the first one, in contentList if contentList is not null.

**6.61.2.3** `static std::string Arc::ConfusaParserUtils::extract_body_information ( const std::string html_string ) [static]`

Get the part only within <body> and </body> in a HTML string For parsing, usually only this part is interesting.

**6.61.2.4** `static xmlDocPtr Arc::ConfusaParserUtils::get_doc ( const std::string xml_file ) [static]`

Construct a lixml2 doc representation from the xml file

**6.61.2.5** `static std::string Arc::ConfusaParserUtils::handle_redirect_step ( Arc::MCCCConfig cfg, const std::string remote_url, std::string * cookies = NULL, std::multimap< std::string, std::string > * httpAttributes = NULL ) [static]`

Handle a single redirect step from the SAML2 WebSSO profile. Store the received cookie in \*cookie and pass the given httpAttributes to the site during redirect.

**6.61.2.6** `static std::string Arc::ConfusaParserUtils::urlencode ( const std::string url ) [static]`

urlencode the passed string

**6.61.2.7** `static std::string Arc::ConfusaParserUtils::urlencode_params ( const std::string url ) [static]`

Urlencode the passed string with respect to the parameters. The difference to urlencode is that the parameters will keep their separators, i.e. the ? and & separating parameters will be preserved.

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

- ConfusaParserUtils.h

## 6.62 Arc::CountedPointer< T > Class Template Reference

Wrapper for pointer with automatic destruction and mutiple references.

```
#include <Utils.h>
```

### Data Structures

- class Base

## Public Member Functions

- `T & operator*` (void) const
- `T * operator->` (void) const
- `operator bool` (void) const
- `bool operator!` (void) const
- `operator T *` (void) const

### 6.62.1 Detailed Description

`template<typename T> class Arc::CountedPointer< T >`

Wrapper for pointer with automatic destruction and mutiple references. If ordinary pointer is wrapped in instance of this class it will be automatically destroyed when all instances refering to it are destroyed. This is useful for maintaing pointers refered from multiple structures wihth automatic destruction of original object when last reference is destroyed. It is similar to Java approach with a difference that destruction time is strictly defined. Only pointers returned by `new()` are supported. This class is not thread-safe

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

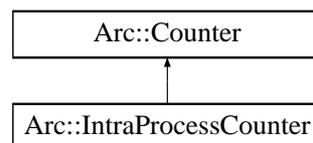
- `Utils.h`

## 6.63 Arc::Counter Class Reference

A class defining a common interface for counters.

```
#include <Counter.h>
```

Inheritance diagram for `Arc::Counter`:



## Public Member Functions

- virtual `~Counter` ()
- virtual int `getLimit` ()=0
- virtual int `setLimit` (int newLimit)=0
- virtual int `changeLimit` (int amount)=0
- virtual int `getExcess` ()=0
- virtual int `setExcess` (int newExcess)=0
- virtual int `changeExcess` (int amount)=0
- virtual int `getValue` ()=0
- virtual `CounterTicket reserve` (int amount=1, Glib::TimeVal duration=**ETERNAL**, bool prioritized=false, Glib::TimeVal timeOut=**ETERNAL**)=0

## Protected Types

- typedef unsigned long long int **IDType**

## Protected Member Functions

- **Counter** ()
- virtual void **cancel** (**IDType** reservationID)=0
- virtual void **extend** (**IDType** &reservationID, Glib::TimeVal &expiryTime, Glib::TimeVal duration=**ETERNAL**)=0
- Glib::TimeVal **getCurrentTime** ()
- Glib::TimeVal **getExpiryTime** (Glib::TimeVal duration)
- **CounterTicket** **getCounterTicket** (**Counter::IDType** reservationID, Glib::TimeVal expiryTime, **Counter** \*counter)
- **ExpirationReminder** **getExpirationReminder** (Glib::TimeVal expTime, **Counter::IDType** resID)

## Friends

- class **CounterTicket**
- class **ExpirationReminder**

### 6.63.1 Detailed Description

A class defining a common interface for counters. This class defines a common interface for counters as well as some common functionality.

The purpose of a counter is to provide housekeeping some resource such as e.g. disk space, memory or network bandwidth. The counter itself will not be aware of what kind of resource it limits the use of. Neither will it be aware of what unit is being used to measure that resource. Counters are thus very similar to semaphores. Furthermore, counters are designed to handle concurrent operations from multiple threads/processes in a consistent manner.

Every counter has a limit, an excess limit and a value. The limit is a number that specify how many units are available for reservation. The value is the number of units that are currently available for reservation, i.e. has not allready been reserved. The excess limit specify how many extra units can be reserved for high priority needs even if there are no normal units available for reservation. The excess limit is similar to the credit limit of e.g. a VISA card.

The users of the resource must thus first call the counter in order to make a reservation of an appropriate amount of the resource, then allocate and use the resource and finally call the counter again to cancel the reservation.

Typical usage is:

```
// Declare a counter. Replace XYZ by some appropriate kind of
// counter and provide required parameters. Unit is MB.
XYZCounter memory(...);
...
// Make a reservation of memory for 2000000 doubles.
CounterTicket tick = memory.reserve(2*sizeof(double));
// Use the memory.
double* A=new double[2000000];
doSomething(A);
delete[] A;
// Cancel the reservation.
tick.cancel();
```

There are also alternative ways to make reservations, including self-expiring reservations, prioritized reservations and reservations that fail if they cannot be made fast enough.

For self expiring reservations, a duration is provided in the reserve call:

```
tick = memory.reserve(2*sizeof(double), Glib::TimeVal(1,0));
```

A self-expiring reservation can be cancelled explicitly before it expires, but if it is not cancelled it will expire automatically when the duration has passed. The default value for the duration is ETERNAL, which means that the reservation will not be cancelled automatically.

Prioritized reservations may use the excess limit and succeed immediately even if there are no normal units available for reservation. The value of the counter will in this case become negative. A prioritized reservation looks like this:

```
tick = memory.reserve(2*sizeof(double), Glib::TimeVal(1,0), true);
```

Finally, a time out option can be provided for a reservation. If some task should be performed within two seconds or not at all, the reservation can look like this:

```
tick = memory.reserve(2*sizeof(double), Glib::TimeVal(1,0),
                    true, Glib::TimeVal(2,0));
if (tick.isValid())
    doSomething(...);
```

## 6.63.2 Member Typedef Documentation

### 6.63.2.1 typedef unsigned long long int Arc::Counter::IDType [protected]

A typedef of identification numbers for reservation.

This is a type that is used as identification numbers (keys) for referencing of reservations. It is used internally in counters for book keeping of reservations as well as in the **CounterTicket** (p. 88) class in order to be able to cancel and extend reservations.

## 6.63.3 Constructor & Destructor Documentation

### 6.63.3.1 Arc::Counter::Counter ( ) [protected]

Default constructor.

This is the default constructor. Since **Counter** (p. 81) is an abstract class, it should only be used by subclasses. Therefore it is protected. Furthermore, since the **Counter** (p. 81) class has no attributes, nothing needs to be initialized and thus this constructor is empty.

### 6.63.3.2 virtual Arc::Counter::~~Counter ( ) [virtual]

The destructor.

This is the destructor of the **Counter** (p. 81) class. Since the **Counter** (p. 81) class has no attributes, nothing needs to be cleaned up and thus the destructor is empty.

## 6.63.4 Member Function Documentation

### 6.63.4.1 `virtual void Arc::Counter::cancel ( IDType reservationID ) [protected, pure virtual]`

Cancellation of a reservation.

This method cancels a reservation. It is called by the **CounterTicket** (p. 88) that corresponds to the reservation.

#### Parameters

*reservationID* The identity number (key) of the reservation to cancel.

### 6.63.4.2 `virtual int Arc::Counter::changeExcess ( int amount ) [pure virtual]`

Changes the excess limit of the counter.

Changes the excess limit of the counter by adding a certain amount to the current excess limit.

#### Parameters

*amount* The amount by which to change the excess limit.

#### Returns

The new excess limit.

Implemented in **Arc::IntraProcessCounter** (p. 181).

### 6.63.4.3 `virtual int Arc::Counter::changeLimit ( int amount ) [pure virtual]`

Changes the limit of the counter.

Changes the limit of the counter by adding a certain amount to the current limit.

#### Parameters

*amount* The amount by which to change the limit.

#### Returns

The new limit.

Implemented in **Arc::IntraProcessCounter** (p. 181).

### 6.63.4.4 `virtual void Arc::Counter::extend ( IDType & reservationID, Glib::TimeVal & expiryTime, Glib::TimeVal duration = ETERNAL ) [protected, pure virtual]`

Extension of a reservation.

This method extends a reservation. It is called by the **CounterTicket** (p. 88) that corresponds to the reservation.

**Parameters**

*reservationID* Used for input as well as output. Contains the identification number of the original reservation on entry and the new identification number of the extended reservation on exit.

*expiryTime* Used for input as well as output. Contains the expiry time of the original reservation on entry and the new expiry time of the extended reservation on exit.

*duration* The time by which to extend the reservation. The new expiration time is computed based on the current time, NOT the previous expiration time.

**6.63.4.5 CounterTicket Arc::Counter::getCounterTicket ( Counter::IDType reservationID, Glib::TimeVal expiryTime, Counter \* counter ) [protected]**

A "relay method" for a constructor of the **CounterTicket** (p. 88) class.

This method acts as a relay for one of the constructors of the **CounterTicket** (p. 88) class. That constructor is private, but needs to be accessible from the subclasses of **Counter** (p. 81) (but not from anywhere else). In order not to have to declare every possible subclass of **Counter** (p. 81) as a friend of **CounterTicket** (p. 88), only the base class **Counter** (p. 81) is a friend and its subclasses access the constructor through this method. (If C++ had supported "package access", as Java does, this trick would not have been necessary.)

**Parameters**

*reservationID* The identity number of the reservation corresponding to the **CounterTicket** (p. 88).

*expiryTime* the expiry time of the reservation corresponding to the **CounterTicket** (p. 88).

*counter* The **Counter** (p. 81) from which the reservation has been made.

**Returns**

The counter ticket that has been created.

**6.63.4.6 Glib::TimeVal Arc::Counter::getCurrentTime ( ) [protected]**

Get the current time.

Returns the current time. An "adapter method" for the assign\_current\_time() method in the Glib::TimeVal class. return The current time.

**6.63.4.7 virtual int Arc::Counter::getExcess ( ) [pure virtual]**

Returns the excess limit of the counter.

Returns the excess limit of the counter, i.e. by how much the usual limit may be exceeded by prioritized reservations.

**Returns**

The excess limit.

Implemented in **Arc::IntraProcessCounter** (p. 182).

#### 6.63.4.8 **ExpirationReminder Arc::Counter::getExpirationReminder ( Glib::TimeVal *expTime*, Counter::IDType *resID* ) [protected]**

A "relay method" for the constructor of **ExpirationReminder** (p. 158).

This method acts as a relay for one of the constructors of the **ExpirationReminder** (p. 158) class. That constructor is private, but needs to be accessible from the subclasses of **Counter** (p. 81) (but not from anywhere else). In order not to have to declare every possible subclass of **Counter** (p. 81) as a friend of **ExpirationReminder** (p. 158), only the base class **Counter** (p. 81) is a friend and its subclasses access the constructor through this method. (If C++ had supported "package access", as Java does, this trick would not have been necessary.)

##### Parameters

*expTime* the expiry time of the reservation corresponding to the **ExpirationReminder** (p. 158).

*resID* The identity number of the reservation corresponding to the **ExpirationReminder** (p. 158).

##### Returns

The **ExpirationReminder** (p. 158) that has been created.

#### 6.63.4.9 **Glib::TimeVal Arc::Counter::getExpiryTime ( Glib::TimeVal *duration* ) [protected]**

Computes an expiry time.

This method computes an expiry time by adding a duration to the current time.

##### Parameters

*duration* The duration.

##### Returns

The expiry time.

#### 6.63.4.10 **virtual int Arc::Counter::getLimit ( ) [pure virtual]**

Returns the current limit of the counter.

This method returns the current limit of the counter, i.e. how many units can be reserved simultaneously by different threads without claiming high priority.

##### Returns

The current limit of the counter.

Implemented in **Arc::IntraProcessCounter** (p. 182).

#### 6.63.4.11 **virtual int Arc::Counter::getValue ( ) [pure virtual]**

Returns the current value of the counter.

Returns the current value of the counter, i.e. the number of unreserved units. Initially, the value is equal to the limit of the counter. When a reservation is made, the value is decreased. Normally, the value should

never be negative, but this may happen if there are prioritized reservations. It can also happen if the limit is decreased after some reservations have been made, since reservations are never revoked.

### Returns

The current value of the counter.

Implemented in **Arc::IntraProcessCounter** (p. 182).

**6.63.4.12** `virtual CounterTicket Arc::Counter::reserve ( int amount = 1, Glib::TimeVal duration = ETERNAL, bool prioritized = false, Glib::TimeVal timeOut = ETERNAL ) [pure virtual]`

Makes a reservation from the counter.

This method makes a reservation from the counter. If the current value of the counter is too low to allow for the reservation, the method blocks until the reservation is possible or times out.

### Parameters

*amount* The amount to reserve, default value is 1.

*duration* The duration of a self expiring reservation, default is that it lasts forever.

*prioritized* Whether this reservation is prioritized and thus allowed to use the excess limit.

*timeOut* The maximum time to block if the value of the counter is too low, default is to allow "eternal" blocking.

### Returns

A **CounterTicket** (p. 88) that can be queried about the status of the reservation as well as for cancellations and extensions.

Implemented in **Arc::IntraProcessCounter** (p. 183).

**6.63.4.13** `virtual int Arc::Counter::setExcess ( int newExcess ) [pure virtual]`

Sets the excess limit of the counter.

This method sets a new excess limit for the counter.

### Parameters

*newExcess* The new excess limit, an absolute number.

### Returns

The new excess limit.

Implemented in **Arc::IntraProcessCounter** (p. 183).

**6.63.4.14** `virtual int Arc::Counter::setLimit ( int newLimit ) [pure virtual]`

Sets the limit of the counter.

This method sets a new limit for the counter.

### Parameters

*newLimit* The new limit, an absolute number.

### Returns

The new limit.

Implemented in **Arc::IntraProcessCounter** (p. 183).

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

- Counter.h

## 6.64 Arc::CounterTicket Class Reference

A class for "tickets" that correspond to counter reservations.

```
#include <Counter.h>
```

### Public Member Functions

- **CounterTicket** ()
- **bool isValid** ()
- **void extend** (Glib::TimeVal duration)
- **void cancel** ()

### Friends

- class **Counter**

#### 6.64.1 Detailed Description

A class for "tickets" that correspond to counter reservations. This is a class for reservation tickets. When a reservation is made from a **Counter** (p. 81), a **ReservationTicket** is returned. This ticket can then be queried about the validity of a reservation. It can also be used for cancelation and extension of reservations.

Typical usage is:

```
// Declare a counter. Replace XYZ by some appropriate kind of
// counter and provide required parameters. Unit is MB.
XYZCounter memory(...);
...
// Make a reservation of memory for 2000000 doubles.
CounterTicket tick = memory.reserve(2*sizeof(double));
// Use the memory.
double* A=new double[2000000];
doSomething(A);
delete[] A;
// Cancel the reservation.
tick.cancel();
```

## 6.64.2 Constructor & Destructor Documentation

### 6.64.2.1 Arc::CounterTicket::CounterTicket ( )

The default constructor.

This is the default constructor. It creates a **CounterTicket** (p. 88) that is not valid. The ticket object that is created can later be assigned a ticket that is returned by the `reserve()` method of a **Counter** (p. 81).

## 6.64.3 Member Function Documentation

### 6.64.3.1 void Arc::CounterTicket::cancel ( )

Cancels a reservation.

This method is called to cancel a reservation. It may be called also for self-expiring reservations, which will then be cancelled before they were originally planned to expire.

### 6.64.3.2 void Arc::CounterTicket::extend ( Glib::TimeVal *duration* )

Extends a reservation.

Extends a self-expiring reservation. In order to succeed the extension should be made before the previous reservation expires.

#### Parameters

*duration* The time by which to extend the reservation. The new expiration time is computed based on the current time, NOT the previous expiration time.

### 6.64.3.3 bool Arc::CounterTicket::isValid ( )

Returns the validity of a **CounterTicket** (p. 88).

This method checks whether a **CounterTicket** (p. 88) is valid. The ticket was probably returned earlier by the `reserve()` method of a **Counter** (p. 81) but the corresponding reservation may have expired.

#### Returns

The validity of the ticket.

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

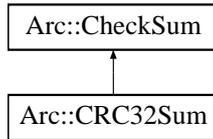
- Counter.h

## 6.65 Arc::CRC32Sum Class Reference

Implementation of CRC32 checksum.

```
#include <Checksum.h>
```

Inheritance diagram for Arc::CRC32Sum:



### 6.65.1 Detailed Description

Implementation of CRC32 checksum.

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

- CheckSum.h

## 6.66 Arc::Credential Class Reference

### Public Member Functions

- **Credential** ()
- **Credential** (int keybits)
- **Credential** (const std::string &CAfile, const std::string &CAkey, const std::string &CAserial, bool CAcreateserial, const std::string &extfile, const std::string &extsect, const std::string &passphrase4key="")
- **Credential** (Time start, Period lifetime=**Period**("PT12H"), int keybits=1024, std::string proxyversion="rfc", std::string policylang="inheritAll", std::string policy="", int pathlength=-1)
- **Credential** (const std::string &cert, const std::string &key, const std::string &cadir, const std::string &cafile, const std::string &passphrase4key="", const bool is\_file=true)
- void **AddCertExtObj** (std::string &sn, std::string &oid)
- void **LogError** (void) const
- bool **GetVerification** (void) const
- EVP\_PKEY \* **GetPrivKey** (void) const
- EVP\_PKEY \* **GetPubKey** (void) const
- X509 \* **GetCert** (void) const
- X509\_REQ \* **GetCertReq** (void) const
- **STACK\_OF** (X509)\***GetCertChain**(void) const
- int **GetCertNumofChain** (void) const
- Credformat **getFormat** (BIO \*in, const bool is\_file=true) const
- std::string **GetDN** (void) const
- std::string **GetIdentityName** (void) const
- **ArcCredential::certType** **GetType** (void) const
- std::string **GetProxyPolicy** (void) const
- void **SetProxyPolicy** (const std::string &proxyversion, const std::string &policylang, const std::string &policy, int pathlength)
- bool **OutputPrivatekey** (std::string &content, bool encryption=false, const std::string &passphrase="")
- bool **OutputPublickey** (std::string &content)
- bool **OutputCertificate** (std::string &content, bool is\_der=false)
- bool **OutputCertificateChain** (std::string &content, bool is\_der=false)
- **Period** **GetLifeTime** (void) const

- **Time GetStartTime** () const
- **Time GetEndTime** () const
- void **SetLifeTime** (const **Period** &period)
- void **SetStartTime** (const **Time** &start\_time)
- bool **AddExtension** (std::string name, std::string data, bool crit=false)
- bool **AddExtension** (std::string name, char \*\*binary, bool crit=false)
- bool **GenerateEECRequest** (BIO \*reqbio, BIO \*keybio, std::string dn="")
- bool **GenerateEECRequest** (std::string &reqcontent, std::string &keycontent, std::string dn="")
- bool **GenerateEECRequest** (const char \*request\_filename, const char \*key\_filename, std::string dn="")
- bool **GenerateRequest** (BIO \*bio, bool if\_der=false)
- bool **GenerateRequest** (std::string &content, bool if\_der=false)
- bool **GenerateRequest** (const char \*filename, bool if\_der=false)
- bool **InquireRequest** (BIO \*reqbio, bool if\_eec=false, bool if\_der=false)
- bool **InquireRequest** (std::string &content, bool if\_eec=false, bool if\_der=false)
- bool **InquireRequest** (const char \*filename, bool if\_eec=false, bool if\_der=false)
- bool **SignRequest** (**Credential** \*proxy, BIO \*outputbio, bool if\_der=false)
- bool **SignRequest** (**Credential** \*proxy, std::string &content, bool if\_der=false)
- bool **SignRequest** (**Credential** \*proxy, const char \*filename, bool foamat=false)
- bool **SignEECRequest** (**Credential** \*eec, const std::string &DN, BIO \*outputbio)
- bool **SignEECRequest** (**Credential** \*eec, const std::string &DN, std::string &content)
- bool **SignEECRequest** (**Credential** \*eec, const std::string &DN, const char \*filename)

## Static Public Member Functions

- static void **InitProxyCertInfo** (void)

## 6.66.1 Constructor & Destructor Documentation

### 6.66.1.1 Arc::Credential::Credential ( )

Default constructor, only acts as a container for inquiring certificate request, is meaningless for any other use.

### 6.66.1.2 Arc::Credential::Credential ( int *keybits* )

Constructor with user-defined keylength. Needed for creation of EE certs, since some applications will only support keys with a certain minimum length > 1024

### 6.66.1.3 Arc::Credential::Credential ( const std::string & *CAfile*, const std::string & *CAkey*, const std::string & *CAserial*, bool *CAcreateserial*, const std::string & *extfile*, const std::string & *extsect*, const std::string & *passphrase4key* = "" )

Constructor, specific constructor for CA certificate is meaningless for any other use.

**6.66.1.4** `Arc::Credential::Credential ( Time start, Period lifetime = Period ("PT12H"), int keybits = 1024, std::string proxyversion = "rfc", std::string policylang = "inheritAll", std::string policy = "", int pathlength = -1 )`

Constructor, specific constructor for proxy certificate, only acts as a container for constraining certificate signing and/or generating certificate request(only keybits is useful for creating certificate request), is meaningless for any other use. The proxyversion and policylang is for specifying the proxy certificate type and the policy language inside proxy. The definition of proxyversion and policy language is based on [http://dev.globus.org/wiki/Security/ProxyCertTypes#RFC\\_3820\\_Proxy\\_Certificates](http://dev.globus.org/wiki/Security/ProxyCertTypes#RFC_3820_Proxy_Certificates) The code is supposed to support proxy version: GSI2(legacy proxy), GSI3(Proxy draft) and RFC(RFC3820 proxy), and corresponding policy language. GSI2(GSI2, GSI2\_LIMITED) GSI3 and RFC (IMPERSONATION\_PROXY--1.3.6.1.5.5.7.21.1, INDEPENDENT\_PROXY--1.3.6.1.5.5.7.21.2, LIMITED\_PROXY--1.3.6.1.4.1.3536.1.1.1.9, RESTRICTED\_PROXY--policy language undefined) In openssl>=0.9.8, there are three types of policy languages: id-ppl-inheritAll--1.3.6.1.5.5.7.21.1, id-ppl-independent--1.3.6.1.5.5.7.21.2, and id-ppl-anyLanguage-1.3.6.1.5.5.7.21.0

#### Parameters

*start,start* time of proxy certificate

*lifetime,lifetime* of proxy certificate

*keybits,modulus* size for RSA key generation, it should be greater than 1024 if 'this' class is used for generating X509 request; it should be '0' if 'this' class is used for constraining certificate signing.

**6.66.1.5** `Arc::Credential::Credential ( const std::string & cert, const std::string & key, const std::string & cadir, const std::string & cafile, const std::string & passphrase4key = "", const bool is_file = true )`

Constructor, specific constructor for usual certificate, constructing from credential files. only acts as a container for parsing the certificate and key files, is meaningless for any other use. this constructor will parse the credential information, and put them into "this" object

#### Parameters

*is\_file,specify* if the cert/key are from file, otherwise they are supposed to be from string. default is from file

## 6.66.2 Member Function Documentation

**6.66.2.1** `void Arc::Credential::AddCertExtObj ( std::string & sn, std::string & oid )`

General method for adding a new nid into openssl's global const

**6.66.2.2** `bool Arc::Credential::AddExtension ( std::string name, std::string data, bool crit = false )`

Add an extension to the extension part of the certificate

#### Parameters

*name,the* name of the extension, there OID related with the name should be registered into openssl firstly

*data,the* data which will be inserted into certificate extension

### 6.66.2.3 bool Arc::Credential::AddExtension ( std::string name, char \*\* binary, bool crit = false )

Add an extension to the extension part of the certificate

#### Parameters

*binary*, the data which will be inserted into certificate extension part as a specific extension there should be specific methods defined inside specific X509V3\_EXT\_METHOD structure to parse the specific extension format. For example, VOMS attribute certificate is a specific extension to proxy certificate. There is specific X509V3\_EXT\_METHOD defined in **VOMSAtribute.h** (p. ??) and VOMSAttribute.c for parsing attribute certificate. In openssl, the specific X509V3\_EXT\_METHOD can be got according to the extension name/id, see X509V3\_EXT\_get\_nid(ext\_nid)

### 6.66.2.4 bool Arc::Credential::GenerateEECRequest ( BIO \* reqbio, BIO \* keybio, std::string dn = "" )

Generate an EEC request, based on the keybits and signing algorithm information inside this object output the certificate request to output BIO

The user will be asked for a private key password

### 6.66.2.5 bool Arc::Credential::GenerateEECRequest ( std::string & reqcontent, std::string & keycontent, std::string dn = "" )

Generate an EEC request, output the certificate request to a string

### 6.66.2.6 bool Arc::Credential::GenerateEECRequest ( const char \* request\_filename, const char \* key\_filename, std::string dn = "" )

Generate an EEC request, output the certificate request and the key to a file

### 6.66.2.7 bool Arc::Credential::GenerateRequest ( BIO \* bio, bool if\_der = false )

Generate a proxy request, base on the keybits and signing algorithm information inside this object output the certificate request to output BIO

### 6.66.2.8 bool Arc::Credential::GenerateRequest ( std::string & content, bool if\_der = false )

Generate a proxy request, output the certificate request to a string

### 6.66.2.9 bool Arc::Credential::GenerateRequest ( const char \* filename, bool if\_der = false )

Generate a proxy request, output the certificate request to a file

### 6.66.2.10 X509\* Arc::Credential::GetCert ( void ) const

Get the certificate attached to this object

**6.66.2.11** `int Arc::Credential::GetCertNumofChain ( void ) const`

Get the number of certificates in the certificate chain attached to this object

**6.66.2.12** `X509_REQ* Arc::Credential::GetCertReq ( void ) const`

Get the certificate request, if there is any

**6.66.2.13** `std::string Arc::Credential::GetDN ( void ) const`

Get the DN of the certificate attached to this object

**6.66.2.14** `Time Arc::Credential::GetEndTime ( ) const`

Returns validity end time of certificate or proxy

**6.66.2.15** `Credformat Arc::Credential::getFormat ( BIO * in, const bool is_file = true ) const`

Get the certificate format, PEM PKCS12 or DER BIO could be memory or file, they should be processed differently.

**6.66.2.16** `std::string Arc::Credential::GetIdentityName ( void ) const`

Get the Identity name of the certificate attached to this object, the result will not include proxy CN

**6.66.2.17** `Period Arc::Credential::GetLifeTime ( void ) const`

Returns lifetime of certificate or proxy

**6.66.2.18** `EVP_PKEY* Arc::Credential::GetPrivKey ( void ) const`

Get the private key attached to this object

**6.66.2.19** `std::string Arc::Credential::GetProxyPolicy ( void ) const`

Get the proxy policy attached to the "proxy certificate information" extension of the proxy certificate

**6.66.2.20** `EVP_PKEY* Arc::Credential::GetPubKey ( void ) const`

Get the public key attached to this object

**6.66.2.21** `Time Arc::Credential::GetStartTime ( ) const`

Returns validity start time of certificate or proxy

**6.66.2.22 ArcCredential::certType Arc::Credential::GetType ( void ) const**

Get type of the certificate attached to this object

**6.66.2.23 bool Arc::Credential::GetVerification ( void ) const [inline]**

Get the verification result about certificate chain checking

**6.66.2.24 static void Arc::Credential::InitProxyCertInfo ( void ) [static]**

Initiate nid for proxy certificate extension

**6.66.2.25 bool Arc::Credential::InquireRequest ( const char \* filename, bool if\_eec = false, bool if\_der = false )**

Inquire the certificate request from a file

**6.66.2.26 bool Arc::Credential::InquireRequest ( BIO \* reqbio, bool if\_eec = false, bool if\_der = false )**

Inquire the certificate request from BIO, and put the request information to X509\_REQ inside this object, and parse the certificate type from the PROXYCERTINFO of request' extension

**Parameters**

*if\_der* false for PEM; true for DER

**6.66.2.27 bool Arc::Credential::InquireRequest ( std::string & content, bool if\_eec = false, bool if\_der = false )**

Inquire the certificate request from a string

**6.66.2.28 void Arc::Credential::LogError ( void ) const**

Log error information related with openssl

**6.66.2.29 bool Arc::Credential::OutputCertificate ( std::string & content, bool is\_der = false )**

Output the certificate into string

**Parameters**

*is\_der* false for PEM, true for DER

**6.66.2.30 bool Arc::Credential::OutputCertificateChain ( std::string & content, bool is\_der = false )**

Output the certificate chain into string

**Parameters**

*is\_der* false for PEM, true for DER

**6.66.2.31** `bool Arc::Credential::OutputPrivatekey ( std::string & content, bool encryption = false, const std::string & passphrase = "" )`

Output the private key into string

**Parameters**

*encryption,whether* encrypt the output private key or not

*passphrase,the* passphrase to encrypt the output private key

**6.66.2.32** `bool Arc::Credential::OutputPublickey ( std::string & content )`

Output the public key into string

**6.66.2.33** `void Arc::Credential::SetLifeTime ( const Period & period )`

Set lifetime of certificate or proxy

**6.66.2.34** `void Arc::Credential::SetProxyPolicy ( const std::string & proxyversion, const std::string & policylang, const std::string & policy, int pathlength )`

Set the proxy policy attached to the "proxy certificate information" extension of the proxy certificate

**6.66.2.35** `void Arc::Credential::SetStartTime ( const Time & start_time )`

Set start time of certificate or proxy

**6.66.2.36** `bool Arc::Credential::SignEECRequest ( Credential * eec, const std::string & DN, BIO * outputbio )`

Sign eec request, and output the signed certificate to output BIO

**6.66.2.37** `bool Arc::Credential::SignEECRequest ( Credential * eec, const std::string & DN, std::string & content )`

Sign request and output the signed certificate to a string

**6.66.2.38** `bool Arc::Credential::SignEECRequest ( Credential * eec, const std::string & DN, const char * filename )`

Sign request and output the signed certificate to a file

**6.66.2.39** `bool Arc::Credential::SignRequest ( Credential * proxy, BIO * outputbio, bool if_der = false )`

Sign request based on the information inside proxy, and output the signed certificate to output BIO

#### Parameters

*if\_der* false for PEM, true for DER

**6.66.2.40** `bool Arc::Credential::SignRequest ( Credential * proxy, const char * filename, bool foamat = false )`

Sign request and output the signed certificate to a file

#### Parameters

*if\_der* false for PEM, true for DER

**6.66.2.41** `bool Arc::Credential::SignRequest ( Credential * proxy, std::string & content, bool if_der = false )`

Sign request and output the signed certificate to a string

#### Parameters

*if\_der* false for PEM, true for DER

**6.66.2.42** `Arc::Credential::STACK_OF ( X509 ) const`

Get the certificate chain attached to this object

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

- Credential.h

## 6.67 Arc::CredentialError Class Reference

```
#include <Credential.h>
```

### Public Member Functions

- `CredentialError` (const std::string &what="")

### 6.67.1 Detailed Description

This is an exception class that is used to handle runtime errors discovered in the `Credential` (p. 90) class.

## 6.67.2 Constructor & Destructor Documentation

### 6.67.2.1 Arc::CredentialError::CredentialError ( const std::string & *what* = "" )

This is the constructor of the `CredentialError` (p. 97) class.

#### Parameters

*what* An explanation of the error.

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

- Credential.h

## 6.68 Arc::CredentialStore Class Reference

```
#include <CredentialStore.h>
```

### 6.68.1 Detailed Description

This class provides functionality for storing delegated credentials and retrieving them from some store services. This is very preliminary implementation and currently support only one type of credentials - X.509 proxies, and only one type of store service - MyProxy. Later it will be extended to support at least following services: ARC delegation service, VOMS service, local file system.

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

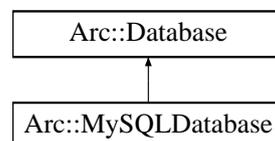
- CredentialStore.h

## 6.69 Arc::Database Class Reference

Interface for calling database client library.

```
#include <DBInterface.h>
```

Inheritance diagram for Arc::Database:



### Public Member Functions

- `Database ()`
- `Database (std::string &server, int port)`
- `Database (const Database &other)`
- `virtual ~Database ()`
- `virtual bool connect (std::string &dbname, std::string &user, std::string &password)=0`

- virtual bool **isconnected** () const =0
- virtual void **close** ()=0
- virtual bool **enable\_ssl** (const std::string keyfile="", const std::string certfile="", const std::string cafile="", const std::string capath="")=0
- virtual bool **shutdown** ()=0

### 6.69.1 Detailed Description

Interface for calling database client library. For different types of database client library, different classes should be implemented by implementing this interface.

### 6.69.2 Constructor & Destructor Documentation

#### 6.69.2.1 Arc::Database::Database ( ) [inline]

Default constructor

#### 6.69.2.2 Arc::Database::Database ( std::string & *server*, int *port* ) [inline]

Constructor which uses the server's name(or IP address) and port as parametes

#### 6.69.2.3 Arc::Database::Database ( const Database & *other* ) [inline]

Copy constructor

#### 6.69.2.4 virtual Arc::Database::~Database ( ) [inline, virtual]

Deconstructor

### 6.69.3 Member Function Documentation

#### 6.69.3.1 virtual void Arc::Database::close ( ) [pure virtual]

Close the connection with database server

Implemented in [Arc::MySQLDatabase](#) (p. 222).

#### 6.69.3.2 virtual bool Arc::Database::connect ( std::string & *dbname*, std::string & *user*, std::string & *password* ) [pure virtual]

Do connection with database server

##### Parameters

*dbname* The database name which will be used.

*user* The username which will be used to access database.

*password* The password which will be used to access database.

Implemented in [Arc::MySQLDatabase](#) (p. 222).

**6.69.3.3** `virtual bool Arc::Database::enable_ssl ( const std::string keyfile = "", const std::string certfile = "", const std::string cafile = "", const std::string capath = "" ) [pure virtual]`

Enable ssl communication for the connection

#### Parameters

- keyfile* The location of key file.
- certfile* The location of certificate file.
- cafile* The location of ca file.
- capath* The location of ca directory

Implemented in `Arc::MySQLDatabase` (p. 222).

**6.69.3.4** `virtual bool Arc::Database::isconnected ( ) const [pure virtual]`

Get the connection status

Implemented in `Arc::MySQLDatabase` (p. 222).

**6.69.3.5** `virtual bool Arc::Database::shutdown ( ) [pure virtual]`

Ask database server to shutdown

Implemented in `Arc::MySQLDatabase` (p. 222).

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

- DBInterface.h

## 6.70 Arc::DataBuffer Class Reference

Represents set of buffers.

```
#include <DataBuffer.h>
```

### Data Structures

- struct `buf_desc`
- class `checksum_desc`

### Public Member Functions

- `operator bool () const`
- `DataBuffer (unsigned int size=65536, int blocks=3)`
- `DataBuffer (Checksum *cksum, unsigned int size=65536, int blocks=3)`
- `~DataBuffer ()`
- `bool set (Checksum *cksum=NULL, unsigned int size=65536, int blocks=3)`
- `int add (Checksum *cksum)`

- char \* **operator[]** (int n)
- bool **for\_read** (int &handle, unsigned int &length, bool wait)
- bool **for\_read** ()
- bool **is\_read** (int handle, unsigned int length, unsigned long long int offset)
- bool **is\_read** (char \*buf, unsigned int length, unsigned long long int offset)
- bool **for\_write** (int &handle, unsigned int &length, unsigned long long int &offset, bool wait)
- bool **for\_write** ()
- bool **is\_written** (int handle)
- bool **is\_written** (char \*buf)
- bool **is\_notwritten** (int handle)
- bool **is\_notwritten** (char \*buf)
- void **eof\_read** (bool v)
- void **eof\_write** (bool v)
- void **error\_read** (bool v)
- void **error\_write** (bool v)
- bool **eof\_read** ()
- bool **eof\_write** ()
- bool **error\_read** ()
- bool **error\_write** ()
- bool **error\_transfer** ()
- bool **error** ()
- bool **wait\_any** ()
- bool **wait\_used** ()
- bool **checksum\_valid** () const
- const CheckSum \* **checksum\_object** () const
- bool **wait\_eof\_read** ()
- bool **wait\_read** ()
- bool **wait\_eof\_write** ()
- bool **wait\_write** ()
- bool **wait\_eof** ()
- unsigned long long int **eof\_position** () const
- unsigned int **buffer\_size** () const

## Data Fields

- DataSpeed speed

### 6.70.1 Detailed Description

Represents set of buffers. This class is used used during data transfer using **DataPoint** (p. 108) classes.

### 6.70.2 Constructor & Destructor Documentation

#### 6.70.2.1 Arc::DataBuffer::DataBuffer ( unsigned int *size* = 65536, int *blocks* = 3 )

Constructor

#### Parameters

*size* size of every buffer in bytes.

*blocks* number of buffers.

### 6.70.2.2 **Arc::DataBuffer::DataBuffer** ( **Checksum** \* *cksum*, **unsigned int** *size* = 65536, **int** *blocks* = 3 )

Constructor

#### Parameters

*size* size of every buffer in bytes.

*blocks* number of buffers.

*cksum* object which will compute checksum. Should not be destroyed till **DataBuffer** (p. 100) itself.

## 6.70.3 Member Function Documentation

### 6.70.3.1 **int** **Arc::DataBuffer::add** ( **Checksum** \* *cksum* )

Add a checksum object which will compute checksum of buffer.

#### Parameters

*cksum* object which will compute checksum. Should not be destroyed till **DataBuffer** (p. 100) itself.

#### Returns

integer position in the list of checksum objects.

### 6.70.3.2 **unsigned int** **Arc::DataBuffer::buffer\_size** ( ) **const**

Returns size of buffer in object. If not initialized then this number represents size of default buffer.

### 6.70.3.3 **const CheckSum\*** **Arc::DataBuffer::checksum\_object** ( ) **const**

Returns **Checksum** (p. 67) object specified in constructor, returns NULL if index is not in list.

#### Parameters

*index* of the checksum in question.

### 6.70.3.4 **bool** **Arc::DataBuffer::checksum\_valid** ( ) **const**

Returns true if checksum was successfully computed, returns false if index is not in list.

#### Parameters

*index* of the checksum in question.

### 6.70.3.5 **bool** **Arc::DataBuffer::eof\_read** ( )

Returns true if object was informed about end of transfer on 'read' side.

**6.70.3.6 void Arc::DataBuffer::eof\_read ( bool *v* )**

Informs object if there will be no more request for 'read' buffers. *v* true if no more requests.

**6.70.3.7 void Arc::DataBuffer::eof\_write ( bool *v* )**

Informs object if there will be no more request for 'write' buffers. *v* true if no more requests.

**6.70.3.8 bool Arc::DataBuffer::eof\_write ( )**

Returns true if object was informed about end of transfer on 'write' side.

**6.70.3.9 bool Arc::DataBuffer::error ( )**

Returns true if object was informed about error or internal error occurred.

**6.70.3.10 void Arc::DataBuffer::error\_read ( bool *v* )**

Informs object if error occurred on 'read' side.

**Parameters**

*v* true if error.

**6.70.3.11 void Arc::DataBuffer::error\_write ( bool *v* )**

Informs object if error occurred on 'write' side.

**Parameters**

*v* true if error.

**6.70.3.12 bool Arc::DataBuffer::for\_read ( int & *handle*, unsigned int & *length*, bool *wait* )**

Request buffer for READING INTO it.

**Parameters**

*handle* returns buffer's number.

*length* returns size of buffer

*wait* if true and there are no free buffers, method will wait for one.

**Returns**

true on success

**6.70.3.13 bool Arc::DataBuffer::for\_read ( )**

Check if there are buffers which can be taken by `for_read()` (p. 103). This function checks only for buffers and does not take eof and error conditions into account.

**6.70.3.14** `bool Arc::DataBuffer::for_write ( int & handle, unsigned int & length, unsigned long long int & offset, bool wait )`

Request buffer for WRITING FROM it.

**Parameters**

*handle* returns buffer's number.

*length* returns size of buffer

*wait* if true and there are no free buffers, method will wait for one.

**6.70.3.15** `bool Arc::DataBuffer::for_write ( )`

Check if there are buffers which can be taken by `for_write()` (p. 104). This function checks only for buffers and does not take eof and error conditions into account.

**6.70.3.16** `bool Arc::DataBuffer::is_notwritten ( int handle )`

Informs object that data was NOT written from buffer (and releases buffer).

**Parameters**

*handle* buffer's number.

**6.70.3.17** `bool Arc::DataBuffer::is_notwritten ( char * buf )`

Informs object that data was NOT written from buffer (and releases buffer).

**Parameters**

*buf* - address of buffer

**6.70.3.18** `bool Arc::DataBuffer::is_read ( char * buf, unsigned int length, unsigned long long int offset )`

Informs object that data was read into buffer.

**Parameters**

*buf* - address of buffer

*length* amount of data.

*offset* offset in stream, file, etc.

**6.70.3.19** `bool Arc::DataBuffer::is_read ( int handle, unsigned int length, unsigned long long int offset )`

Informs object that data was read into buffer.

**Parameters**

*handle* buffer's number.  
*length* amount of data.  
*offset* offset in stream, file, etc.

**6.70.3.20 bool Arc::DataBuffer::is\_written ( int *handle* )**

Informs object that data was written from buffer.

**Parameters**

*handle* buffer's number.

**6.70.3.21 bool Arc::DataBuffer::is\_written ( char \* *buf* )**

Informs object that data was written from buffer.

**Parameters**

*buf* - address of buffer

**6.70.3.22 bool Arc::DataBuffer::set ( CheckSum \* *cksum* = *NULL*, unsigned int *size* = 65536, int *blocks* = 3 )**

Reinitialize buffers with different parameters.

**Parameters**

*size* size of every buffer in bytes.  
*blocks* number of buffers.  
*cksum* object which will compute checksum. Should not be destroyed till **DataBuffer** (p. 100) itself.

**6.70.3.23 bool Arc::DataBuffer::wait\_any ( )**

Wait (max 60 sec.) till any action happens in object. Returns true if action is eof on any side.

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

- DataBuffer.h

## 6.71 Arc::DataCallback Class Reference

```
#include <DataCallback.h>
```

### 6.71.1 Detailed Description

This class is used by **DataHandle** (p. 106) to report missing space on local filesystem. One of 'cb' functions here will be called if operation initiated by `DataHandle::start_reading` runs out of disk space.

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

- `DataCallback.h`

## 6.72 Arc::DataHandle Class Reference

This class is a wrapper around the **DataPoint** (p. 108) class.

```
#include <DataHandle.h>
```

### 6.72.1 Detailed Description

This class is a wrapper around the **DataPoint** (p. 108) class. It simplifies the construction, use and destruction of **DataPoint** (p. 108) objects.

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

- `DataHandle.h`

## 6.73 Arc::DataMover Class Reference

```
#include <DataMover.h>
```

### Public Member Functions

- **DataMover** ()
- **~DataMover** ()
- **DataStatus Transfer** (**DataPoint** &source, **DataPoint** &destination, **FileCache** &cache, const **URLMap** &map, callback cb=NULL, void \*arg=NULL, const char \*prefix=NULL)
- **DataStatus Transfer** (**DataPoint** &source, **DataPoint** &destination, **FileCache** &cache, const **URLMap** &map, unsigned long long int min\_speed, time\_t min\_speed\_time, unsigned long long int min\_average\_speed, time\_t max\_inactivity\_time, callback cb=NULL, void \*arg=NULL, const char \*prefix=NULL)
- bool **verbose** ()
- void **verbose** (bool)
- void **verbose** (const std::string &prefix)
- bool **retry** ()
- void **retry** (bool)
- void **secure** (bool)
- void **passive** (bool)
- void **force\_to\_meta** (bool)
- bool **checks** ()
- void **checks** (bool v)
- void **set\_default\_min\_speed** (unsigned long long int min\_speed, time\_t min\_speed\_time)
- void **set\_default\_min\_average\_speed** (unsigned long long int min\_average\_speed)
- void **set\_default\_max\_inactivity\_time** (time\_t max\_inactivity\_time)

### 6.73.1 Detailed Description

A purpose of this class is to provide an interface that moves data between two locations specified by URLs. It's main action is represented by methods **DataMover::Transfer** (p. 108). Instance represents only attributes used during transfer.

### 6.73.2 Member Function Documentation

#### 6.73.2.1 bool Arc::DataMover::checks ( )

Check if check for existence of remote file is done before initiating 'reading' and 'writing' operations.

#### 6.73.2.2 void Arc::DataMover::checks ( bool *v* )

Set if to make check for existence of remote file (and probably other checks too) before initiating 'reading' and 'writing' operations.

#### Parameters

*v* true if allowed (default is true).

#### 6.73.2.3 void Arc::DataMover::force\_to\_meta ( bool )

Set if file should be transferred and registered even if such LFN is already registered and source is not one of registered locations.

#### 6.73.2.4 void Arc::DataMover::secure ( bool )

Set if high level of security (encryption) will be used during transfer if available.

#### 6.73.2.5 void Arc::DataMover::set\_default\_max\_inactivity\_time ( time\_t *max\_inactivity\_time* ) [inline]

Set maximal allowed time for waiting for any data. For more information see description of **DataSpeed** (p. 129) class.

#### 6.73.2.6 void Arc::DataMover::set\_default\_min\_average\_speed ( unsigned long long int *min\_average\_speed* ) [inline]

Set minimal allowed average transfer speed (default is 0 averaged over whole time of transfer. For more information see description of **DataSpeed** (p. 129) class.

#### 6.73.2.7 void Arc::DataMover::set\_default\_min\_speed ( unsigned long long int *min\_speed*, time\_t *min\_speed\_time* ) [inline]

Set minimal allowed transfer speed (default is 0) to 'min\_speed'. If speed drops below for time longer than 'min\_speed\_time' error is raised. For more information see description of **DataSpeed** (p. 129) class.

### 6.73.2.8 `DataStatus Arc::DataMover::Transfer ( DataPoint & source, DataPoint & destination, FileCache & cache, const URLMap & map, callback cb = NULL, void * arg = NULL, const char * prefix = NULL )`

Initiates transfer from 'source' to 'destination'.

#### Parameters

*source* source [URL](#) (p. 323).

*destination* destination [URL](#) (p. 323).

*cache* controls caching of downloaded files (if destination url is "file://"). If caching is not needed default constructor `FileCache()` can be used.

*map* [URL](#) (p. 323) mapping/conversion table (for 'source' [URL](#) (p. 323)).

*cb* if not `NULL`, transfer is done in separate thread and 'cb' is called after transfer completes/fails.

*arg* passed to 'cb'.

*prefix* if 'verbose' is activated this information will be printed before each line representing current transfer status.

### 6.73.2.9 `DataStatus Arc::DataMover::Transfer ( DataPoint & source, DataPoint & destination, FileCache & cache, const URLMap & map, unsigned long long int min_speed, time_t min_speed_time, unsigned long long int min_average_speed, time_t max_inactivity_time, callback cb = NULL, void * arg = NULL, const char * prefix = NULL )`

Initiates transfer from 'source' to 'destination'.

#### Parameters

*min\_speed* minimal allowed current speed.

*min\_speed\_time* time for which speed should be less than 'min\_speed' before transfer fails.

*min\_average\_speed* minimal allowed average speed.

*max\_inactivity\_time* time for which should be no activity before transfer fails.

### 6.73.2.10 `void Arc::DataMover::verbose ( const std::string & prefix )`

Activate printing information about transfer status.

#### Parameters

*prefix* use this string if 'prefix' in `DataMover::Transfer` (p. 108) is `NULL`.

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

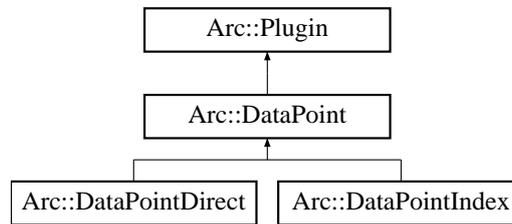
- `DataMover.h`

## 6.74 `Arc::DataPoint` Class Reference

This base class is an abstraction of [URL](#) (p. 323).

```
#include <DataPoint.h>
```

Inheritance diagram for Arc::DataPoint:



## Public Types

- enum **DataPointAccessLatency** { ACCESS\_LATENCY\_ZERO, ACCESS\_LATENCY\_SMALL, ACCESS\_LATENCY\_LARGE }
- enum **DataPointInfoType** { ,  
INFO\_TYPE\_NAME = 1, INFO\_TYPE\_TYPE = 2, INFO\_TYPE\_TIMES = 4, INFO\_TYPE\_CONTENT = 8,  
INFO\_TYPE\_ACCESS = 16, INFO\_TYPE\_STRUCT = 32, INFO\_TYPE\_REST = 64, INFO\_TYPE\_ALL = 127 }

## Public Member Functions

- **DataPoint** (const URL &url, const UserConfig &usercfg)
- virtual ~**DataPoint** ()
- virtual const URL & **GetURL** () const
- virtual const UserConfig & **GetUserConfig** () const
- virtual std::string **str** () const
- virtual **operator bool** () const
- virtual **operator!** () const
- virtual **DataStatus StartReading** (DataBuffer &buffer)=0
- virtual **DataStatus StartWriting** (DataBuffer &buffer, DataCallback \*space\_cb=NULL)=0
- virtual **DataStatus StopReading** ()=0
- virtual **DataStatus StopWriting** ()=0
- virtual **DataStatus Check** ()=0
- virtual **DataStatus Remove** ()=0
- virtual **DataStatus Stat** (FileInfo &file, DataPointInfoType verb=INFO\_TYPE\_ALL)=0
- virtual **DataStatus List** (std::list< FileInfo > &files, DataPointInfoType verb=INFO\_TYPE\_ALL)=0
- virtual void **ReadOutOfOrder** (bool v)=0
- virtual bool **WriteOutOfOrder** ()=0
- virtual void **SetAdditionalChecks** (bool v)=0
- virtual bool **GetAdditionalChecks** () const =0
- virtual void **SetSecure** (bool v)=0
- virtual bool **GetSecure** () const =0
- virtual void **Passive** (bool v)=0
- virtual **DataStatus GetFailureReason** (void) const
- virtual void **Range** (unsigned long long int start=0, unsigned long long int end=0)=0
- virtual **DataStatus Resolve** (bool source)=0

- virtual bool **Registered** () const =0
- virtual **DataStatus PreRegister** (bool replication, bool force=false)=0
- virtual **DataStatus PostRegister** (bool replication)=0
- virtual **DataStatus PreUnregister** (bool replication)=0
- virtual **DataStatus Unregister** (bool all)=0
- virtual bool **CheckSize** () const
- virtual void **SetSize** (const unsigned long long int val)
- virtual unsigned long long int **GetSize** () const
- virtual bool **CheckChecksum** () const
- virtual void **SetChecksum** (const std::string &val)
- virtual const std::string & **GetChecksum** () const
- virtual const std::string **DefaultChecksum** () const
- virtual bool **CheckCreated** () const
- virtual void **SetCreated** (const **Time** &val)
- virtual const **Time** & **GetCreated** () const
- virtual bool **CheckValid** () const
- virtual void **SetValid** (const **Time** &val)
- virtual const **Time** & **GetValid** () const
- virtual void **SetAccessLatency** (const **DataPointAccessLatency** &latency)
- virtual **DataPointAccessLatency GetAccessLatency** () const
- virtual long long int **BufSize** () const =0
- virtual int **BufNum** () const =0
- virtual bool **Cache** () const
- virtual bool **Local** () const =0
- virtual int **GetTries** () const
- virtual void **SetTries** (const int n)
- virtual void **NextTry** (void)
- virtual bool **IsIndex** () const =0
- virtual bool **AcceptsMeta** ()=0
- virtual bool **ProvidesMeta** ()=0
- virtual void **SetMeta** (const **DataPoint** &p)
- virtual bool **CompareMeta** (const **DataPoint** &p) const
- virtual const **URL** & **CurrentLocation** () const =0
- virtual const std::string & **CurrentLocationMetadata** () const =0
- virtual **DataStatus CompareLocationMetadata** () const =0
- virtual bool **NextLocation** ()=0
- virtual bool **LocationValid** () const =0
- virtual bool **LastLocation** ()=0
- virtual bool **HaveLocations** () const =0
- virtual **DataStatus AddLocation** (const **URL** &url, const std::string &meta)=0
- virtual **DataStatus RemoveLocation** ()=0
- virtual **DataStatus RemoveLocations** (const **DataPoint** &p)=0
- virtual int **AddChecksumObject** (**Checksum** \*cksum)=0
- virtual void **SortLocations** (const std::string &pattern, const **URLMap** &url\_map)=0

## Protected Attributes

- std::list< std::string > **valid\_url\_options**

## 6.74.1 Detailed Description

This base class is an abstraction of **URL** (p. 323). Specializations should be provided for different kind of direct access URLs (`file://`, `ftp://`, `gsiftp://`, `http://`, `https://`, `httpg://`, ...) or indexing service URLs (`rls://`, `lfc://`, ...). **DataPoint** (p. 108) provides means to resolve an indexing service **URL** (p. 323) into multiple URLs and to loop through them.

## 6.74.2 Member Enumeration Documentation

### 6.74.2.1 enum Arc::DataPoint::DataPointAccessLatency

Describes the latency to access this **URL** (p. 323).

For now this value is one of a small set specified by the enumeration. In the future with more sophisticated protocols or information it could be replaced by a more fine-grained list of possibilities such as an int value.

#### Enumerator:

*ACCESS\_LATENCY\_ZERO* **URL** (p. 323) can be accessed instantly.

*ACCESS\_LATENCY\_SMALL* **URL** (p. 323) has low (but non-zero) access latency, for example staged from disk.

*ACCESS\_LATENCY\_LARGE* **URL** (p. 323) has a large access latency, for example staged from tape.

### 6.74.2.2 enum Arc::DataPoint::DataPointInfoType

Describes type of information about **URL** (p. 323) to request.

#### Enumerator:

*INFO\_TYPE\_NAME* Whatever protocol can get with no additional effort.

*INFO\_TYPE\_TYPE* Only name of object (relative).

*INFO\_TYPE\_TIMES* Type of object - currently file or dir.

*INFO\_TYPE\_CONTENT* Timestamps associated with object.

*INFO\_TYPE\_ACCESS* Metadata describing content, like size, checksum, etc.

*INFO\_TYPE\_STRUCT* Access control - ownership, permission, etc.

*INFO\_TYPE\_REST* Fine structure - replicas, transfer locations, redirections.

*INFO\_TYPE\_ALL* All the other parameters.

## 6.74.3 Constructor & Destructor Documentation

### 6.74.3.1 Arc::DataPoint::DataPoint ( const URL & url, const UserConfig & usercfg )

Constructor requires **URL** (p. 323) to be provided.

References to url and usercfg arguments are stored internally and hence corresponding objects must stay available during whole lifetime of this instance.

## 6.74.4 Member Function Documentation

### 6.74.4.1 `virtual int Arc::DataPoint::AddChecksumObject ( CheckSum * cksum ) [pure virtual]`

Add a checksum object which will compute checksum during transmission.

#### Parameters

*cksum* object which will compute checksum. Should not be destroyed till DataPointer itself.

#### Returns

integer position in the list of checksum objects.

Implemented in `Arc::DataPointDirect` (p. 119), and `Arc::DataPointIndex` (p. 124).

### 6.74.4.2 `virtual DataStatus Arc::DataPoint::AddLocation ( const URL & url, const std::string & meta ) [pure virtual]`

Add `URL` (p. 323) to list.

#### Parameters

*url* Location `URL` (p. 323) to add.

*meta* Location meta information.

Implemented in `Arc::DataPointDirect` (p. 119), and `Arc::DataPointIndex` (p. 125).

### 6.74.4.3 `virtual DataStatus Arc::DataPoint::Check ( ) [pure virtual]`

**Query** (p. 259) the `DataPoint` (p. 108) to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implemented in `Arc::DataPointIndex` (p. 125).

### 6.74.4.4 `virtual DataStatus Arc::DataPoint::CompareLocationMetadata ( ) const [pure virtual]`

Compare metadata of `DataPoint` (p. 108) and current location.

Returns inconsistency error or error encountered during operation, or success

Implemented in `Arc::DataPointDirect` (p. 120), and `Arc::DataPointIndex` (p. 125).

### 6.74.4.5 `virtual bool Arc::DataPoint::CompareMeta ( const DataPoint & p ) const [virtual]`

Compare meta information from another object.

Undefined values are not used for comparison.

#### Parameters

*p* object to which to compare.

**6.74.4.6 virtual const std::string& Arc::DataPoint::CurrentLocationMetadata ( ) const [pure virtual]**

Returns meta information used to create current **URL** (p. 323).

Usage differs between different indexing services.

Implemented in **Arc::DataPointDirect** (p. 120), and **Arc::DataPointIndex** (p. 125).

**6.74.4.7 virtual DataStatus Arc::DataPoint::GetFailureReason ( void ) const [virtual]**

Returns reason of transfer failure, as reported by callbacks. This could be different from the failure returned by the methods themselves.

**6.74.4.8 virtual DataStatus Arc::DataPoint::List ( std::list< FileInfo > & files, DataPointInfoType verb = INFO\_TYPE\_ALL ) [pure virtual]**

List hierarchical content of this object.

If the **DataPoint** (p. 108) represents a directory or something similar its contents will be listed.

**Parameters**

*files* will contain list of file names and requested attributes. There may be more attributes than requested. There may be less if object can't provide particular information.

*verb* defines attribute types which method must try to retrieve. It is not a failure if some attributes could not be retrieved due to limitation of protocol or access control.

**6.74.4.9 virtual bool Arc::DataPoint::NextLocation ( ) [pure virtual]**

Switch to next location in list of URLs.

At last location switch to first if number of allowed retries is not exceeded. Returns false if no retries left.

Implemented in **Arc::DataPointDirect** (p. 120), and **Arc::DataPointIndex** (p. 125).

**6.74.4.10 virtual void Arc::DataPoint::Passive ( bool v ) [pure virtual]**

Request passive transfers for FTP-like protocols.

**Parameters**

*true* to request.

Implemented in **Arc::DataPointDirect** (p. 120), and **Arc::DataPointIndex** (p. 125).

**6.74.4.11 virtual DataStatus Arc::DataPoint::PostRegister ( bool replication ) [pure virtual]**

Index **Service** (p. 285) postregistration.

Used for same purpose as PreRegister. Should be called after actual transfer of file successfully finished.

**Parameters**

*replication* if true, the file is being replicated between two locations registered in Indexing **Service** (p. 285) under same name.

Implemented in **Arc::DataPointDirect** (p. 120).

#### 6.74.4.12 virtual **DataStatus** **Arc::DataPoint::PreRegister** ( **bool** *replication*, **bool** *force = false* ) [**pure virtual**]

Index service preregistration.

This function registers the physical location of a file into an indexing service. It should be called *\*before\** the actual transfer to that location happens.

**Parameters**

*replication* if true, the file is being replicated between two locations registered in the indexing service under same name.

*force* if true, perform registration of a new file even if it already exists. Should be used to fix failures in Indexing **Service** (p. 285).

Implemented in **Arc::DataPointDirect** (p. 120).

#### 6.74.4.13 virtual **DataStatus** **Arc::DataPoint::PreUnregister** ( **bool** *replication* ) [**pure virtual**]

Index **Service** (p. 285) preunregistration.

Should be called if file transfer failed. It removes changes made by **PreRegister**.

**Parameters**

*replication* if true, the file is being replicated between two locations registered in Indexing **Service** (p. 285) under same name.

Implemented in **Arc::DataPointDirect** (p. 121).

#### 6.74.4.14 virtual **bool** **Arc::DataPoint::ProvidesMeta** ( ) [**pure virtual**]

If endpoint can provide at least some meta information directly.

Implemented in **Arc::DataPointDirect** (p. 121), and **Arc::DataPointIndex** (p. 126).

#### 6.74.4.15 virtual **void** **Arc::DataPoint::Range** ( **unsigned long long int** *start = 0*, **unsigned long long int** *end = 0* ) [**pure virtual**]

Set range of bytes to retrieve.

Default values correspond to whole file.

Implemented in **Arc::DataPointDirect** (p. 121), and **Arc::DataPointIndex** (p. 126).

**6.74.4.16 virtual void Arc::DataPoint::ReadOutOfOrder ( bool v ) [pure virtual]**

List file(s).

If the **DataPoint** (p. 108) represents a directory its contents will be listed.

**Parameters**

*files* will contain list of file names and optionally their attributes.

*long\_list* if true, list additional properties of each file.

*resolve* if true, resolve physical locations (relevant for indexing services only).

*metadata* if true, find all available metadata. Allow/disallow **DataPoint** (p. 108) to produce scattered data during reading\* operation.

*v* true if allowed (default is false).

Implemented in **Arc::DataPointDirect** (p. 121), and **Arc::DataPointIndex** (p. 126).

**6.74.4.17 virtual bool Arc::DataPoint::Registered ( ) const [pure virtual]**

Check if file is registered in Indexing **Service** (p. 285).

Proper value is obtainable only after Resolve.

Implemented in **Arc::DataPointDirect** (p. 122), and **Arc::DataPointIndex** (p. 126).

**6.74.4.18 virtual DataStatus Arc::DataPoint::Resolve ( bool source ) [pure virtual]**

Resolves index service **URL** (p. 323) into list of ordinary URLs.

Also obtains meta information about the file.

**Parameters**

*source* true if **DataPoint** (p. 108) object represents source of information.

Implemented in **Arc::DataPointDirect** (p. 122).

**6.74.4.19 virtual void Arc::DataPoint::SetAdditionalChecks ( bool v ) [pure virtual]**

Allow/disallow additional checks.

Check for existence of remote file (and probably other checks too) before initiating reading and writing operations.

**Parameters**

*v* true if allowed (default is true).

Implemented in **Arc::DataPointDirect** (p. 122), and **Arc::DataPointIndex** (p. 126).

**6.74.4.20 virtual void Arc::DataPoint::SetMeta ( const DataPoint & p ) [virtual]**

Copy meta information from another object.

Already defined values are not overwritten.

**Parameters**

*p* object from which information is taken.

Reimplemented in **Arc::DataPointIndex** (p. 127).

**6.74.4.21 virtual void Arc::DataPoint::SetSecure ( bool *v* ) [pure virtual]**

Allow/disallow heavy security during data transfer.

**Parameters**

*v* true if allowed (default depends on protocol).

Implemented in **Arc::DataPointDirect** (p. 122), and **Arc::DataPointIndex** (p. 127).

**6.74.4.22 virtual void Arc::DataPoint::SortLocations ( const std::string & *pattern*, const URLMap & *url\_map* ) [pure virtual]**

Sort locations according to the specified pattern.

**Parameters**

*pattern* a set of strings, separated by |, to match against.

Implemented in **Arc::DataPointDirect** (p. 122), and **Arc::DataPointIndex** (p. 127).

**6.74.4.23 virtual DataStatus Arc::DataPoint::StartReading ( DataBuffer & *buffer* ) [pure virtual]**

Start reading data from **URL** (p. 323).

Separate thread to transfer data will be created. No other operation can be performed while reading is in progress.

**Parameters**

*buffer* operation will use this buffer to put information into. Should not be destroyed before stop\_<sub>-</sub>reading was called and returned.

Implemented in **Arc::DataPointIndex** (p. 127).

**6.74.4.24 virtual DataStatus Arc::DataPoint::StartWriting ( DataBuffer & *buffer*, DataCallback \* *space\_cb* = NULL ) [pure virtual]**

Start writing data to **URL** (p. 323).

Separate thread to transfer data will be created. No other operation can be performed while writing is in progress.

**Parameters**

*buffer* operation will use this buffer to get information from. Should not be destroyed before stop\_<sub>-</sub>writing was called and returned.

*space\_cb* callback which is called if there is not enough space to store data. May not implemented for all protocols.

Implemented in **Arc::DataPointIndex** (p. 127).

#### 6.74.4.25 virtual `DataStatus Arc::DataPoint::Stat ( FileInfo & file, DataPointInfoType verb = INFO_TYPE_ALL ) [pure virtual]`

Retrieve information about this object.

If the **DataPoint** (p. 108) represents a directory or something similar its contents will be listed.

##### Parameters

*file* will contain object name and requested attributes. There may be more attributes than requested. There may be less if object can't provide particular information.

*verb* defines attribute types which method must try to retrieve. It is not a failure if some attributes could not be retrieved due to limitation of protocol or access control.

#### 6.74.4.26 virtual `DataStatus Arc::DataPoint::StopReading ( ) [pure virtual]`

Stop reading.

Must be called after corresponding `start_reading` method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implemented in **Arc::DataPointIndex** (p. 128).

#### 6.74.4.27 virtual `DataStatus Arc::DataPoint::StopWriting ( ) [pure virtual]`

Stop writing.

Must be called after corresponding `start_writing` method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implemented in **Arc::DataPointIndex** (p. 128).

#### 6.74.4.28 virtual `DataStatus Arc::DataPoint::Unregister ( bool all ) [pure virtual]`

Index **Service** (p. 285) unregistration.

Remove information about file registered in Indexing **Service** (p. 285).

##### Parameters

*all* if true, information about file itself is (LFN) is removed. Otherwise only particular physical instance is unregistered.

Implemented in **Arc::DataPointDirect** (p. 123).

#### 6.74.4.29 virtual bool Arc::DataPoint::WriteOutOfOrder ( ) [pure virtual]

Returns true if **URL** (p. 323) can accept scattered data for \*writing\* operation.

Implemented in **Arc::DataPointDirect** (p. 123), and **Arc::DataPointIndex** (p. 128).

### 6.74.5 Field Documentation

#### 6.74.5.1 std::list<std::string> Arc::DataPoint::valid\_url\_options [protected]

Subclasses should add their own specific options to this list

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

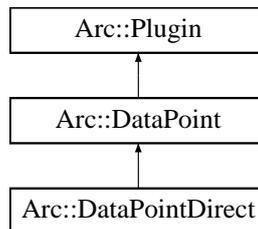
- DataPoint.h

## 6.75 Arc::DataPointDirect Class Reference

This is a kind of generalized file handle.

```
#include <DataPointDirect.h>
```

Inheritance diagram for Arc::DataPointDirect:



### Public Member Functions

- virtual bool **IsIndex** ( ) const
- virtual long long int **BufSize** ( ) const
- virtual int **BufNum** ( ) const
- virtual bool **Local** ( ) const
- virtual void **ReadOutOfOrder** (bool v)
- virtual bool **WriteOutOfOrder** ( )
- virtual void **SetAdditionalChecks** (bool v)
- virtual bool **GetAdditionalChecks** ( ) const
- virtual void **SetSecure** (bool v)
- virtual bool **GetSecure** ( ) const
- virtual void **Passive** (bool v)
- virtual void **Range** (unsigned long long int start=0, unsigned long long int end=0)
- virtual int **AddChecksumObject** (**Checksum** \*cksum)
- virtual **DataStatus** **Resolve** (bool source)
- virtual bool **Registered** ( ) const
- virtual **DataStatus** **PreRegister** (bool replication, bool force=false)

- virtual **DataStatus PostRegister** (bool replication)
- virtual **DataStatus PreUnregister** (bool replication)
- virtual **DataStatus Unregister** (bool all)
- virtual bool **AcceptsMeta** ()
- virtual bool **ProvidesMeta** ()
- virtual const **URL & CurrentLocation** () const
- virtual const std::string & **CurrentLocationMetadata** () const
- virtual **DataStatus CompareLocationMetadata** () const
- virtual bool **NextLocation** ()
- virtual bool **LocationValid** () const
- virtual bool **HaveLocations** () const
- virtual bool **LastLocation** ()
- virtual **DataStatus AddLocation** (const **URL** &url, const std::string &meta)
- virtual **DataStatus RemoveLocation** ()
- virtual **DataStatus RemoveLocations** (const **DataPoint** &p)
- virtual void **SortLocations** (const std::string &, const **URLMap** &)

### 6.75.1 Detailed Description

This is a kind of generalized file handle. Differently from file handle it does not support operations read() and write(). Instead it initiates operation and uses object of class **DataBuffer** (p. 100) to pass actual data. It also provides other operations like querying parameters of remote object. It is used by higher-level classes DataMove and DataMovePar to provide data transfer service for application.

### 6.75.2 Member Function Documentation

#### 6.75.2.1 virtual int Arc::DataPointDirect::AddChecksumObject ( **Checksum** \* *cksum* ) [**virtual**]

Add a checksum object which will compute checksum during transmission.

##### Parameters

*cksum* object which will compute checksum. Should not be destroyed till DataPointer itself.

##### Returns

integer position in the list of checksum objects.

Implements **Arc::DataPoint** (p. 112).

#### 6.75.2.2 virtual DataStatus Arc::DataPointDirect::AddLocation ( const **URL** & *url*, const std::string & *meta* ) [**virtual**]

Add **URL** (p. 323) to list.

##### Parameters

*url* Location **URL** (p. 323) to add.

*meta* Location meta information.

Implements **Arc::DataPoint** (p. 112).

**6.75.2.3 virtual DataStatus Arc::DataPointDirect::CompareLocationMetadata ( ) const [virtual]**

Compare metadata of **DataPoint** (p. 108) and current location.

Returns inconsistency error or error encountered during operation, or success

Implements **Arc::DataPoint** (p. 112).

**6.75.2.4 virtual const std::string& Arc::DataPointDirect::CurrentLocationMetadata ( ) const [virtual]**

Returns meta information used to create current **URL** (p. 323).

Usage differs between different indexing services.

Implements **Arc::DataPoint** (p. 113).

**6.75.2.5 virtual bool Arc::DataPointDirect::NextLocation ( ) [virtual]**

Switch to next location in list of URLs.

At last location switch to first if number of allowed retries is not exceeded. Returns false if no retries left.

Implements **Arc::DataPoint** (p. 113).

**6.75.2.6 virtual void Arc::DataPointDirect::Passive ( bool v ) [virtual]**

Request passive transfers for FTP-like protocols.

**Parameters**

*true* to request.

Implements **Arc::DataPoint** (p. 113).

**6.75.2.7 virtual DataStatus Arc::DataPointDirect::PostRegister ( bool replication ) [virtual]**

Index **Service** (p. 285) postregistration.

Used for same purpose as PreRegister. Should be called after actual transfer of file successfully finished.

**Parameters**

*replication* if true, the file is being replicated between two locations registered in Indexing **Service** (p. 285) under same name.

Implements **Arc::DataPoint** (p. 113).

**6.75.2.8 virtual DataStatus Arc::DataPointDirect::PreRegister ( bool replication, bool force = false ) [virtual]**

Index service preregistration.

This function registers the physical location of a file into an indexing service. It should be called *\*before\** the actual transfer to that location happens.

**Parameters**

*replication* if true, the file is being replicated between two locations registered in the indexing service under same name.

*force* if true, perform registration of a new file even if it already exists. Should be used to fix failures in Indexing **Service** (p. 285).

Implements **Arc::DataPoint** (p. 114).

### 6.75.2.9 virtual DataStatus Arc::DataPointDirect::PreUnregister ( bool *replication* ) [virtual]

Index **Service** (p. 285) preunregistration.

Should be called if file transfer failed. It removes changes made by PreRegister.

**Parameters**

*replication* if true, the file is being replicated between two locations registered in Indexing **Service** (p. 285) under same name.

Implements **Arc::DataPoint** (p. 114).

### 6.75.2.10 virtual bool Arc::DataPointDirect::ProvidesMeta ( ) [virtual]

If endpoint can provide at least some meta information directly.

Implements **Arc::DataPoint** (p. 114).

### 6.75.2.11 virtual void Arc::DataPointDirect::Range ( unsigned long long int *start* = 0, unsigned long long int *end* = 0 ) [virtual]

Set range of bytes to retrieve.

Default values correspond to whole file.

Implements **Arc::DataPoint** (p. 114).

### 6.75.2.12 virtual void Arc::DataPointDirect::ReadOutOfOrder ( bool *v* ) [virtual]

List file(s).

If the **DataPoint** (p. 108) represents a directory its contents will be listed.

**Parameters**

*files* will contain list of file names and optionally their attributes.

*long\_list* if true, list additional properties of each file.

*resolve* if true, resolve physical locations (relevant for indexing services only).

*metadata* if true, find all available metadata. Allow/disallow **DataPoint** (p. 108) to produce scattered data during reading\* operation.

*v* true if allowed (default is false).

Implements **Arc::DataPoint** (p. 115).

**6.75.2.13 virtual bool Arc::DataPointDirect::Registered ( ) const [virtual]**

Check if file is registered in Indexing **Service** (p. 285).

Proper value is obtainable only after Resolve.

Implements **Arc::DataPoint** (p. 115).

**6.75.2.14 virtual DataStatus Arc::DataPointDirect::Resolve ( bool source ) [virtual]**

Resolves index service **URL** (p. 323) into list of ordinary URLs.

Also obtains meta information about the file.

**Parameters**

*source* true if **DataPoint** (p. 108) object represents source of information.

Implements **Arc::DataPoint** (p. 115).

**6.75.2.15 virtual void Arc::DataPointDirect::SetAdditionalChecks ( bool v ) [virtual]**

Allow/disallow additional checks.

Check for existence of remote file (and probably other checks too) before initiating reading and writing operations.

**Parameters**

*v* true if allowed (default is true).

Implements **Arc::DataPoint** (p. 115).

**6.75.2.16 virtual void Arc::DataPointDirect::SetSecure ( bool v ) [virtual]**

Allow/disallow heavy security during data transfer.

**Parameters**

*v* true if allowed (default depends on protocol).

Implements **Arc::DataPoint** (p. 116).

**6.75.2.17 virtual void Arc::DataPointDirect::SortLocations ( const std::string & pattern, const URLMap & url\_map ) [inline, virtual]**

Sort locations according to the specified pattern.

**Parameters**

*pattern* a set of strings, separated by |, to match against.

Implements **Arc::DataPoint** (p. 116).

**6.75.2.18 virtual DataStatus Arc::DataPointDirect::Unregister ( bool *all* ) [virtual]**

Index **Service** (p. 285) unregistration.

Remove information about file registered in Indexing **Service** (p. 285).

**Parameters**

- all* if true, information about file itself is (LFN) is removed. Otherwise only particular physical instance is unregistered.

Implements **Arc::DataPoint** (p. 117).

**6.75.2.19 virtual bool Arc::DataPointDirect::WriteOutOfOrder ( ) [virtual]**

Returns true if **URL** (p. 323) can accept scattered data for \*writing\* operation.

Implements **Arc::DataPoint** (p. 118).

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

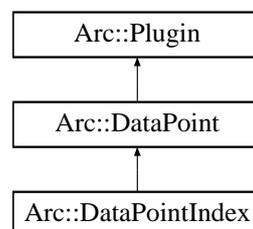
- DataPointDirect.h

**6.76 Arc::DataPointIndex Class Reference**

Complements **DataPoint** (p. 108) with attributes common for Indexing **Service** (p. 285) URLs.

```
#include <DataPointIndex.h>
```

Inheritance diagram for Arc::DataPointIndex:

**Public Member Functions**

- virtual const **URL & CurrentLocation** () const
- virtual const std::string & **CurrentLocationMetadata** () const
- virtual **DataStatus CompareLocationMetadata** () const
- virtual bool **NextLocation** ()
- virtual bool **LocationValid** () const
- virtual bool **HaveLocations** () const
- virtual bool **LastLocation** ()
- virtual **DataStatus RemoveLocation** ()
- virtual **DataStatus RemoveLocations** (const **DataPoint** &p)
- virtual **DataStatus AddLocation** (const **URL** &url, const std::string &meta)
- virtual void **SortLocations** (const std::string &pattern, const **URLMap** &url\_map)

- virtual bool **IsIndex** () const
- virtual bool **AcceptsMeta** ()
- virtual bool **ProvidesMeta** ()
- virtual void **SetMeta** (const **DataPoint** &p)
- virtual void **SetChecksum** (const std::string &val)
- virtual void **SetSize** (const unsigned long long int val)
- virtual bool **Registered** () const
- virtual void **SetTries** (const int n)
- virtual long long int **BufSize** () const
- virtual int **BufNum** () const
- virtual bool **Local** () const
- virtual **DataStatus** **StartReading** (**DataBuffer** &buffer)
- virtual **DataStatus** **StartWriting** (**DataBuffer** &buffer, **DataCallback** \*space\_cb=NULL)
- virtual **DataStatus** **StopReading** ()
- virtual **DataStatus** **StopWriting** ()
- virtual **DataStatus** **Check** ()
- virtual **DataStatus** **Remove** ()
- virtual void **ReadOutOfOrder** (bool v)
- virtual bool **WriteOutOfOrder** ()
- virtual void **SetAdditionalChecks** (bool v)
- virtual bool **GetAdditionalChecks** () const
- virtual void **SetSecure** (bool v)
- virtual bool **GetSecure** () const
- virtual **DataPointAccessLatency** **GetAccessLatency** () const
- virtual void **Passive** (bool v)
- virtual void **Range** (unsigned long long int start=0, unsigned long long int end=0)
- virtual int **AddChecksumObject** (**Checksum** \*cksum)

### 6.76.1 Detailed Description

Complements **DataPoint** (p. 108) with attributes common for Indexing **Service** (p. 285) URLs. It should never be used directly. Instead inherit from it to provide a class for specific a Indexing **Service** (p. 285).

### 6.76.2 Member Function Documentation

#### 6.76.2.1 virtual int Arc::DataPointIndex::AddChecksumObject ( **Checksum** \* *cksum* ) [virtual]

Add a checksum object which will compute checksum during transmission.

#### Parameters

*cksum* object which will compute checksum. Should not be destroyed till DataPointer itself.

#### Returns

integer position in the list of checksum objects.

Implements **Arc::DataPoint** (p. 112).

**6.76.2.2 virtual DataStatus Arc::DataPointIndex::AddLocation ( const URL & *url*, const std::string & *meta* ) [virtual]**

Add **URL** (p. 323) to list.

**Parameters**

*url* Location **URL** (p. 323) to add.

*meta* Location meta information.

Implements **Arc::DataPoint** (p. 112).

**6.76.2.3 virtual DataStatus Arc::DataPointIndex::Check ( ) [virtual]**

**Query** (p. 259) the **DataPoint** (p. 108) to check if object is accessible.

If possible this method will also try to provide meta information about the object. It returns positive response if object's content can be retrieved.

Implements **Arc::DataPoint** (p. 112).

**6.76.2.4 virtual DataStatus Arc::DataPointIndex::CompareLocationMetadata ( ) const [virtual]**

Compare metadata of **DataPoint** (p. 108) and current location.

Returns inconsistency error or error encountered during operation, or success

Implements **Arc::DataPoint** (p. 112).

**6.76.2.5 virtual const std::string& Arc::DataPointIndex::CurrentLocationMetadata ( ) const [virtual]**

Returns meta information used to create current **URL** (p. 323).

Usage differs between different indexing services.

Implements **Arc::DataPoint** (p. 113).

**6.76.2.6 virtual bool Arc::DataPointIndex::NextLocation ( ) [virtual]**

Switch to next location in list of URLs.

At last location switch to first if number of allowed retries is not exceeded. Returns false if no retries left.

Implements **Arc::DataPoint** (p. 113).

**6.76.2.7 virtual void Arc::DataPointIndex::Passive ( bool *v* ) [virtual]**

Request passive transfers for FTP-like protocols.

**Parameters**

*true* to request.

Implements **Arc::DataPoint** (p. 113).

**6.76.2.8 virtual bool Arc::DataPointIndex::ProvidesMeta ( ) [virtual]**

If endpoint can provide at least some meta information directly.

Implements **Arc::DataPoint** (p. 114).

**6.76.2.9 virtual void Arc::DataPointIndex::Range ( unsigned long long int start = 0, unsigned long long int end = 0 ) [virtual]**

Set range of bytes to retrieve.

Default values correspond to whole file.

Implements **Arc::DataPoint** (p. 114).

**6.76.2.10 virtual void Arc::DataPointIndex::ReadOutOfOrder ( bool v ) [virtual]**

List file(s).

If the **DataPoint** (p. 108) represents a directory its contents will be listed.

**Parameters**

*files* will contain list of file names and optionally their attributes.

*long\_list* if true, list additional properties of each file.

*resolve* if true, resolve physical locations (relevant for indexing services only).

*metadata* if true, find all available metadata. Allow/disallow **DataPoint** (p. 108) to produce scattered data during reading\* operation.

*v* true if allowed (default is false).

Implements **Arc::DataPoint** (p. 115).

**6.76.2.11 virtual bool Arc::DataPointIndex::Registered ( ) const [virtual]**

Check if file is registered in Indexing **Service** (p. 285).

Proper value is obtainable only after Resolve.

Implements **Arc::DataPoint** (p. 115).

**6.76.2.12 virtual void Arc::DataPointIndex::SetAdditionalChecks ( bool v ) [virtual]**

Allow/disallow additional checks.

Check for existence of remote file (and probably other checks too) before initiating reading and writing operations.

**Parameters**

*v* true if allowed (default is true).

Implements **Arc::DataPoint** (p. 115).

**6.76.2.13 virtual void Arc::DataPointIndex::SetMeta ( const DataPoint & *p* ) [virtual]**

Copy meta information from another object.

Already defined values are not overwritten.

**Parameters**

*p* object from which information is taken.

Reimplemented from **Arc::DataPoint** (p. 115).

**6.76.2.14 virtual void Arc::DataPointIndex::SetSecure ( bool *v* ) [virtual]**

Allow/disallow heavy security during data transfer.

**Parameters**

*v* true if allowed (default depends on protocol).

Implements **Arc::DataPoint** (p. 116).

**6.76.2.15 virtual void Arc::DataPointIndex::SortLocations ( const std::string & *pattern*, const URLMap & *url\_map* ) [virtual]**

Sort locations according to the specified pattern.

**Parameters**

*pattern* a set of strings, separated by |, to match against.

Implements **Arc::DataPoint** (p. 116).

**6.76.2.16 virtual DataStatus Arc::DataPointIndex::StartReading ( DataBuffer & *buffer* ) [virtual]**

Start reading data from **URL** (p. 323).

Separate thread to transfer data will be created. No other operation can be performed while reading is in progress.

**Parameters**

*buffer* operation will use this buffer to put information into. Should not be destroyed before stop\_  
reading was called and returned.

Implements **Arc::DataPoint** (p. 116).

**6.76.2.17 virtual DataStatus Arc::DataPointIndex::StartWriting ( DataBuffer & *buffer*, DataCallback \* *space\_cb* = NULL ) [virtual]**

Start writing data to **URL** (p. 323).

Separate thread to transfer data will be created. No other operation can be performed while writing is in progress.

**Parameters**

*buffer* operation will use this buffer to get information from. Should not be destroyed before stop\_ writing was called and returned.

*space\_cb* callback which is called if there is not enough space to store data. May not implemented for all protocols.

Implements **Arc::DataPoint** (p. 116).

**6.76.2.18 virtual DataStatus Arc::DataPointIndex::StopReading ( ) [virtual]**

Stop reading.

Must be called after corresponding start\_reading method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements **Arc::DataPoint** (p. 117).

**6.76.2.19 virtual DataStatus Arc::DataPointIndex::StopWriting ( ) [virtual]**

Stop writing.

Must be called after corresponding start\_writing method, either after all data is transferred or to cancel transfer. Use buffer object to find out when data is transferred. Must return failure if any happened during transfer.

Implements **Arc::DataPoint** (p. 117).

**6.76.2.20 virtual bool Arc::DataPointIndex::WriteOutOfOrder ( ) [virtual]**

Returns true if **URL** (p. 323) can accept scattered data for \*writing\* operation.

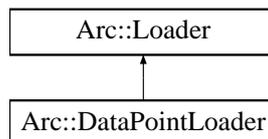
Implements **Arc::DataPoint** (p. 118).

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

- DataPointIndex.h

**6.77 Arc::DataPointLoader Class Reference**

Inheritance diagram for Arc::DataPointLoader:

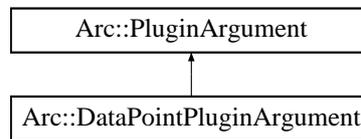


The documentation for this class was generated from the following file:

- DataPoint.h

## 6.78 Arc::DataPointPluginArgument Class Reference

Inheritance diagram for Arc::DataPointPluginArgument:



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

- DataPoint.h

## 6.79 Arc::DataSourceType Class Reference

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

- JobDescription.h

## 6.80 Arc::DataSpeed Class Reference

Keeps track of average and instantaneous transfer speed.

```
#include <DataSpeed.h>
```

### Public Member Functions

- **DataSpeed** (time\_t base=DATASPEED\_AVERAGING\_PERIOD)
- **DataSpeed** (unsigned long long int min\_speed, time\_t min\_speed\_time, unsigned long long int min\_average\_speed, time\_t max\_inactivity\_time, time\_t base=DATASPEED\_AVERAGING\_PERIOD)
- **~DataSpeed** (void)
- void **verbose** (bool val)
- void **verbose** (const std::string &prefix)
- bool **verbose** (void)
- void **set\_min\_speed** (unsigned long long int min\_speed, time\_t min\_speed\_time)
- void **set\_min\_average\_speed** (unsigned long long int min\_average\_speed)
- void **set\_max\_inactivity\_time** (time\_t max\_inactivity\_time)
- time\_t **get\_max\_inactivity\_time** ()
- void **set\_base** (time\_t base\_=DATASPEED\_AVERAGING\_PERIOD)
- void **set\_max\_data** (unsigned long long int max=0)
- void **set\_progress\_indicator** (show\_progress\_t func=NULL)
- void **reset** (void)
- bool **transfer** (unsigned long long int n=0)
- void **hold** (bool disable)
- bool **min\_speed\_failure** ()

- bool `min_average_speed_failure` ()
- bool `max_inactivity_time_failure` ()
- unsigned long long int `transferred_size` (void)

### 6.80.1 Detailed Description

Keeps track of average and instantaneous transfer speed. Also detects data transfer inactivity and other transfer timeouts.

### 6.80.2 Constructor & Destructor Documentation

#### 6.80.2.1 `Arc::DataSpeed::DataSpeed ( time_t base = DATASPEED_AVERAGING_PERIOD )`

Constructor

##### Parameters

*base* time period used to average values (default 1 minute).

#### 6.80.2.2 `Arc::DataSpeed::DataSpeed ( unsigned long long int min_speed, time_t min_speed_time, unsigned long long int min_average_speed, time_t max_inactivity_time, time_t base = DATASPEED_AVERAGING_PERIOD )`

Constructor

##### Parameters

*base* time period used to average values (default 1 minute).

*min\_speed* minimal allowed speed (Butes per second). If speed drops and holds below threshold for *min\_speed\_time*\_seconds error is triggered.

*min\_speed\_time*

*min\_average\_speed*\_ minimal average speed (Bytes per second) to trigger error. Averaged over whole current transfer time.

*max\_inactivity\_time* - if no data is passing for specified amount of time (seconds), error is triggered.

### 6.80.3 Member Function Documentation

#### 6.80.3.1 `void Arc::DataSpeed::hold ( bool disable )`

Turn off speed control.

##### Parameters

*disable* true to turn off.

#### 6.80.3.2 `void Arc::DataSpeed::set_base ( time_t base_ = DATASPEED_AVERAGING_PERIOD )`

Set averaging time period.

##### Parameters

*base* time period used to average values (default 1 minute).

**6.80.3.3 void Arc::DataSpeed::set\_max\_data ( unsigned long long int *max* = 0 )**

Set amount of data to be transferred. Used in verbose messages.

**Parameters**

*max* amount of data in bytes.

**6.80.3.4 void Arc::DataSpeed::set\_max\_inactivity\_time ( time\_t *max\_inactivity\_time* )**

Set inactivity timeout.

**Parameters**

*max\_inactivity\_time* - if no data is passing for specified amount of time (seconds), error is triggered.

**6.80.3.5 void Arc::DataSpeed::set\_min\_average\_speed ( unsigned long long int *min\_average\_speed* )**

Set minimal average speed.

**Parameters**

*min\_average\_speed* - minimal average speed (Bytes per second) to trigger error. Averaged over whole current transfer time.

**6.80.3.6 void Arc::DataSpeed::set\_min\_speed ( unsigned long long int *min\_speed*, time\_t *min\_speed\_time* )**

Set minimal allowed speed.

**Parameters**

*min\_speed* minimal allowed speed (Bytes per second). If speed drops and holds below threshold for *min\_speed\_time* seconds error is triggered.

*min\_speed\_time*

**6.80.3.7 void Arc::DataSpeed::set\_progress\_indicator ( show\_progress\_t *func* = *NULL* )**

Specify which external function will print verbose messages. If not specified internal one is used.

**Parameters**

*pointer* to function which prints information.

**6.80.3.8 bool Arc::DataSpeed::transfer ( unsigned long long int *n* = 0 )**

Inform object, about amount of data has been transferred. All errors are triggered by this method. To make them work application must call this method periodically even with zero value.

**Parameters**

*n* amount of data transferred (bytes).

**6.80.3.9 void Arc::DataSpeed::verbose ( bool *val* )**

Activate printing information about current time speeds, amount of transferred data.

**6.80.3.10 void Arc::DataSpeed::verbose ( const std::string & *prefix* )**

Print information about current speed and amount of data.

**Parameters**

'*prefix*' add this string at the beginning of every string.

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

- DataSpeed.h

**6.81 Arc::DataStagingType Class Reference**

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

- JobDescription.h

**6.82 Arc::DataStatus Class Reference**

```
#include <DataStatus.h>
```

**Public Types**

- enum **DataStatusType** {  
**Success** = 0, **ReadAcquireError** = 1, **WriteAcquireError** = 2, **ReadResolveError** = 3,  
**WriteResolveError** = 4, **ReadStartError** = 5, **WriteStartError** = 6, **ReadError** = 7,  
**WriteError** = 8, **TransferError** = 9, **ReadStopError** = 10, **WriteStopError** = 11,  
**PreRegisterError** = 12, **PostRegisterError** = 13, **UnregisterError** = 14, **CacheError** = 15,  
**CredentialsExpiredError** = 16, **DeleteError** = 17, **NoLocationError** = 18, **LocationAlreadyExistsError** = 19,  
**NotSupportedForDirectDataPointsError** = 20, **UnimplementedError** = 21, **IsReadingError** = 22,  
**IsWritingError** = 23,  
**CheckError** = 24, **ListError** = 25, **StatError** = 27, **NotInitializedError** = 29,  
**SystemError** = 30, **StageError** = 31, **InconsistentMetadataError** = 32, **SuccessCached** = 33,  
**UnknownError** = 34 }

**6.82.1 Detailed Description**

A class to be used for return types of all major data handling methods. It describes the outcome of the method.

## 6.82.2 Member Enumeration Documentation

### 6.82.2.1 enum Arc::DataStatus::DataStatusType

#### Enumerator:

- Success* Operation completed successfully.
- ReadAcquireError* Source is bad **URL** (p. 323) or can't be used due to some reason.
- WriteAcquireError* Destination is bad **URL** (p. 323) or can't be used due to some reason.
- ReadResolveError* Resolving of index service **URL** (p. 323) for source failed.
- WriteResolveError* Resolving of index service **URL** (p. 323) for destination failed.
- ReadStartError* Can't read from source.
- WriteStartError* Can't write to destination.
- ReadError* Failed while reading from source.
- WriteError* Failed while writing to destination.
- TransferError* Failed while transferring data (mostly timeout).
- ReadStopError* Failed while finishing reading from source.
- WriteStopError* Failed while finishing writing to destination.
- PreRegisterError* First stage of registration of index service **URL** (p. 323) failed.
- PostRegisterError* Last stage of registration of index service **URL** (p. 323) failed.
- UnregisterError* Unregistration of index service **URL** (p. 323) failed.
- CacheError* Error in caching procedure.
- CredentialsExpiredError* Error due to provided credentials are expired.
- DeleteError* Error deleting location or **URL** (p. 323).
- NoLocationError* No valid location available.
- LocationAlreadyExistsError* No valid location available.
- NotSupportedForDirectDataPointsError* Operation has no sense for this kind of **URL** (p. 323).
- UnimplementedError* Feature is unimplemented.
- IsReadingError* **DataPoint** (p. 108) is already reading.
- IsWritingError* **DataPoint** (p. 108) is already writing.
- CheckError* Access check failed.
- ListError* File listing failed.
- StatError* File/dir stating failed.
- NotInitializedError* Object initialization failed.
- SystemError* Error in OS.
- StageError* Staging error.
- InconsistentMetadataError* Inconsistent metadata.
- SuccessCached* Data was already cached.
- UnknownError* Undefined.

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

- DataStatus.h

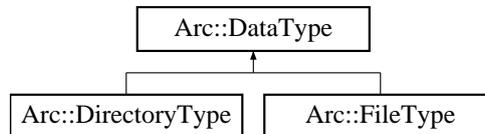
## 6.83 Arc::DataTargetType Class Reference

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

- JobDescription.h

## 6.84 Arc::DataType Class Reference

Inheritance diagram for Arc::DataType:

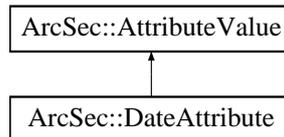


The documentation for this class was generated from the following file:

- JobDescription.h

## 6.85 ArcSec::DateAttribute Class Reference

Inheritance diagram for ArcSec::DateAttribute:



### Public Member Functions

- virtual bool **equal** (AttributeValue \*other, bool check\_id=true)
- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.85.1 Member Function Documentation

#### 6.85.1.1 virtual std::string ArcSec::DateAttribute::encode ( ) [virtual]

encode the value in a string format

Implements ArcSec::AttributeValue (p. 57).

**6.85.1.2** `virtual bool ArcSec::DateAttribute::equal ( AttributeValue * value, bool check_id = true ) [virtual]`

Evaluate whether "this" equals to the parameter value

Implements `ArcSec::AttributeValue` (p. 58).

**6.85.1.3** `virtual std::string ArcSec::DateAttribute::getId ( ) [inline, virtual]`

Get the AttributeId of the <Attribute>

Implements `ArcSec::AttributeValue` (p. 58).

**6.85.1.4** `virtual std::string ArcSec::DateAttribute::getType ( ) [inline, virtual]`

Get the DataType of the <Attribute>

Implements `ArcSec::AttributeValue` (p. 58).

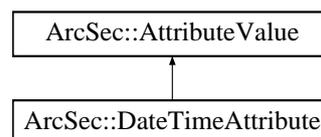
The documentation for this class was generated from the following file:

- DateTimeAttribute.h

## 6.86 ArcSec::DateTimeAttribute Class Reference

```
#include <DateTimeAttribute.h>
```

Inheritance diagram for ArcSec::DateTimeAttribute:



### Public Member Functions

- virtual bool **equal** (`AttributeValue *other`, bool check\_id=true)
- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.86.1 Detailed Description

Format: YYYYMMDDHHMMSSZ Day Month DD HH:MM:SS YYYY YYYY-MM-DD HH:MM:SS  
 YYYY-MM-DDTHH:MM:SS+HH:MM YYYY-MM-DDTHH:MM:SSZ

## 6.86.2 Member Function Documentation

### 6.86.2.1 virtual std::string ArcSec::DateTimeAttribute::encode ( ) [virtual]

encode the value in a string format

Implements `ArcSec::AttributeValue` (p. 57).

### 6.86.2.2 virtual bool ArcSec::DateTimeAttribute::equal ( AttributeValue \* value, bool check\_id = true ) [virtual]

Evaluate whether "this" equal to the parameter value

Implements `ArcSec::AttributeValue` (p. 58).

### 6.86.2.3 virtual std::string ArcSec::DateTimeAttribute::getId ( ) [inline, virtual]

Get the AttributeId of the <Attribute>

Implements `ArcSec::AttributeValue` (p. 58).

### 6.86.2.4 virtual std::string ArcSec::DateTimeAttribute::getType ( ) [inline, virtual]

Get the DataType of the <Attribute>

Implements `ArcSec::AttributeValue` (p. 58).

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

- DateTimeAttribute.h

## 6.87 Arc::DBranch Class Reference

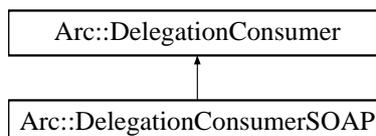
The documentation for this class was generated from the following file:

- DBranch.h

## 6.88 Arc::DelegationConsumer Class Reference

```
#include <DelegationInterface.h>
```

Inheritance diagram for `Arc::DelegationConsumer`:



## Public Member Functions

- **DelegationConsumer** (void)
- **DelegationConsumer** (const std::string &content)
- const std::string & **ID** (void)
- bool **Backup** (std::string &content)
- bool **Restore** (const std::string &content)
- bool **Request** (std::string &content)
- bool **Acquire** (std::string &content)
- bool **Acquire** (std::string &content, std::string &identity)

## Protected Member Functions

- bool **Generate** (void)
- void **LogError** (void)

### 6.88.1 Detailed Description

A consumer of delegated X509 credentials. During delegation procedure this class acquires delegated credentials aka proxy - certificate, private key and chain of previous certificates. Delegation procedure consists of calling **Request()** (p. 138) method for generating certificate request followed by call to **Acquire()** (p. 137) method for making complete credentials from certificate chain.

### 6.88.2 Constructor & Destructor Documentation

#### 6.88.2.1 Arc::DelegationConsumer::DelegationConsumer ( void )

Creates object with new private key

#### 6.88.2.2 Arc::DelegationConsumer::DelegationConsumer ( const std::string & content )

Creates object with provided private key

### 6.88.3 Member Function Documentation

#### 6.88.3.1 bool Arc::DelegationConsumer::Acquire ( std::string & content )

Ads private key into certificates chain in 'content' On exit content contains complete delegated credentials.

#### 6.88.3.2 bool Arc::DelegationConsumer::Acquire ( std::string & content, std::string & identity )

Includes the functionality in Acquire(content); plus extracting the credential identity

#### 6.88.3.3 bool Arc::DelegationConsumer::Backup ( std::string & content )

Stores content of this object into a string

**6.88.3.4** `bool Arc::DelegationConsumer::Generate ( void ) [protected]`

Private key

**6.88.3.5** `const std::string& Arc::DelegationConsumer::ID ( void )`

Return identifier of this object - not implemented

**6.88.3.6** `void Arc::DelegationConsumer::LogError ( void ) [protected]`

Creates private key

**6.88.3.7** `bool Arc::DelegationConsumer::Request ( std::string & content )`

Make X509 certificate request from internal private key

**6.88.3.8** `bool Arc::DelegationConsumer::Restore ( const std::string & content )`

Restores content of object from string

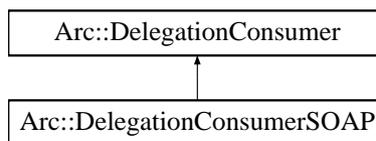
The documentation for this class was generated from the following file:

- DelegationInterface.h

**6.89 Arc::DelegationConsumerSOAP Class Reference**

```
#include <DelegationInterface.h>
```

Inheritance diagram for Arc::DelegationConsumerSOAP:

**Public Member Functions**

- **DelegationConsumerSOAP** (void)
- **DelegationConsumerSOAP** (const std::string &content)
- bool **DelegateCredentialsInit** (const std::string &id, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool **UpdateCredentials** (std::string &credentials, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool **UpdateCredentials** (std::string &credentials, std::string &identity, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool **DelegatedToken** (std::string &credentials, XMLNode token)

### 6.89.1 Detailed Description

This class extends **DelegationConsumer** (p. 136) to support SOAP message exchange. Implements WS interface <http://www.nordugrid.org/schemas/delegation> described in delegation.wsdl.

### 6.89.2 Constructor & Destructor Documentation

#### 6.89.2.1 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP ( void )

Creates object with new private key

#### 6.89.2.2 Arc::DelegationConsumerSOAP::DelegationConsumerSOAP ( const std::string & content )

Creates object with specified private key

### 6.89.3 Member Function Documentation

#### 6.89.3.1 bool Arc::DelegationConsumerSOAP::DelegateCredentialsInit ( const std::string & id, const SOAPEnvelope & in, SOAPEnvelope & out )

Process SOAP message which starts delagation. Generated message in 'out' is meant to be sent back to DelagationProviderSOAP. Argument 'id' contains identifier of procedure and is used only to produce SOAP message.

#### 6.89.3.2 bool Arc::DelegationConsumerSOAP::DelegatedToken ( std::string & credentials, XMLNode token )

Similar to UpdateCredentials but takes only DelegatedToken XML element

#### 6.89.3.3 bool Arc::DelegationConsumerSOAP::UpdateCredentials ( std::string & credentials, std::string & identity, const SOAPEnvelope & in, SOAPEnvelope & out )

Includes the functionality in above UpdateCredentials method; plus extracting the credential identity

#### 6.89.3.4 bool Arc::DelegationConsumerSOAP::UpdateCredentials ( std::string & credentials, const SOAPEnvelope & in, SOAPEnvelope & out )

Accepts delegated credentials. Process 'in' SOAP message and stores full proxy credentials in 'credentials'. 'out' message is generated for sending to DelagationProviderSOAP.

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

- DelegationInterface.h

## 6.90 Arc::DelegationContainerSOAP Class Reference

```
#include <DelegationInterface.h>
```

## Public Member Functions

- bool **DelegateCredentialsInit** (const SOAPEnvelope &in, SOAPEnvelope &out)
- bool **UpdateCredentials** (std::string &credentials, const SOAPEnvelope &in, SOAPEnvelope &out)
- bool **DelegatedToken** (std::string &credentials, XMLNode token)

## Protected Attributes

- int **max\_size\_**
- int **max\_duration\_**
- int **max\_usage\_**
- bool **context\_lock\_**
- bool **restricted\_**

### 6.90.1 Detailed Description

Manages multiple delegated credentials. Delegation consumers are created automatically with DelegateCredentialsInit method up to max\_size\_ and assigned unique identifier. It's methods are similar to those of **DelegationConsumerSOAP** (p. 138) with identifier included in SOAP message used to route execution to one of managed **DelegationConsumerSOAP** (p. 138) instances.

### 6.90.2 Member Function Documentation

**6.90.2.1** bool Arc::DelegationContainerSOAP::DelegateCredentialsInit ( const SOAPEnvelope & *in*, SOAPEnvelope & *out* )

See **DelegationConsumerSOAP::DelegateCredentialsInit** (p. 139)

**6.90.2.2** bool Arc::DelegationContainerSOAP::DelegatedToken ( std::string & *credentials*, XMLNode *token* )

See **DelegationConsumerSOAP::DelegatedToken** (p. 139)

**6.90.2.3** bool Arc::DelegationContainerSOAP::UpdateCredentials ( std::string & *credentials*, const SOAPEnvelope & *in*, SOAPEnvelope & *out* )

See **DelegationConsumerSOAP::UpdateCredentials** (p. 139)

### 6.90.3 Field Documentation

**6.90.3.1** bool Arc::DelegationContainerSOAP::context\_lock\_ [protected]

If true delegation consumer is deleted when connection context is destroyed

**6.90.3.2** int Arc::DelegationContainerSOAP::max\_duration\_ [protected]

Lifetime of unused delegation consumer

**6.90.3.3** `int Arc::DelegationContainerSOAP::max_size_` `[protected]`

Max. number of delegation consumers

**6.90.3.4** `int Arc::DelegationContainerSOAP::max_usage_` `[protected]`

Max. times same delegation consumer may accept credentials

**6.90.3.5** `bool Arc::DelegationContainerSOAP::restricted_` `[protected]`

If true all delegation phases must be performed by same identity

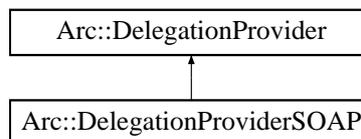
The documentation for this class was generated from the following file:

- DelegationInterface.h

**6.91 Arc::DelegationProvider Class Reference**

```
#include <DelegationInterface.h>
```

Inheritance diagram for Arc::DelegationProvider:

**Public Member Functions**

- **DelegationProvider** (const std::string &credentials)
- **DelegationProvider** (const std::string &cert\_file, const std::string &key\_file, std::istream \*inpwd=NULL)
- std::string **Delegate** (const std::string &request, const DelegationRestrictions &restrictions=DelegationRestrictions())

**6.91.1 Detailed Description**

A provider of delegated credentials. During delegation procedure this class generates new credential to be used in proxy/delegated credential.

**6.91.2 Constructor & Destructor Documentation****6.91.2.1 Arc::DelegationProvider::DelegationProvider ( const std::string & *credentials* )**

Creates instance from provided credentials. Credentials are used to sign delegated credentials. Arguments should contain PEM-encoded certificate, private key and optionally certificates chain.

### 6.91.2.2 `Arc::DelegationProvider::DelegationProvider` ( `const std::string & cert_file`, `const std::string & key_file`, `std::istream * inpwd = NULL` )

Creates instance from provided credentials. Credentials are used to sign delegated credentials. Arguments should contain filesystem path to PEM-encoded certificate and private key. Optionally `cert_file` may contain certificates chain.

## 6.91.3 Member Function Documentation

### 6.91.3.1 `std::string Arc::DelegationProvider::Delegate` ( `const std::string & request`, `const DelegationRestrictions & restrictions = DelegationRestrictions()` )

Perform delegation. Takes X509 certificate request and creates proxy credentials excluding private key. Result is then to be fed into `DelegationConsumer::Acquire` (p. 137)

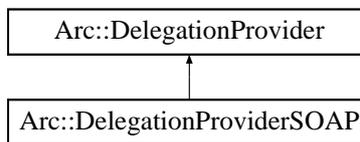
The documentation for this class was generated from the following file:

- `DelegationInterface.h`

## 6.92 `Arc::DelegationProviderSOAP` Class Reference

```
#include <DelegationInterface.h>
```

Inheritance diagram for `Arc::DelegationProviderSOAP`:



### Public Member Functions

- `DelegationProviderSOAP` (`const std::string &credentials`)
- `DelegationProviderSOAP` (`const std::string &cert_file`, `const std::string &key_file`, `std::istream *inpwd=NULL`)
- `bool DelegateCredentialsInit` (`MCCInterface &mcc_interface`, `MessageContext *context`)
- `bool DelegateCredentialsInit` (`MCCInterface &mcc_interface`, `MessageAttributes *attributes_in`, `MessageAttributes *attributes_out`, `MessageContext *context`)
- `bool UpdateCredentials` (`MCCInterface &mcc_interface`, `MessageContext *context`, `const DelegationRestrictions &restrictions=DelegationRestrictions()`)
- `bool UpdateCredentials` (`MCCInterface &mcc_interface`, `MessageAttributes *attributes_in`, `MessageAttributes *attributes_out`, `MessageContext *context`, `const DelegationRestrictions &restrictions=DelegationRestrictions()`)
- `bool DelegatedToken` (`XMLNode parent`)
- `const std::string & ID` (`void`)

## 6.92.1 Detailed Description

Extension of **DelegationProvider** (p. 141) with SOAP exchange interface. This class is also a temporary container for intermediate information used during delegation procedure.

## 6.92.2 Constructor & Destructor Documentation

### 6.92.2.1 Arc::DelegationProviderSOAP::DelegationProviderSOAP ( const std::string & *credentials* )

Creates instance from provided credentials. Credentials are used to sign delegated credentials.

### 6.92.2.2 Arc::DelegationProviderSOAP::DelegationProviderSOAP ( const std::string & *cert\_file*, const std::string & *key\_file*, std::istream \* *inpwd* = *NULL* )

Creates instance from provided credentials. Credentials are used to sign delegated credentials. Arguments should contain filesystem path to PEM-encoded certificate and private key. Optionally *cert\_file* may contain certificates chain.

## 6.92.3 Member Function Documentation

### 6.92.3.1 bool Arc::DelegationProviderSOAP::DelegateCredentialsInit ( MCCInterface & *mcc\_interface*, MessageContext \* *context* )

Performs DelegateCredentialsInit SOAP operation. As result request for delegated credentials is received by this instance and stored internally. Call to UpdateCredentials should follow.

### 6.92.3.2 bool Arc::DelegationProviderSOAP::DelegateCredentialsInit ( MCCInterface & *mcc\_interface*, MessageAttributes \* *attributes\_in*, MessageAttributes \* *attributes\_out*, MessageContext \* *context* )

Extended version of **DelegateCredentialsInit(MCCInterface&,MessageContext\*)** (p. 143). Additionally takes attributes for request and response message to make fine control on message processing possible.

### 6.92.3.3 bool Arc::DelegationProviderSOAP::DelegatedToken ( XMLNode *parent* )

Generates DelegatedToken element. Element is created as child of provided XML element and contains structure described in delegation.wsdl.

### 6.92.3.4 const std::string& Arc::DelegationProviderSOAP::ID ( void ) [inline]

Returns the identifier by service accepting delegated credentials. This identifier may then be used to refer to credentials stored at service.

**6.92.3.5** `bool Arc::DelegationProviderSOAP::UpdateCredentials ( MCCInterface & mcc_interface, MessageAttributes * attributes_in, MessageAttributes * attributes_out, MessageContext * context, const DelegationRestrictions & restrictions = DelegationRestrictions () )`

Extended version of UpdateCredentials(MCCInterface&,MessageContext\*). Additionally takes attributes for request and response message to make fine control on message processing possible.

**6.92.3.6** `bool Arc::DelegationProviderSOAP::UpdateCredentials ( MCCInterface & mcc_interface, MessageContext * context, const DelegationRestrictions & restrictions = DelegationRestrictions () )`

Performs UpdateCredentials SOAP operation. This concludes delegation procedure and passes delegated credentials to **DelegationConsumerSOAP** (p. 138) instance.

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

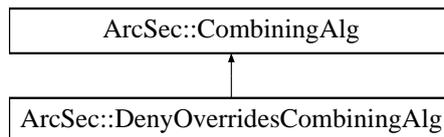
- DelegationInterface.h

## 6.93 ArcSec::DenyOverridesCombiningAlg Class Reference

Implement the "Deny-Overrides" algorithm.

```
#include <DenyOverridesAlg.h>
```

Inheritance diagram for ArcSec::DenyOverridesCombiningAlg:



### Public Member Functions

- virtual Result **combine** (EvaluationCtx \*ctx, std::list< Policy \* > policies)
- virtual const std::string & **getalgId** (void) const

### 6.93.1 Detailed Description

Implement the "Deny-Overrides" algorithm. Deny-Overrides, scans the policy set which is given as the parameters of "combine" method, if gets "deny" result from any policy, then stops scanning and gives "deny" as result, otherwise gives "permit".

## 6.93.2 Member Function Documentation

**6.93.2.1** `virtual Result ArcSec::DenyOverridesCombiningAlg::combine ( EvaluationCtx * ctx, std::list< Policy * > policies ) [virtual]`

If there is one policy which return negative evaluation result, then omit the other policies and return DECISION\_DENY

### Parameters

*ctx* This object contains request information which will be used to evaluated against policy.

*policies* This is a container which contains policy objects.

### Returns

The combined result according to the algorithm.

Implements `ArcSec::CombiningAlg` (p. 76).

**6.93.2.2** `virtual const std::string& ArcSec::DenyOverridesCombiningAlg::getalgId ( void ) const [inline, virtual]`

Get the identifier

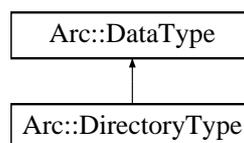
Implements `ArcSec::CombiningAlg` (p. 76).

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

- DenyOverridesAlg.h

## 6.94 Arc::DirectoryType Class Reference

Inheritance diagram for Arc::DirectoryType:



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

- JobDescription.h

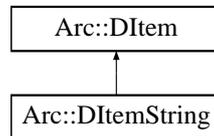
## 6.95 Arc::DiskSpaceRequirementType Class Reference

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

- JobDescription.h

## 6.96 Arc::DItem Class Reference

Inheritance diagram for Arc::DItem:

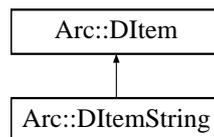


The documentation for this class was generated from the following file:

- DBranch.h

## 6.97 Arc::DItemString Class Reference

Inheritance diagram for Arc::DItemString:

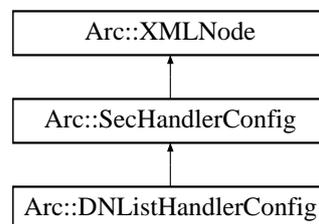


The documentation for this class was generated from the following file:

- DBranch.h

## 6.98 Arc::DNListHandlerConfig Class Reference

Inheritance diagram for Arc::DNListHandlerConfig:



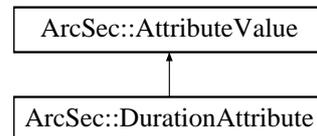
The documentation for this class was generated from the following file:

- ClientInterface.h

## 6.99 ArcSec::DurationAttribute Class Reference

```
#include <DateTimeAttribute.h>
```

Inheritance diagram for ArcSec::DurationAttribute:



### Public Member Functions

- virtual bool **equal** (**AttributeValue** \*other, bool check\_id=true)
- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

#### 6.99.1 Detailed Description

Formate: P??Y??M??DT??H??M??S

#### 6.99.2 Member Function Documentation

##### 6.99.2.1 virtual std::string ArcSec::DurationAttribute::encode ( ) [virtual]

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 57).

##### 6.99.2.2 virtual bool ArcSec::DurationAttribute::equal ( AttributeValue \* value, bool check\_id = true ) [virtual]

Evaluate whether "this" equale to the parameter value

Implements **ArcSec::AttributeValue** (p. 58).

##### 6.99.2.3 virtual std::string ArcSec::DurationAttribute::getId ( ) [inline, virtual]

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 58).

##### 6.99.2.4 virtual std::string ArcSec::DurationAttribute::getType ( ) [inline, virtual]

Get the DataType of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 58).

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

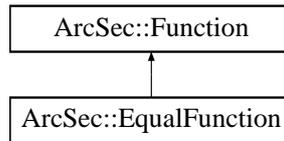
- DateTimeAttribute.h

## 6.100 ArcSec::EqualFunction Class Reference

Evaluate whether the two values are equal.

```
#include <EqualFunction.h>
```

Inheritance diagram for ArcSec::EqualFunction:



### Public Member Functions

- virtual `AttributeValue * evaluate (AttributeValue *arg0, AttributeValue *arg1, bool check_id=true)`
- virtual `std::list< AttributeValue * > evaluate (std::list< AttributeValue * > args, bool check_id=true)`

### Static Public Member Functions

- static `std::string getFunctionName (std::string datatype)`

#### 6.100.1 Detailed Description

Evaluate whether the two values are equal.

#### 6.100.2 Member Function Documentation

**6.100.2.1** `virtual AttributeValue* ArcSec::EqualFunction::evaluate ( AttributeValue * arg0, AttributeValue * arg1, bool check_id = true ) [virtual]`

Evaluate two `AttributeValue` (p. 56) objects, and return one `AttributeValue` (p. 56) object

Implements `ArcSec::Function` (p. 167).

**6.100.2.2** `virtual std::list<AttributeValue*> ArcSec::EqualFunction::evaluate ( std::list< AttributeValue * > args, bool check_id = true ) [virtual]`

Evaluate a list of `AttributeValue` (p. 56) objects, and return a list of Attribute objects

Implements `ArcSec::Function` (p. 167).

### 6.100.2.3 static std::string ArcSec::EqualFunction::getFunctionName ( std::string *datatype* ) [static]

help function to get the FunctionName

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

- EqualFunction.h

## 6.101 ArcSec::EvalResult Struct Reference

Struct to record the xml node and effect, which will be used by **Evaluator** (p. 150) to get the information about which rule/policy(in xmlnode) is satisfied.

```
#include <Result.h>
```

### 6.101.1 Detailed Description

Struct to record the xml node and effect, which will be used by **Evaluator** (p. 150) to get the information about which rule/policy(in xmlnode) is satisfied.

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

- Result.h

## 6.102 ArcSec::EvaluationCtx Class Reference

**EvaluationCtx** (p. 149), in charge of storing some context information for.

```
#include <EvaluationCtx.h>
```

### Public Member Functions

- **EvaluationCtx** (Request \*request)

### 6.102.1 Detailed Description

**EvaluationCtx** (p. 149), in charge of storing some context information for.

### 6.102.2 Constructor & Destructor Documentation

#### 6.102.2.1 ArcSec::EvaluationCtx::EvaluationCtx ( Request \* *request* ) [inline]

Construct a new **EvaluationCtx** (p. 149) based on the given request

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

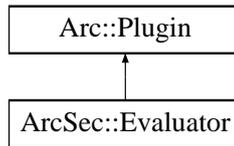
- EvaluationCtx.h

## 6.103 ArcSec::Evaluator Class Reference

Interface for policy evaluation. Execute the policy evaluation, based on the request and policy.

```
#include <Evaluator.h>
```

Inheritance diagram for ArcSec::Evaluator:



### Public Member Functions

- virtual **Response** \* **evaluate** (**Request** \*request)=0
- virtual **Response** \* **evaluate** (const **Source** &request)=0
- virtual **Response** \* **evaluate** (**Request** \*request, const **Source** &policy)=0
- virtual **Response** \* **evaluate** (const **Source** &request, const **Source** &policy)=0
- virtual **Response** \* **evaluate** (**Request** \*request, **Policy** \*policyobj)=0
- virtual **Response** \* **evaluate** (const **Source** &request, **Policy** \*policyobj)=0
- virtual **AttributeFactory** \* **getAttrFactory** ()=0
- virtual **FnFactory** \* **getFnFactory** ()=0
- virtual **AlgFactory** \* **getAlgFactory** ()=0
- virtual void **addPolicy** (const **Source** &policy, const std::string &id="")=0
- virtual void **addPolicy** (**Policy** \*policy, const std::string &id="")=0
- virtual void **setCombiningAlg** (**EvaluatorCombiningAlg** alg)=0
- virtual void **setCombiningAlg** (**CombiningAlg** \*alg=NULL)=0
- virtual const char \* **getName** (void) const =0

### Protected Member Functions

- virtual **Response** \* **evaluate** (**EvaluationCtx** \*ctx)=0

#### 6.103.1 Detailed Description

Interface for policy evaluation. Execute the policy evaluation, based on the request and policy.

#### 6.103.2 Member Function Documentation

- 6.103.2.1** virtual void ArcSec::Evaluator::addPolicy ( const **Source** & *policy*, const std::string & *id* = "" ) [pure virtual]

Add policy from specified source to the evaluator. **Policy** (p. 254) will be marked with id.

**6.103.2.2** `virtual void ArcSec::Evaluator::addPolicy ( Policy * policy, const std::string & id = "" ) [pure virtual]`

Add policy to the evaluator. **Policy** (p. 254) will be marked with *id*. The policy object is taken over by this instance and will be destroyed in destructor.

**6.103.2.3** `virtual Response* ArcSec::Evaluator::evaluate ( const Source & request, const Source & policy ) [pure virtual]`

Evaluate the request from specified source against the policy from specified source. In some implementations all of the existing policies inside the evaluator may be destroyed by this method.

**6.103.2.4** `virtual Response* ArcSec::Evaluator::evaluate ( Request * request ) [pure virtual]`

Evaluates the request by using a **Request** (p. 263) object. Evaluation is done till at least one of policies is satisfied.

**6.103.2.5** `virtual Response* ArcSec::Evaluator::evaluate ( Request * request, Policy * policyobj ) [pure virtual]`

Evaluate the specified request against the specified policy. In some implementations all of the existing policy inside the evaluator may be destroyed by this method.

**6.103.2.6** `virtual Response* ArcSec::Evaluator::evaluate ( const Source & request ) [pure virtual]`

Evaluates the request by using a specified source

**6.103.2.7** `virtual Response* ArcSec::Evaluator::evaluate ( Request * request, const Source & policy ) [pure virtual]`

Evaluate the specified request against the policy from specified source. In some implementations all of the existing policies inside the evaluator may be destroyed by this method.

**6.103.2.8** `virtual Response* ArcSec::Evaluator::evaluate ( const Source & request, Policy * policyobj ) [pure virtual]`

Evaluate the request from specified source against the specified policy. In some implementations all of the existing policies inside the evaluator may be destroyed by this method.

**6.103.2.9** `virtual Response* ArcSec::Evaluator::evaluate ( EvaluationCtx * ctx ) [protected, pure virtual]`

Evaluate the request by using the **EvaluationCtx** (p. 149) object (which includes the information about request). The *ctx* is destroyed inside this method (why?!?!?).

**6.103.2.10 virtual AlgFactory\* ArcSec::Evaluator::getAlgFactory ( ) [pure virtual]**

Get the **AlgFactory** (p. 48) object

**6.103.2.11 virtual AttributeFactory\* ArcSec::Evaluator::getAttrFactory ( ) [pure virtual]**

Get the **AttributeFactory** (p. 52) object

**6.103.2.12 virtual FnFactory\* ArcSec::Evaluator::getFnFactory ( ) [pure virtual]**

Get the **FnFactory** (p. 165) object

**6.103.2.13 virtual const char\* ArcSec::Evaluator::getName ( void ) const [pure virtual]**

Get the name of this evaluator

**6.103.2.14 virtual void ArcSec::Evaluator::setCombiningAlg ( EvaluatorCombiningAlg alg ) [pure virtual]**

Specifies one of simple combining algorithms. In case of multiple policies their results will be combined using this algorithm.

**6.103.2.15 virtual void ArcSec::Evaluator::setCombiningAlg ( CombiningAlg \* alg = NULL ) [pure virtual]**

Specifies loadable combining algorithms. In case of multiple policies their results will be combined using this algorithm. To switch to simple algorithm specify NULL argument.

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

- Evaluator.h

## 6.104 ArcSec::EvaluatorContext Class Reference

Context for evaluator. It includes the factories which will be used to create related objects.

```
#include <Evaluator.h>
```

### Public Member Functions

- **operator AttributeFactory \* ()**
- **operator FnFactory \* ()**
- **operator AlgFactory \* ()**

#### 6.104.1 Detailed Description

Context for evaluator. It includes the factories which will be used to create related objects.

## 6.104.2 Member Function Documentation

### 6.104.2.1 ArcSec::EvaluatorContext::operator AlgFactory \* ( ) [inline]

Returns associated **AlgFactory** (p. 48) object

### 6.104.2.2 ArcSec::EvaluatorContext::operator AttributeFactory \* ( ) [inline]

Returns associated **AttributeFactory** (p. 52) object

### 6.104.2.3 ArcSec::EvaluatorContext::operator FnFactory \* ( ) [inline]

Returns associated **FnFactory** (p. 165) object

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

- Evaluator.h

## 6.105 ArcSec::EvaluatorLoader Class Reference

**EvaluatorLoader** (p. 153) is implemented as a helper class for loading different **Evaluator** (p. 150) objects, like ArcEvaluator.

```
#include <EvaluatorLoader.h>
```

### Public Member Functions

- **Evaluator** \* **getEvaluator** (const std::string &classname)
- **Evaluator** \* **getEvaluator** (const **Policy** \*policy)
- **Evaluator** \* **getEvaluator** (const **Request** \*request)
- **Request** \* **getRequest** (const std::string &classname, const **Source** &requestsource)
- **Request** \* **getRequest** (const **Source** &requestsource)
- **Policy** \* **getPolicy** (const std::string &classname, const **Source** &polycysource)
- **Policy** \* **getPolicy** (const **Source** &polycysource)

### 6.105.1 Detailed Description

**EvaluatorLoader** (p. 153) is implemented as a helper class for loading different **Evaluator** (p. 150) objects, like ArcEvaluator. The object loading is based on the configuration information about evaluator, including information for factory class, request, policy and evaluator itself

## 6.105.2 Member Function Documentation

### 6.105.2.1 Evaluator\* ArcSec::EvaluatorLoader::getEvaluator ( const std::string & *classname* )

Get evaluator object according to the class name

**6.105.2.2** `Evaluator* ArcSec::EvaluatorLoader::getEvaluator ( const Policy * policy )`

Get evaluator object suitable for presented policy

**6.105.2.3** `Evaluator* ArcSec::EvaluatorLoader::getEvaluator ( const Request * request )`

Get evaluator object suitable for presented request

**6.105.2.4** `Policy* ArcSec::EvaluatorLoader::getPolicy ( const Source & polycysource )`

Get proper policy object according to the policy source

**6.105.2.5** `Policy* ArcSec::EvaluatorLoader::getPolicy ( const std::string & classname, const Source & polycysource )`

Get policy object according to the class name, based on the policy source

**6.105.2.6** `Request* ArcSec::EvaluatorLoader::getRequest ( const std::string & classname, const Source & requestsource )`

Get request object according to the class name, based on the request source

**6.105.2.7** `Request* ArcSec::EvaluatorLoader::getRequest ( const Source & requestsource )`

Get request object according to the request source

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

- `EvaluatorLoader.h`

**6.106 Arc::ExecutableType Class Reference**

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

- `JobDescription.h`

**6.107 Arc::ExecutionTarget Class Reference**

`ExecutionTarget` (p. 154).

```
#include <ExecutionTarget.h>
```

**Public Member Functions**

- `ExecutionTarget ()`
- `ExecutionTarget (const ExecutionTarget &target)`
- `ExecutionTarget (const long int addrptr)`

- **ExecutionTarget** & **operator=** (const **ExecutionTarget** &target)
- **Submitter** \* **GetSubmitter** (const **UserConfig** &ucfg) const
- void **Update** (const **JobDescription** &jobdesc)
- void **Print** (bool longlist) const

## Data Fields

- std::string **ComputingShareName**
- int64\_t **MaxMainMemory**
- int64\_t **MaxVirtualMemory**
- int64\_t **MaxDiskSpace**
- std::map< **Period**, int > **FreeSlotsWithDuration**
- **Software OperatingSystem**
- std::list< **ApplicationEnvironment** > **ApplicationEnvironments**

### 6.107.1 Detailed Description

**ExecutionTarget** (p. 154). This class describe a target which accept computing jobs. All of the members contained in this class, with a few exceptions, are directly linked to attributes defined in the GLUE Specification v. 2.0 (GFD-R-P.147).

### 6.107.2 Constructor & Destructor Documentation

#### 6.107.2.1 Arc::ExecutionTarget::ExecutionTarget ( )

Create an **ExecutionTarget** (p. 154).

Default constructor to create an **ExecutionTarget** (p. 154). Takes no arguments.

#### 6.107.2.2 Arc::ExecutionTarget::ExecutionTarget ( const **ExecutionTarget** & *target* )

Create an **ExecutionTarget** (p. 154).

Copy constructor.

#### Parameters

*target* **ExecutionTarget** (p. 154) to copy.

#### 6.107.2.3 Arc::ExecutionTarget::ExecutionTarget ( const long int *addrptr* )

Create an **ExecutionTarget** (p. 154).

Copy constructor? Needed from Python?

#### Parameters

*addrptr*

### 6.107.3 Member Function Documentation

#### 6.107.3.1 Submitter\* Arc::ExecutionTarget::GetSubmitter ( const UserConfig & *ucfg* ) const

Get **Submitter** (p. 308) to the computing resource represented by the **ExecutionTarget** (p. 154).

Method which returns a specialized **Submitter** (p. 308) which can be used for submitting jobs to the computing resource represented by the **ExecutionTarget** (p. 154). In order to return the correct specialized **Submitter** (p. 308) the GridFlavour variable must be correctly set.

#### Parameters

*ucfg* **UserConfig** (p. 333) object with paths to user credentials etc.

#### 6.107.3.2 ExecutionTarget& Arc::ExecutionTarget::operator= ( const ExecutionTarget & *target* )

Create an **ExecutionTarget** (p. 154).

Assignment operator

#### Parameters

*target* is **ExecutionTarget** (p. 154) to copy.

#### 6.107.3.3 void Arc::ExecutionTarget::Print ( bool *longlist* ) const

Print the **ExecutionTarget** (p. 154) information to std::cout.

Method to print the **ExecutionTarget** (p. 154) attributes to std::cout

#### Parameters

*longlist* is true for long list printing.

#### 6.107.3.4 void Arc::ExecutionTarget::Update ( const JobDescription & *jobdesc* )

Update **ExecutionTarget** (p. 154) after succesful job submission.

Method to update the **ExecutionTarget** (p. 154) after a job succesfully has been submitted to the computing resource it represents. E.g. if a job is sent to the computing resource and is expected to enter the queue, then the WaitingJobs attribute is incremented with 1.

#### Parameters

*jobdesc* contains all information about the job submitted.

### 6.107.4 Field Documentation

#### 6.107.4.1 std::list<ApplicationEnvironment> Arc::ExecutionTarget::ApplicationEnvironments

ApplicationEnvironments.

The ApplicationEnvironments member is a list of ApplicationEnvironment's, defined in section 6.7 GLUE2.

**6.107.4.2 std::string Arc::ExecutionTarget::ComputingShareName**

ComputingShareName String 0..1.

Human-readable name. This variable represents the ComputingShare.Name attribute of GLUE2.

**6.107.4.3 std::map<Period, int> Arc::ExecutionTarget::FreeSlotsWithDuration**

FreeSlotsWithDuration std::map<Period, int>

This attribute express the number of free slots with their time limits. The keys in the std::map are the time limit (**Period** (p. 243)) for the number of free slots stored as the value (int). If no time limit has been specified for a set of free slots then the key will equal Period(LONG\_MAX).

**6.107.4.4 int64\_t Arc::ExecutionTarget::MaxDiskSpace**

MaxDiskSpace UInt64 0..1 GB.

The maximum disk space that a job is allowed use in the working; if the limit is hit, then the LRMS MAY kill the job. A negative value specifies that this member is undefined.

**6.107.4.5 int64\_t Arc::ExecutionTarget::MaxMainMemory**

MaxMainMemory UInt64 0..1 MB.

The maximum physical RAM that a job is allowed to use; if the limit is hit, then the LRMS MAY kill the job. A negative value specifies that this member is undefined.

**6.107.4.6 int64\_t Arc::ExecutionTarget::MaxVirtualMemory**

MaxVirtualMemory UInt64 0..1 MB.

The maximum total memory size (RAM plus swap) that a job is allowed to use; if the limit is hit, then the LRMS MAY kill the job. A negative value specifies that this member is undefined.

**6.107.4.7 Software Arc::ExecutionTarget::OperatingSystem**

OperatingSystem.

The OperatingSystem member is not present in GLUE2 but contains the three GLUE2 attributes OSFamily, OSName and OSVersion.

- OSFamily OSFamily\_t 1 \* The general family to which the Execution Environment operating \* system belongs.
- OSName OSName\_t 0..1 \* The specific name of the operating sytem
- OSVersion String 0..1 \* The version of the operating system, as defined by the vendor.

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

- ExecutionTarget.h

## 6.108 Arc::ExpirationReminder Class Reference

A class intended for internal use within counters.

```
#include <Counter.h>
```

### Public Member Functions

- `bool operator< (const ExpirationReminder &other) const`
- `Glib::TimeVal getExpiryTime () const`
- `Counter::IDType getReservationID () const`

### Friends

- class **Counter**

### 6.108.1 Detailed Description

A class intended for internal use within counters. This class is used for "reminder objects" that are used for automatic deallocation of self-expiring reservations.

### 6.108.2 Member Function Documentation

#### 6.108.2.1 Glib::TimeVal Arc::ExpirationReminder::getExpiryTime ( ) const

Returns the expiry time.

This method returns the expiry time of the reservation that this **ExpirationReminder** (p. 158) is associated with.

#### Returns

The expiry time.

#### 6.108.2.2 Counter::IDType Arc::ExpirationReminder::getReservationID ( ) const

Returns the identification number of the reservation.

This method returns the identification number of the self-expiring reservation that this **ExpirationReminder** (p. 158) is associated with.

#### Returns

The identification number.

#### 6.108.2.3 bool Arc::ExpirationReminder::operator< ( const **ExpirationReminder** & *other* ) const

Less than operator, compares "soonness".

This is the less than operator for the **ExpirationReminder** (p. 158) class. It compares the priority of such objects with respect to which reservation expires first. It is used when reminder objects are inserted in a priority queue in order to allways place the next reservation to expire at the top.

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

- Counter.h

## 6.109 Arc::FileCache Class Reference

```
#include <FileCache.h>
```

### Public Member Functions

- **FileCache** (std::string cache\_path, std::string id, uid\_t job\_uid, gid\_t job\_gid)
- **FileCache** (std::vector< std::string > caches, std::string id, uid\_t job\_uid, gid\_t job\_gid)
- **FileCache** (std::vector< std::string > caches, std::vector< std::string > remote\_caches, std::vector< std::string > draining\_caches, std::string id, uid\_t job\_uid, gid\_t job\_gid, int cache\_max=100, int cache\_min=100)
- **FileCache** ()
- bool **Start** (std::string url, bool &available, bool &is\_locked, bool use\_remote=true)
- bool **Stop** (std::string url)
- bool **StopAndDelete** (std::string url)
- std::string **File** (std::string url)
- bool **Link** (std::string link\_path, std::string url)
- bool **Copy** (std::string dest\_path, std::string url, bool executable=false)
- bool **Release** ()
- bool **AddDN** (std::string url, std::string DN, **Time** expiry\_time)
- bool **CheckDN** (std::string url, std::string DN)
- bool **CheckCreated** (std::string url)
- **Time GetCreated** (std::string url)
- bool **CheckValid** (std::string url)
- **Time GetValid** (std::string url)
- bool **SetValid** (std::string url, **Time** val)
- **operator bool** ()
- bool **operator==** (const **FileCache** &a)

### 6.109.1 Detailed Description

**FileCache** (p. 159) provides an interface to all cache operations to be used by external classes. An instance should be created per job, and all files within the job are managed by that instance. When it is decided a file should be downloaded to the cache, **Start**() (p. 163) should be called, so that the cache file can be prepared and locked. When a transfer has finished successfully, **Link**() (p. 162) or **Copy**() (p. 162) should be called to create a hard link to a per-job directory in the cache and then soft link, or copy the file directly to the session directory so it can be accessed from the user's job. **Stop**() (p. 163) must then be called to release any locks on the cache file.

The cache directory(ies) and the optional directory to link to when the soft-links are made are set in the global configuration file. The names of cache files are formed from a hash of the **URL** (p. 323) specified as input to the job. To ease the load on the file system, the cache files are split into subdirectories based on

the first two characters in the hash. For example the file with hash 76f11edda169848038efbd9fa3df5693 is stored in 76/f11edda169848038efbd9fa3df5693. A cache filename can be found by passing the **URL** (p. 323) to `Find()`. For more information on the structure of the cache, see the Grid Manager Administration Guide.

A metadata file with the `.meta` suffix is stored next to each cache file. This contains the **URL** (p. 323) corresponding to the cache file and the expiry time, if it is available. For example `lfc://lfc1.ndgf.org/grid/atlas/test/test1 20081007151045Z`

While cache files are downloaded, they are locked by creating a lock file with the `.lock` suffix next to the cache file. Calling `Start()` (p. 163) creates this lock and `Stop()` (p. 163) releases it. All processes calling `Start()` (p. 163) must wait until they successfully obtain the lock before downloading can begin.

## 6.109.2 Constructor & Destructor Documentation

### 6.109.2.1 `Arc::FileCache::FileCache ( std::string cache_path, std::string id, uid_t job_uid, gid_t job_gid )`

Create a new `FileCache` (p. 159) instance.

#### Parameters

*cache\_path* The format is "cache\_dir[ link\_path]". path is the path to the cache directory and the optional link\_path is used to create a link in case the cache directory is visible under a different name during actual usage. When linking from the session dir this path is used instead of cache\_path.

*id* the job id. This is used to create the per-job dir which the job's cache files will be hard linked from

*job\_uid* owner of job. The per-job dir will only be readable by this user

*job\_gid* owner group of job

### 6.109.2.2 `Arc::FileCache::FileCache ( std::vector< std::string > caches, std::string id, uid_t job_uid, gid_t job_gid )`

Create a new `FileCache` (p. 159) instance with multiple cache dirs

#### Parameters

*caches* a vector of strings describing caches. The format of each string is "cache\_dir[ link\_path]".

*id* the job id. This is used to create the per-job dir which the job's cache files will be hard linked from

*job\_uid* owner of job. The per-job dir will only be readable by this user

*job\_gid* owner group of job

### 6.109.2.3 `Arc::FileCache::FileCache ( std::vector< std::string > caches, std::vector< std::string > remote_caches, std::vector< std::string > draining_caches, std::string id, uid_t job_uid, gid_t job_gid, int cache_max = 100, int cache_min = 100 )`

Create a new `FileCache` (p. 159) instance with multiple cache dirs, remote caches and draining cache directories.

#### Parameters

*caches* a vector of strings describing caches. The format of each string is "cache\_dir[ link\_path]".

*remote\_caches* Same format as caches. These are the paths to caches which are under the control of other Grid Managers and are read-only for this process.

*draining\_caches* Same format as caches. These are the paths to caches which are to be drained.

*id* the job id. This is used to create the per-job dir which the job's cache files will be hard linked from

*job\_uid* owner of job. The per-job dir will only be readable by this user

*job\_gid* owner group of job

*cache\_max* maximum used space by cache, as percentage of the file system

*cache\_min* minimum used space by cache, as percentage of the file system

#### 6.109.2.4 Arc::FileCache::FileCache ( ) [inline]

Default constructor. Invalid cache.

### 6.109.3 Member Function Documentation

#### 6.109.3.1 bool Arc::FileCache::AddDN ( std::string url, std::string DN, Time expiry\_time )

Add the given DN to the list of cached DNs with the given expiry time

##### Parameters

*url* the url corresponding to the cache file to which we want to add a cached DN

*DN* the DN of the user

*expiry\_time* the expiry time of this DN in the DN cache

#### 6.109.3.2 bool Arc::FileCache::CheckCreated ( std::string url )

Check if there is an information about creation time. Returns true if the file exists in the cache, since the creation time is the creation time of the cache file.

##### Parameters

*url* the url corresponding to the cache file for which we want to know if the creation date exists

#### 6.109.3.3 bool Arc::FileCache::CheckDN ( std::string url, std::string DN )

Check if the given DN is cached for authorisation.

##### Parameters

*url* the url corresponding to the cache file for which we want to check the cached DN

*DN* the DN of the user

#### 6.109.3.4 bool Arc::FileCache::CheckValid ( std::string url )

Check if there is an information about expiry time.

##### Parameters

*url* the url corresponding to the cache file for which we want to know if the expiration time exists

**6.109.3.5** `bool Arc::FileCache::Copy ( std::string dest_path, std::string url, bool executable = false )`

Copy the cache file corresponding to *url* to the *dest\_path*

**6.109.3.6** `std::string Arc::FileCache::File ( std::string url )`

Returns the full pathname of the file in the cache which corresponds to the given *url*.

**6.109.3.7** `Time Arc::FileCache::GetCreated ( std::string url )`

Get the creation time of a cached file. If the cache file does not exist, 0 is returned.

**Parameters**

*url* the url corresponding to the cache file for which we want to know the creation date

**6.109.3.8** `Time Arc::FileCache::GetValid ( std::string url )`

Get expiry time of a cached file. If the time is not available, a time equivalent to 0 is returned.

**Parameters**

*url* the url corresponding to the cache file for which we want to know the expiry time

**6.109.3.9** `bool Arc::FileCache::Link ( std::string link_path, std::string url )`

Create a hard-link to the per-job dir from the cache dir, and then a soft-link from here to the session directory. This is effectively 'claiming' the file for the job, so even if the original cache file is deleted, eg by some external process, the hard link still exists until it is explicitly released by calling **Release()** (p. 162).

If *cache\_link\_path* is set to "." then files will be copied directly to the session directory rather than via the hard link.

**Parameters**

*link\_path* path to the session dir for soft-link or new file

*url* url of file to link to or copy

**6.109.3.10** `Arc::FileCache::operator bool ( void ) [inline]`

Returns true if object is useable.

**6.109.3.11** `bool Arc::FileCache::operator==( const FileCache & a )`

Return true if all attributes are equal

**6.109.3.12** `bool Arc::FileCache::Release ( )`

Release claims on input files for the job specified by id. For each cache directory the per-job directory with the hard-links will be deleted.

**6.109.3.13** `bool Arc::FileCache::SetValid ( std::string url, Time val )`

Set expiry time.

**Parameters**

*url* the url corresponding to the cache file for which we want to set the expiry time  
*val* expiry time

**6.109.3.14** `bool Arc::FileCache::Start ( std::string url, bool & available, bool & is_locked, bool use_remote = true )`

Prepare cache for downloading file, and lock the cached file. On success returns true. If there is another process downloading the same url, false is returned and `is_locked` is set to true. In this case the client should wait and retry later. If the lock has expired this process will take over the lock and the method will return as if no lock was present, ie `available` and `is_locked` are false.

**Parameters**

*url* url that is being downloaded  
*available* true on exit if the file is already in cache  
*is\_locked* true on exit if the file is already locked, ie cannot be used by this process  
*remote\_caches* Same format as `caches`. These are the paths to caches which are under the control of other Grid Managers and are read-only for this process.

**6.109.3.15** `bool Arc::FileCache::Stop ( std::string url )`

This method (or `stopAndDelete`) must be called after file was downloaded or download failed, to release the lock on the cache file. `Stop()` (p. 163) does not delete the cache file. It returns false if the lock file does not exist, or another pid was found inside the lock file (this means another process took over the lock so this process must go back to `Start()` (p. 163)), or if it fails to delete the lock file.

**Parameters**

*url* the url of the file that was downloaded

**6.109.3.16** `bool Arc::FileCache::StopAndDelete ( std::string url )`

Release the cache file and delete it, because for example a failed download left an incomplete copy, or it has expired. This method also deletes the meta file which contains the url corresponding to the cache file. The logic of the return value is the same as `Stop()` (p. 163).

**Parameters**

*url* the url corresponding to the cache file that has to be released and deleted

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

- FileCache.h

## 6.110 FileCacheHash Class Reference

```
#include <FileCacheHash.h>
```

### Static Public Member Functions

- static std::string **getHash** (std::string url)
- static int **maxLength** ()

#### 6.110.1 Detailed Description

**FileCacheHash** (p. 164) provides methods to make hashes from strings. Currently the md5 hash from the openssl library is used.

#### 6.110.2 Member Function Documentation

##### 6.110.2.1 static std::string FileCacheHash::getHash ( std::string *url* ) [static]

Return a hash of the given URL, according to the current hash scheme.

##### 6.110.2.2 static int FileCacheHash::maxLength ( ) [inline, static]

Return the maximum length of a hash string.

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

- FileCacheHash.h

## 6.111 Arc::FileInfo Class Reference

**FileInfo** (p. 164) stores information about files (metadata).

```
#include <FileInfo.h>
```

### 6.111.1 Detailed Description

**FileInfo** (p. 164) stores information about files (metadata).

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

- FileInfo.h

## 6.112 Arc::FileLock Class Reference

A general file locking class.

```
#include <FileLock.h>
```

## Public Member Functions

- **FileLock** (const std::string &filename)
- **~FileLock** ()
- **operator bool** ()
- **bool operator!** ()

### 6.112.1 Detailed Description

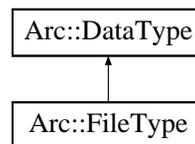
A general file locking class.

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

- FileLock.h

## 6.113 Arc::FileType Class Reference

Inheritance diagram for Arc::FileType:



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

- JobDescription.h

## 6.114 Arc::FinderLoader Class Reference

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

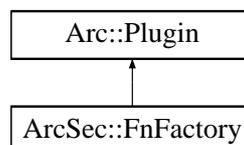
- FinderLoader.h

## 6.115 ArcSec::FnFactory Class Reference

Interface for function factory class.

```
#include <FnFactory.h>
```

Inheritance diagram for ArcSec::FnFactory:



## Public Member Functions

- virtual **Function** \* **createFn** (const std::string &type)=0

### 6.115.1 Detailed Description

Interface for function factory class. **FnFactory** (p. 165) is in charge of creating **Function** (p. 166) object according to the algorithm type given as argument of method **createFn**. This class can be inherited for implementing a factory class which can create some specific **Function** (p. 166) objects.

### 6.115.2 Member Function Documentation

#### 6.115.2.1 virtual Function\* ArcSec::FnFactory::createFn ( const std::string & type ) [pure virtual]

creat algorithm object based on the type algorithm type

#### Parameters

*type* The type of **Function** (p. 166)

#### Returns

The object of **Function** (p. 166)

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

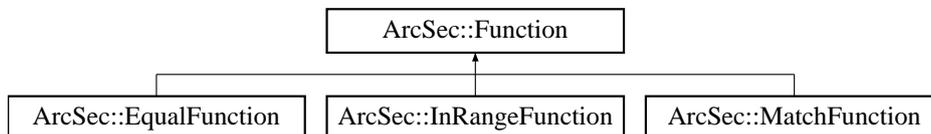
- FnFactory.h

## 6.116 ArcSec::Function Class Reference

Interface for function, which is in charge of evaluating two **AttributeValue** (p. 56).

```
#include <Function.h>
```

Inheritance diagram for ArcSec::Function:



## Public Member Functions

- virtual **AttributeValue** \* **evaluate** (**AttributeValue** \*arg0, **AttributeValue** \*arg1, bool check\_id=true)=0
- virtual std::list< **AttributeValue** \* > **evaluate** (std::list< **AttributeValue** \* > args, bool check\_id=true)=0

## 6.116.1 Detailed Description

Interface for function, which is in charge of evaluating two `AttributeValue` (p. 56).

## 6.116.2 Member Function Documentation

### 6.116.2.1 `virtual AttributeValue* ArcSec::Function::evaluate ( AttributeValue * arg0, AttributeValue * arg1, bool check_id = true ) [pure virtual]`

Evaluate two `AttributeValue` (p. 56) objects, and return one `AttributeValue` (p. 56) object

Implemented in `ArcSec::EqualFunction` (p. 148), `ArcSec::InRangeFunction` (p. 179), and `ArcSec::MatchFunction` (p. 201).

### 6.116.2.2 `virtual std::list<AttributeValue*> ArcSec::Function::evaluate ( std::list< AttributeValue * > args, bool check_id = true ) [pure virtual]`

Evaluate a list of `AttributeValue` (p. 56) objects, and return a list of Attribute objects

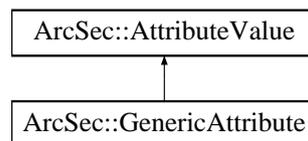
Implemented in `ArcSec::EqualFunction` (p. 148), `ArcSec::InRangeFunction` (p. 179), and `ArcSec::MatchFunction` (p. 201).

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

- Function.h

## 6.117 ArcSec::GenericAttribute Class Reference

Inheritance diagram for ArcSec::GenericAttribute:



## Public Member Functions

- virtual bool `equal (AttributeValue *other, bool check_id=true)`
- virtual std::string `encode ()`
- virtual std::string `getType ()`
- virtual std::string `getId ()`

## 6.117.1 Member Function Documentation

### 6.117.1.1 `virtual std::string ArcSec::GenericAttribute::encode ( ) [inline, virtual]`

encode the value in a string format

Implements `ArcSec::AttributeValue` (p. 57).

**6.117.1.2** `virtual bool ArcSec::GenericAttribute::equal ( AttributeValue * value, bool check_id = true ) [virtual]`

Evaluate whether "this" equals to the parameter value

Implements `ArcSec::AttributeValue` (p. 58).

**6.117.1.3** `virtual std::string ArcSec::GenericAttribute::getId ( ) [inline, virtual]`

Get the AttributeId of the <Attribute>

Implements `ArcSec::AttributeValue` (p. 58).

**6.117.1.4** `virtual std::string ArcSec::GenericAttribute::getType ( ) [inline, virtual]`

Get the DataType of the <Attribute>

Implements `ArcSec::AttributeValue` (p. 58).

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

- GenericAttribute.h

## 6.118 Arc::GlobusResult Class Reference

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

- GlobusErrorUtils.h

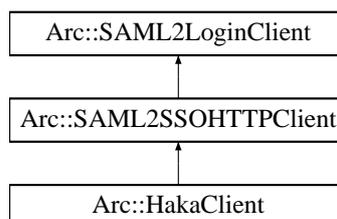
## 6.119 Arc::GSSCredential Class Reference

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

- GSSCredential.h

## 6.120 Arc::HakaClient Class Reference

Inheritance diagram for `Arc::HakaClient`:



## Protected Member Functions

- **MCC\_Status processIdPLogin** (const std::string username, const std::string password)
- **MCC\_Status processConsent** ()
- **MCC\_Status processIdP2Confusa** ()

### 6.120.1 Member Function Documentation

#### 6.120.1.1 MCC\_Status Arc::HakaClient::processConsent ( ) [protected, virtual]

If the IdP has a consent module and the user has not saved her consent, this method will ask the user for consent to transmission of her data to Confusa

Implements **Arc::SAML2SSOHTTPClient** (p. 276).

#### 6.120.1.2 MCC\_Status Arc::HakaClient::processIdP2Confusa ( ) [protected, virtual]

Redirects the user back from identity provider to the Confusa SP

Implements **Arc::SAML2SSOHTTPClient** (p. 276).

#### 6.120.1.3 MCC\_Status Arc::HakaClient::processIdPLogin ( const std::string *username*, const std::string *password* ) [protected, virtual]

Actual identity provider parsers for next three methods implemented in subdirectory idp/

Parse identity provider login page and submit username and password in the provisioned way

Implements **Arc::SAML2SSOHTTPClient** (p. 276).

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

- HakaClient.h

## 6.121 Arc::HTTPClientInfo Struct Reference

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

- ClientInterface.h

## 6.122 Arc::InfoCache Class Reference

Stores XML document in filesystem split into parts.

```
#include <InfoCache.h>
```

### Public Member Functions

- **InfoCache** (const **Config** &cfg, const std::string &service\_id)

### 6.122.1 Detailed Description

Stores XML document in filesystem split into parts.

### 6.122.2 Constructor & Destructor Documentation

#### 6.122.2.1 `Arc::InfoCache::InfoCache ( const Config & cfg, const std::string & service_id )`

Creates object according to configuration (see InfoCacheConfig.xsd).

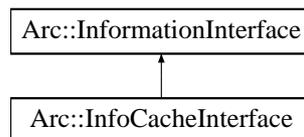
XML configuration is passed in `cfg`. Argument `service_id` is used to distinguish between various documents stored under same path - corresponding files will be stored in subdirectory with `service_id` name.

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

- InfoCache.h

## 6.123 `Arc::InfoCacheInterface` Class Reference

Inheritance diagram for `Arc::InfoCacheInterface`:



### Protected Member Functions

- virtual void **Get** (const std::list< std::string > &path, **XMLNodeContainer** &result)

### 6.123.1 Member Function Documentation

#### 6.123.1.1 `virtual void Arc::InfoCacheInterface::Get ( const std::list< std::string > & path, XMLNodeContainer & result ) [protected, virtual]`

This method is called by this object's `Process` method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single `Process` call. Here is a set on XML element names specifying how to reach requested node(s).

Reimplemented from `Arc::InformationInterface` (p. 176).

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

- InfoCache.h

## 6.124 `Arc::InfoFilter` Class Reference

Filters information document according to identity of requestor.

```
#include <InfoFilter.h>
```

## Public Member Functions

- **InfoFilter** (**MessageAuth** &id)
- **bool Filter** (**XMLNode** doc) const
- **bool Filter** (**XMLNode** doc, const **InfoFilterPolicies** &policies, const **NS** &ns) const

### 6.124.1 Detailed Description

Filters information document according to identity of requestor. Identity is compared to policies stored inside information document and external ones. Parts of document which do not pass policy evaluation are removed.

### 6.124.2 Constructor & Destructor Documentation

#### 6.124.2.1 Arc::InfoFilter::InfoFilter ( MessageAuth & id )

Creates object and associates identity.

Associated identity is not copied, hence passed argument must not be destroyed while this method is used.

### 6.124.3 Member Function Documentation

#### 6.124.3.1 bool Arc::InfoFilter::Filter ( XMLNode doc ) const

Filter information document according to internal policies.

In provided document all policies and nodes which have their policies evaluated to negative result are removed.

#### 6.124.3.2 bool Arc::InfoFilter::Filter ( XMLNode doc, const InfoFilterPolicies & policies, const NS & ns ) const

Filter information document according to internal and external policies.

In provided document all policies and nodes which have their policies evaluated to negative result are removed. External policies are provided in policies argument. First element of every pair is XPath defining to which XML node policy must be applied. Second element is policy itself. Argument ns defines XML namespaces for XPath evaluation.

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

- InfoFilter.h

## 6.125 Arc::InfoRegister Class Reference

Registration to ISIS interface.

```
#include <InfoRegister.h>
```

### 6.125.1 Detailed Description

Registration to ISIS interface. This class represents service registering to Information Indexing **Service** (p. 285). It does not perform registration itself. It only collects configuration information. Configuration is as described in InfoRegisterConfig.xsd for element InfoRegistration.

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

- InfoRegister.h

## 6.126 Arc::InfoRegisterContainer Class Reference

```
#include <InfoRegister.h>
```

### Public Member Functions

- **InfoRegistrar** \* **addRegistrar** (**XMLNode** doc)
- void **addService** (**InfoRegister** \*reg, const std::list< std::string > &ids, **XMLNode** cfg=XMLNode())
- void **removeService** (**InfoRegister** \*reg)

### 6.126.1 Detailed Description

Singleton class for scanning configuration and storing references to registration elements.

### 6.126.2 Member Function Documentation

#### 6.126.2.1 InfoRegistrar\* Arc::InfoRegisterContainer::addRegistrar ( XMLNode doc )

Adds ISISes to list of handled services.

Supplied configuration document is scanned for **InfoRegistrar** (p. 173) elements and those are turned into **InfoRegistrar** (p. 173) classes for handling connection to ISIS service each.

#### 6.126.2.2 void Arc::InfoRegisterContainer::addService ( InfoRegister \* reg, const std::list< std::string > & ids, XMLNode cfg = XMLNode () )

Adds service to list of handled.

This method must be called first time after last addRegistrar was called - services will be only associated with ISISes which are already added. Argument ids contains list of ISIS identifiers to which service is associated. If ids is empty then service is associated to all ISISes currently added. If argument cfg is available and no ISISes are configured then addRegistrars is called with cfg used as configuration document.

#### 6.126.2.3 void Arc::InfoRegisterContainer::removeService ( InfoRegister \* reg )

This method must be called if service being destroyed.

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

- InfoRegister.h

## 6.127 Arc::InfoRegisters Class Reference

Handling multiple registrations to ISISes.

```
#include <InfoRegister.h>
```

### Public Member Functions

- **InfoRegisters** (XMLNode &cfg, Service \*service\_)

#### 6.127.1 Detailed Description

Handling multiple registrations to ISISes.

#### 6.127.2 Constructor & Destructor Documentation

##### 6.127.2.1 Arc::InfoRegisters::InfoRegisters ( XMLNode & cfg, Service \* service\_ )

Constructor creates **InfoRegister** (p. 171) objects according to configuration.

Inside cfg elements InfoRegistration are found and for each corresponding **InfoRegister** (p. 171) object is created. Those objects are destroyed in destructor of this class.

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

- InfoRegister.h

## 6.128 Arc::InfoRegistrar Class Reference

Registration process associated with particular ISIS.

```
#include <InfoRegister.h>
```

### Public Member Functions

- void **registration** (void)
- bool **addService** (InfoRegister \*, XMLNode &)
- bool **removeService** (InfoRegister \*)

#### 6.128.1 Detailed Description

Registration process associated with particular ISIS. Instance of this class starts thread which takes care passing information about associated services to ISIS service defined in configuration. Configuration is as described in InfoRegister.xsd for element **InfoRegistrar** (p. 173).

## 6.128.2 Member Function Documentation

### 6.128.2.1 `bool Arc::InfoRegistrar::addService ( InfoRegister *, XMLNode & )`

Adds new service to list of handled services.

**Service** (p. 285) is described by it's **InfoRegister** (p. 171) object which must be valid as long as this object is functional.

### 6.128.2.2 `void Arc::InfoRegistrar::registration ( void )`

Performs registration in a loop.

Never exits unless there is a critical error or requested by destructor.

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

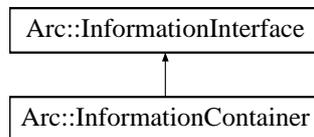
- InfoRegister.h

## 6.129 Arc::InformationContainer Class Reference

Information System document container and processor.

```
#include <InformationInterface.h>
```

Inheritance diagram for Arc::InformationContainer:



### Public Member Functions

- **InformationContainer** (**XMLNode** doc, bool copy=false)
- **XMLNode Acquire** (void)
- void **Assign** (**XMLNode** doc, bool copy=false)

### Protected Member Functions

- virtual void **Get** (const std::list< std::string > &path, **XMLNodeContainer** &result)

### Protected Attributes

- **XMLNode** doc\_

### 6.129.1 Detailed Description

Information System document container and processor. This class inherits from **InformationInterface** (p. 175) and offers container for storing informational XML document.

## 6.129.2 Constructor & Destructor Documentation

### 6.129.2.1 Arc::InformationContainer::InformationContainer ( XMLNode *doc*, bool *copy* = *false* )

Creates an instance with XML document . If is true this method makes a copy of for internal use.

## 6.129.3 Member Function Documentation

### 6.129.3.1 XMLNode Arc::InformationContainer::Acquire ( void )

Get a lock on contained XML document. To be used in multi-threaded environment. Do not forget to release it with Release()

### 6.129.3.2 void Arc::InformationContainer::Assign ( XMLNode *doc*, bool *copy* = *false* )

Replaces internal XML document with . If is true this method makes a copy of for internal use.

### 6.129.3.3 virtual void Arc::InformationContainer::Get ( const std::list< std::string > & *path*, XMLNodeContainer & *result* ) [protected, virtual]

This method is called by this object's Process method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single Process call. Here is a set on XML element names specifying how to reach requested node(s).

Reimplemented from Arc::InformationInterface (p. 176).

## 6.129.4 Field Documentation

### 6.129.4.1 XMLNode Arc::InformationContainer::doc\_ [protected]

Either link or container of XML document

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

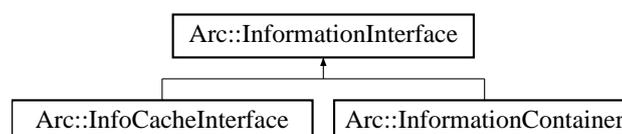
- InformationInterface.h

## 6.130 Arc::InformationInterface Class Reference

Information System message processor.

```
#include <InformationInterface.h>
```

Inheritance diagram for Arc::InformationInterface:



## Public Member Functions

- **InformationInterface** (bool safe=true)

## Protected Member Functions

- virtual void **Get** (const std::list< std::string > &path, **XMLNodeContainer** &result)

## Protected Attributes

- Glib::Mutex **lock\_**

### 6.130.1 Detailed Description

Information System message processor. This class provides callback for 2 operations of WS-ResourceProperties and convenient parsing/generation of corresponding SOAP messages. In a future it may extend range of supported specifications.

### 6.130.2 Constructor & Destructor Documentation

#### 6.130.2.1 Arc::InformationInterface::InformationInterface ( bool safe = true )

Constructor. If 'safe' is true all calls to Get will be locked.

### 6.130.3 Member Function Documentation

#### 6.130.3.1 virtual void Arc::InformationInterface::Get ( const std::list< std::string > & path, XMLNodeContainer & result ) [protected, virtual]

This method is called by this object's Process method. Real implementation of this class should return (sub)tree of XML document. This method may be called multiple times per single Process call. Here is a set on XML element names specifying how to reach requested node(s).

Reimplemented in **Arc::InfoCacheInterface** (p. 170), and **Arc::InformationContainer** (p. 175).

### 6.130.4 Field Documentation

#### 6.130.4.1 Glib::Mutex Arc::InformationInterface::lock\_ [protected]

Mutex used to protect access to Get methods in multi-threaded env.

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

- InformationInterface.h

## 6.131 Arc::InformationRequest Class Reference

Request for information in InfoSystem.

```
#include <InformationInterface.h>
```

## Public Member Functions

- **InformationRequest** (void)
- **InformationRequest** (const std::list< std::string > &path)
- **InformationRequest** (const std::list< std::list< std::string > > &paths)
- **InformationRequest** (XMLNode query)
- SOAPEnvelope \* **SOAP** (void)

### 6.131.1 Detailed Description

Request for information in InfoSystem. This is a convenience wrapper creating proper WS-ResourceProperties request targeted InfoSystem interface of service.

### 6.131.2 Constructor & Destructor Documentation

#### 6.131.2.1 Arc::InformationRequest::InformationRequest ( void )

Dummy constructor

#### 6.131.2.2 Arc::InformationRequest::InformationRequest ( const std::list< std::string > & path )

Request for attribute specified by elements of path. Currently only first element is used.

#### 6.131.2.3 Arc::InformationRequest::InformationRequest ( const std::list< std::list< std::string > > & paths )

Request for attribute specified by elements of paths. Currently only first element of every path is used.

#### 6.131.2.4 Arc::InformationRequest::InformationRequest ( XMLNode query )

Request for attributes specified by XPath query.

### 6.131.3 Member Function Documentation

#### 6.131.3.1 SOAPEnvelope\* Arc::InformationRequest::SOAP ( void )

Returns generated SOAP message

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

- InformationInterface.h

## 6.132 Arc::InformationResponse Class Reference

Informational response from InfoSystem.

```
#include <InformationInterface.h>
```

### Public Member Functions

- **InformationResponse** (SOAPEnvelope &soap)
- `std::list< XMLNode > Result` (void)

### 6.132.1 Detailed Description

Informational response from InfoSystem. This is a convenience wrapper analyzing WS-ResourceProperties response from InfoSystem interface of service.

### 6.132.2 Constructor & Destructor Documentation

#### 6.132.2.1 Arc::InformationResponse::InformationResponse ( SOAPEnvelope & soap )

Constructor parses WS-ResourceProperties response. Provided SOAPEnvelope object must be valid as long as this object is in use.

### 6.132.3 Member Function Documentation

#### 6.132.3.1 `std::list<XMLNode> Arc::InformationResponse::Result ( void )`

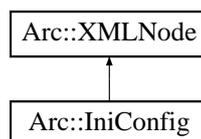
Returns set of attributes which were in SOAP message passed to constructor.

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

- InformationInterface.h

## 6.133 Arc::IniConfig Class Reference

Inheritance diagram for Arc::IniConfig:



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

- IniConfig.h

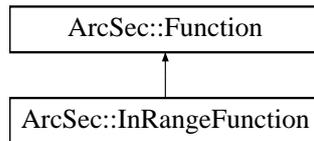
## 6.134 Arc::initializeCredentialsType Class Reference

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

- UserConfig.h

## 6.135 ArcSec::InRangeFunction Class Reference

Inheritance diagram for ArcSec::InRangeFunction:



### Public Member Functions

- virtual `AttributeValue * evaluate (AttributeValue *arg0, AttributeValue *arg1, bool check_id=true)`
- virtual `std::list< AttributeValue * > evaluate (std::list< AttributeValue * > args, bool check_id=true)`

### 6.135.1 Member Function Documentation

#### 6.135.1.1 virtual `AttributeValue* ArcSec::InRangeFunction::evaluate ( AttributeValue * arg0, AttributeValue * arg1, bool check_id = true ) [virtual]`

Evaluate two `AttributeValue` (p. 56) objects, and return one `AttributeValue` (p. 56) object

Implements `ArcSec::Function` (p. 167).

#### 6.135.1.2 virtual `std::list<AttributeValue*> ArcSec::InRangeFunction::evaluate ( std::list< AttributeValue * > args, bool check_id = true ) [virtual]`

Evaluate a list of `AttributeValue` (p. 56) objects, and return a list of Attribute objects

Implements `ArcSec::Function` (p. 167).

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

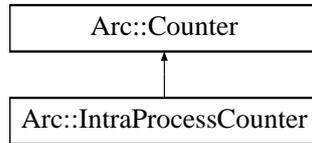
- InRangeFunction.h

## 6.136 Arc::IntraProcessCounter Class Reference

A class for counters used by threads within a single process.

```
#include <IntraProcessCounter.h>
```

Inheritance diagram for Arc::IntraProcessCounter:



## Public Member Functions

- **IntraProcessCounter** (int limit, int excess)
- virtual  $\sim$ **IntraProcessCounter** ()
- virtual int **getLimit** ()
- virtual int **setLimit** (int newLimit)
- virtual int **changeLimit** (int amount)
- virtual int **getExcess** ()
- virtual int **setExcess** (int newExcess)
- virtual int **changeExcess** (int amount)
- virtual int **getValue** ()
- virtual **CounterTicket reserve** (int amount=1, Glib::TimeVal duration=**ETERNAL**, bool prioritized=false, Glib::TimeVal timeOut=**ETERNAL**)

## Protected Member Functions

- virtual void **cancel** (IDType reservationID)
- virtual void **extend** (IDType &reservationID, Glib::TimeVal &expiryTime, Glib::TimeVal duration=**ETERNAL**)

### 6.136.1 Detailed Description

A class for counters used by threads within a single process. This is a class for shared among different threads within a single process. See the **Counter** (p. 81) class for further information about counters and examples of usage.

### 6.136.2 Constructor & Destructor Documentation

#### 6.136.2.1 Arc::IntraProcessCounter::IntraProcessCounter ( int *limit*, int *excess* )

Creates an **IntraProcessCounter** (p. 179) with specified limit and excess.

This constructor creates a counter with the specified limit (amount of resources available for reservation) and excess limit (an extra amount of resources that may be used for prioritized reservations).

#### Parameters

*limit* The limit of the counter.

*excess* The excess limit of the counter.

### 6.136.2.2 virtual Arc::IntraProcessCounter::~~IntraProcessCounter ( ) [virtual]

Destructor.

This is the destructor of the **IntraProcessCounter** (p. 179) class. Does not need to do anything.

## 6.136.3 Member Function Documentation

### 6.136.3.1 virtual void Arc::IntraProcessCounter::cancel ( IDType *reservationID* ) [protected, virtual]

Cancellation of a reservation.

This method cancels a reservation. It is called by the **CounterTicket** (p. 88) that corresponds to the reservation.

#### Parameters

*reservationID* The identity number (key) of the reservation to cancel.

### 6.136.3.2 virtual int Arc::IntraProcessCounter::changeExcess ( int *amount* ) [virtual]

Changes the excess limit of the counter.

Changes the excess limit of the counter by adding a certain amount to the current excess limit.

#### Parameters

*amount* The amount by which to change the excess limit.

#### Returns

The new excess limit.

Implements **Arc::Counter** (p. 84).

### 6.136.3.3 virtual int Arc::IntraProcessCounter::changeLimit ( int *amount* ) [virtual]

Changes the limit of the counter.

Changes the limit of the counter by adding a certain amount to the current limit.

#### Parameters

*amount* The amount by which to change the limit.

#### Returns

The new limit.

Implements **Arc::Counter** (p. 84).

**6.136.3.4** `virtual void Arc::IntraProcessCounter::extend ( IDType & reservationID, Glib::TimeVal & expiryTime, Glib::TimeVal duration = ETERNAL ) [protected, virtual]`

Extension of a reservation.

This method extends a reservation. It is called by the **CounterTicket** (p. 88) that corresponds to the reservation.

**Parameters**

*reservationID* Used for input as well as output. Contains the identification number of the original reservation on entry and the new identification number of the extended reservation on exit.

*expiryTime* Used for input as well as output. Contains the expiry time of the original reservation on entry and the new expiry time of the extended reservation on exit.

*duration* The time by which to extend the reservation. The new expiration time is computed based on the current time, NOT the previous expiration time.

**6.136.3.5** `virtual int Arc::IntraProcessCounter::getExcess ( ) [virtual]`

Returns the excess limit of the counter.

Returns the excess limit of the counter, i.e. by how much the usual limit may be exceeded by prioritized reservations.

**Returns**

The excess limit.

Implements **Arc::Counter** (p. 85).

**6.136.3.6** `virtual int Arc::IntraProcessCounter::getLimit ( ) [virtual]`

Returns the current limit of the counter.

This method returns the current limit of the counter, i.e. how many units can be reserved simultaneously by different threads without claiming high priority.

**Returns**

The current limit of the counter.

Implements **Arc::Counter** (p. 86).

**6.136.3.7** `virtual int Arc::IntraProcessCounter::getValue ( ) [virtual]`

Returns the current value of the counter.

Returns the current value of the counter, i.e. the number of unreserved units. Initially, the value is equal to the limit of the counter. When a reservation is made, the value is decreased. Normally, the value should never be negative, but this may happen if there are prioritized reservations. It can also happen if the limit is decreased after some reservations have been made, since reservations are never revoked.

**Returns**

The current value of the counter.

Implements **Arc::Counter** (p. 86).

**6.136.3.8** `virtual CounterTicket Arc::IntraProcessCounter::reserve ( int amount = 1, Glib::TimeVal duration = ETERNAL, bool prioritized = false, Glib::TimeVal timeOut = ETERNAL ) [virtual]`

Makes a reservation from the counter.

This method makes a reservation from the counter. If the current value of the counter is too low to allow for the reservation, the method blocks until the reservation is possible or times out.

#### Parameters

*amount* The amount to reserve, default value is 1.

*duration* The duration of a self expiring reservation, default is that it lasts forever.

*prioritized* Whether this reservation is prioritized and thus allowed to use the excess limit.

*timeOut* The maximum time to block if the value of the counter is too low, default is to allow "eternal" blocking.

#### Returns

A **CounterTicket** (p. 88) that can be queried about the status of the reservation as well as for cancellations and extensions.

Implements **Arc::Counter** (p. 87).

**6.136.3.9** `virtual int Arc::IntraProcessCounter::setExcess ( int newExcess ) [virtual]`

Sets the excess limit of the counter.

This method sets a new excess limit for the counter.

#### Parameters

*newExcess* The new excess limit, an absolute number.

#### Returns

The new excess limit.

Implements **Arc::Counter** (p. 87).

**6.136.3.10** `virtual int Arc::IntraProcessCounter::setLimit ( int newLimit ) [virtual]`

Sets the limit of the counter.

This method sets a new limit for the counter.

#### Parameters

*newLimit* The new limit, an absolute number.

#### Returns

The new limit.

Implements **Arc::Counter** (p. 87).

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

- IntraProcessCounter.h

## 6.137 Arc::ISIS\_description Struct Reference

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

- InfoRegister.h

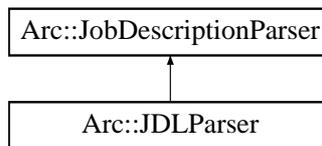
## 6.138 Arc::IString Class Reference

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

- IString.h

## 6.139 Arc::JDLParser Class Reference

Inheritance diagram for Arc::JDLParser:



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

- JDLParser.h

## 6.140 Arc::Job Class Reference

**Job** (p. 184).

```
#include <Job.h>
```

### Public Member Functions

- **Job** ()
- void **Print** (bool longlist) const

### 6.140.1 Detailed Description

**Job** (p. 184). This class describe a Grid job. Most of the members contained in this class are directly linked to the ComputingActivity defined in the GLUE Specification v. 2.0 (GFD-R-P.147).

## 6.140.2 Constructor & Destructor Documentation

### 6.140.2.1 Arc::Job::Job ( )

Create a **Job** (p. 184) object.

Default constructor. Takes no arguments.

## 6.140.3 Member Function Documentation

### 6.140.3.1 void Arc::Job::Print ( bool *longlist* ) const

Print the **Job** (p. 184) information to std::cout.

Method to print the **Job** (p. 184) attributes to std::cout

#### Parameters

*longlist* is boolean for long listing (more details).

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

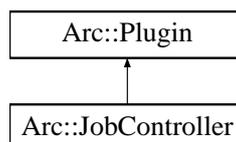
- Job.h

## 6.141 Arc::JobController Class Reference

Base class for the JobControllers.

```
#include <JobController.h>
```

Inheritance diagram for Arc::JobController:



### Public Member Functions

- void **FillJobStore** (const std::list< **URL** > &jobids)
- bool **PrintJobStatus** (const std::list< std::string > &status, const bool longlist)
- bool **Migrate** (**TargetGenerator** &targetGen, **Broker** \*broker, const **UserConfig** &usercfg, const bool forcemigration, std::list< **URL** > &migratedJobIDs)

### 6.141.1 Detailed Description

Base class for the JobControllers. The **JobController** (p. 185) is the base class for middleware specialized derived classes. The **JobController** (p. 185) base class is also the implementer of all public functionality that should be offered by the middleware specific specializations. In other words all virtual functions of the **JobController** (p. 185) are private. The initialization of a (specialized) **JobController** (p. 185) object takes

two steps. First the **JobController** (p. 185) specialization for the required grid flavour must be loaded by the **JobControllerLoader** (p. 187), which sees to that the **JobController** (p. 185) receives information about its Grid flavour and the local joblist file containing information about all active jobs (flavour independent). The next step is the filling of the **JobController** (p. 185) job pool (JobStore) which is the pool of jobs that the **JobController** (p. 185) can manage. Must be specialiced for each supported middleware flavour.

## 6.141.2 Member Function Documentation

### 6.141.2.1 void Arc::JobController::FillJobStore ( const std::list< URL > & *jobids* )

Fill jobstore.

Method to fill the jobstore with jobs that should be managed.

#### Parameters

*jobids* List of jobids to be loaded to the jobstore. If empty all jobs of the specialized grid flavour present in the joblist file (given through the *usercfg* to the constructor) will be loaded to the jobstore.

### 6.141.2.2 bool Arc::JobController::Migrate ( TargetGenerator & *targetGen*, Broker \* *broker*, const UserConfig & *usercfg*, const bool *forcemigration*, std::list< URL > & *migratedJobIDs* )

Migrate job from cluster A to Cluster B.

Method to migrate the jobs contained in the jobstore.

#### Parameters

*targetGen* **TargetGenerator** (p. 310) with targets to migrate the job to.

*broker* **Broker** (p. 62) to be used when selecting target.

*forcemigration* boolean which specifies whether a migrated job should persist if the new cluster does not succeed sending a kill/terminate request for the job.

### 6.141.2.3 bool Arc::JobController::PrintJobStatus ( const std::list< std::string > & *status*, const bool *longlist* )

Print job status to stdout.

The job status is printed to stdout when calling this method. More specifically the **Job::Print** (p. 185) method is called on each of the **Job** (p. 184) objects stored in this object, and the boolean argument *longlist* is passed directly to the method indicating whether verbose job status should be printed. The *status* argument is a list of strings each representing a job state (**JobState** (p. 189)) which is used to indicate that only jobs with a job state in the list should be considered. If the list *status* is empty all jobs will be considered.

This method is not supposed to be overloaded by extending classes.

#### Parameters

*status* a list of strings representing states to be considered.

*longlist* a boolean indicating whether verbose job information should be printed.

**Returns**

This method always returns true.

**See also**

GetJobInformation  
**Job::Print** (p. 185)  
**JobState** (p. 189)

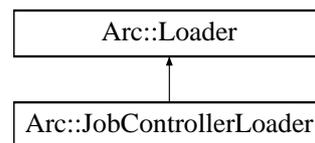
The documentation for this class was generated from the following file:

- JobController.h

## 6.142 Arc::JobControllerLoader Class Reference

```
#include <JobController.h>
```

Inheritance diagram for Arc::JobControllerLoader:

**Public Member Functions**

- **JobControllerLoader** ()
- **~JobControllerLoader** ()
- **JobController \* load** (const std::string &name, const **UserConfig** &usercfg)
- const std::list< **JobController** \* > & **GetJobControllers** () const

### 6.142.1 Detailed Description

Class responsible for loading **JobController** (p. 185) plugins The **JobController** (p. 185) objects returned by a **JobControllerLoader** (p. 187) must not be used after the **JobControllerLoader** (p. 187) goes out of scope.

### 6.142.2 Constructor & Destructor Documentation

#### 6.142.2.1 Arc::JobControllerLoader::JobControllerLoader ( )

Constructor Creates a new **JobControllerLoader** (p. 187).

#### 6.142.2.2 Arc::JobControllerLoader::~~JobControllerLoader ( )

Destructor Calling the destructor destroys all JobControllers loaded by the **JobControllerLoader** (p. 187) instance.

### 6.142.3 Member Function Documentation

#### 6.142.3.1 `const std::list<JobController*> & Arc::JobControllerLoader::GetJobControllers ( )` `const [inline]`

Retrieve the list of loaded JobControllers.

#### Returns

A reference to the list of JobControllers.

Referenced by `Arc::JobSupervisor::GetJobControllers()`.

#### 6.142.3.2 `JobController* Arc::JobControllerLoader::load ( const std::string & name, const UserConfig & usercfg )`

Load a new **JobController** (p. 185)

#### Parameters

*name* The name of the **JobController** (p. 185) to load.

*usercfg* The **UserConfig** (p. 333) object for the new **JobController** (p. 185).

#### Returns

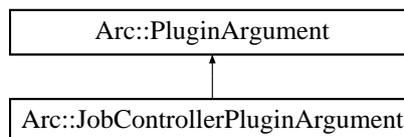
A pointer to the new **JobController** (p. 185) (NULL on error).

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

- JobController.h

## 6.143 Arc::JobControllerPluginArgument Class Reference

Inheritance diagram for `Arc::JobControllerPluginArgument`:



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

- JobController.h

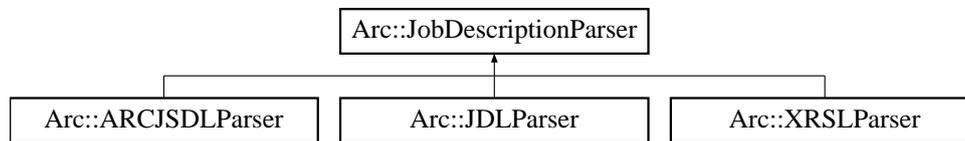
## 6.144 Arc::JobDescription Class Reference

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

- JobDescription.h

## 6.145 Arc::JobDescriptionParser Class Reference

Inheritance diagram for Arc::JobDescriptionParser:



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

- JobDescriptionParser.h

## 6.146 Arc::JobIdentificationType Class Reference

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

- JobDescription.h

## 6.147 Arc::JobMetaType Class Reference

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

- JobDescription.h

## 6.148 Arc::JobState Class Reference

```
#include <JobState.h>
```

### 6.148.1 Detailed Description

ARC general state model. The class comprise the general state model of the ARC-lib, and are herein used to compare job states from the different middlewares supported by the plugin structure of the ARC-lib. Which is why every ACC plugin should contain a class derived from this class. The derived class should consist of a constructor and a mapping function (a JobStateMap) which maps a std::string to a **JobState** (p.189):StateType. An example of a constructor in a plugin could be: JobStatePlugin::JobStatePlugging(const std::string& state) : JobState(state, &pluginStateMap) {} where &pluginStateMap is a reference to the JobStateMap defined by the derived class.

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

- JobState.h

## 6.149 Arc::JobSupervisor Class Reference

% **JobSupervisor** (p. 190) class

```
#include <JobSupervisor.h>
```

### Public Member Functions

- **JobSupervisor** (const **UserConfig** &usercfg, const std::list< std::string > &jobs)
- const std::list< **JobController** \* > & **GetJobControllers** ()

### 6.149.1 Detailed Description

% **JobSupervisor** (p. 190) class The **JobSupervisor** (p. 190) class is tool for loading **JobController** (p. 185) plugins for managing Grid jobs.

### 6.149.2 Constructor & Destructor Documentation

#### 6.149.2.1 Arc::JobSupervisor::JobSupervisor ( const UserConfig & usercfg, const std::list< std::string > & jobs )

Create a **JobSupervisor** (p. 190) object.

Default constructor to create a **JobSupervisor** (p. 190). Automatically loads **JobController** (p. 185) plugins based upon the input jobids.

#### Parameters

- usercfg* Reference to **UserConfig** (p. 333) object with information about user credentials and joblist-file.
- jobs* List of jobs(jobid or job name) to be managed.

### 6.149.3 Member Function Documentation

#### 6.149.3.1 const std::list<JobController\*>& Arc::JobSupervisor::GetJobControllers ( ) [inline]

Get list of JobControllers.

Method to get the list of JobControllers loaded by constructor.

References Arc::JobControllerLoader::GetJobControllers().

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

- JobSupervisor.h

## 6.150 Arc::LoadableModuleDescription Class Reference

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

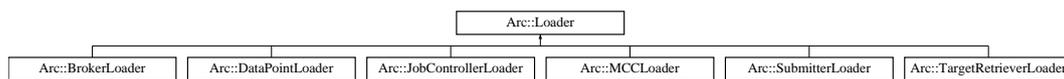
- ModuleManager.h

## 6.151 Arc::Loader Class Reference

Plugins loader.

```
#include <Loader.h>
```

Inheritance diagram for Arc::Loader:



### Public Member Functions

- **Loader** (XMLNode cfg)
- **~Loader** ()

### Protected Attributes

- **PluginsFactory** \* factory\_

#### 6.151.1 Detailed Description

Plugins loader. This class processes XML configuration and loads specified plugins. Accepted configuration is defined by XML schema mcc.xsd. "Plugins" elements are parsed by this class and corresponding libraries are loaded.

#### 6.151.2 Constructor & Destructor Documentation

##### 6.151.2.1 Arc::Loader::Loader ( XMLNode cfg )

Constructor that takes whole XML configuration and performs common configuration part

##### 6.151.2.2 Arc::Loader::~~Loader ( )

Destructor destroys all components created by constructor

#### 6.151.3 Field Documentation

##### 6.151.3.1 PluginsFactory\* Arc::Loader::factory\_ [protected]

Link to Factory responsible for loading and creation of **Plugin** (p. 249) and derived objects

Referenced by Arc::ChainContext::operator PluginsFactory \*().

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

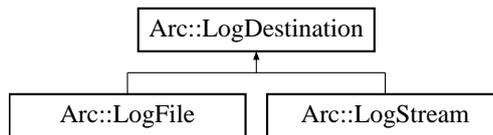
- Loader.h

## 6.152 Arc::LogDestination Class Reference

A base class for log destinations.

```
#include <Logger.h>
```

Inheritance diagram for Arc::LogDestination:



### Public Member Functions

- virtual void **log** (const **LogMessage** &message)=0

### Protected Member Functions

- **LogDestination** ()
- **LogDestination** (const std::string &locale)

#### 6.152.1 Detailed Description

A base class for log destinations. This class defines an interface for LogDestinations. **LogDestination** (p. 192) objects will typically contain synchronization mechanisms and should therefore never be copied.

#### 6.152.2 Constructor & Destructor Documentation

##### 6.152.2.1 Arc::LogDestination::LogDestination ( ) [protected]

Default constructor.

This destination will use the default locale.

##### 6.152.2.2 Arc::LogDestination::LogDestination ( const std::string & locale ) [protected]

Constructor with specific locale.

This destination will use the specified locale.

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

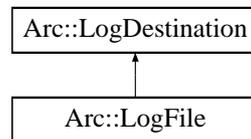
- Logger.h

## 6.153 Arc::LogFile Class Reference

A class for logging to files.

```
#include <Logger.h>
```

Inheritance diagram for Arc::LogFile:



## Public Member Functions

- **LogFile** (const std::string &path)
- **LogFile** (const std::string &path, const std::string &locale)
- void **setMaxSize** (int newsize)
- void **setBackups** (int newbackup)
- void **setReopen** (bool newreopen)
- **operator bool** (void)
- **bool operator!** (void)
- virtual void **log** (const **LogMessage** &message)

### 6.153.1 Detailed Description

A class for logging to files. This class is used for logging to files. It provides synchronization in order to prevent different LogMessages to appear mixed with each other in the stream. It is possible to limit size of created file. Whenever specified size is exceeded file is deleted and new one is created. Old files may be moved into backup files instead of being deleted. Those files have names same as initial file with additional number suffix - similar to those found in /var/log of many Unix-like systems.

### 6.153.2 Constructor & Destructor Documentation

#### 6.153.2.1 Arc::LogFile::LogFile ( const std::string & *path* )

Creates a **LogFile** (p. 192) connected to a file.

Creates a **LogFile** (p. 192) connected to the file located at specified path. In order not to break synchronization, it is important not to connect more than one **LogFile** (p. 192) object to a certain file. If file does not exist it will be created.

#### Parameters

*path* The path to file to which to write LogMessages.

#### 6.153.2.2 Arc::LogFile::LogFile ( const std::string & *path*, const std::string & *locale* )

Creates a **LogFile** (p. 192) connected to a file.

Creates a **LogFile** (p. 192) connected to the file located at specified path. The output will be localised to the specified locale.

### 6.153.3 Member Function Documentation

#### 6.153.3.1 virtual void Arc::LogFile::log ( const LogMessage & *message* ) [virtual]

Writes a **LogMessage** (p. 197) to the file.

This method writes a **LogMessage** (p. 197) to the file that is connected to this **LogFile** (p. 192) object. If after writitng size of file exceeds one set by **setMaxSize()** (p. 194) file is moved to backup and new one is created.

##### Parameters

*message* The **LogMessage** (p. 197) to write.

Implements **Arc::LogDestination** (p. 192).

#### 6.153.3.2 void Arc::LogFile::setBackups ( int *newbackup* )

Set number of backups to store.

Set number of backups to store. When file size exceeds one specified with **setMaxSize()** (p. 194) file is closed and moved to one named path.1. If path.1 exists it is moved to path.2 and so on. Number of path.# files is one set in newbackup.

##### Parameters

*newbackup* Number of backup files.

#### 6.153.3.3 void Arc::LogFile::setMaxSize ( int *newsiz*e )

Set maximal allowed size of file.

Set maximal allowed size of file. This value is not obeyed exactly. Spesified size may be exceeded by amount of one **LogMessage** (p. 197). To disable limit specify -1.

##### Parameters

*newsiz*e Max size of log file.

#### 6.153.3.4 void Arc::LogFile::setReopen ( bool *newreopen* )

Set file reopen on every write.

Set file reopen on every write. If set to true file is opened before writing every log record and closed afterward.

##### Parameters

*newreopen* If file to be reopened for every log record.

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

- Logger.h

## 6.154 Arc::Logger Class Reference

A logger class.

```
#include <Logger.h>
```

### Public Member Functions

- **Logger** (**Logger** &parent, const std::string &subdomain)
- **Logger** (**Logger** &parent, const std::string &subdomain, **LogLevel** threshold)
- **~Logger** ()
- void **addDestination** (**LogDestination** &destination)
- void **removeDestinations** (void)
- void **setThreshold** (**LogLevel** threshold)
- **LogLevel** **getThreshold** () const
- void **msg** (**LogMessage** message)
- void **msg** (**LogLevel** level, const std::string &str)

### Static Public Member Functions

- static **Logger** & **getRootLogger** ()

#### 6.154.1 Detailed Description

A logger class. This class defines a **Logger** (p. 195) to which LogMessages can be sent.

Every **Logger** (p. 195) (except for the rootLogger) has a parent **Logger** (p. 195). The domain of a **Logger** (p. 195) (a string that indicates the origin of LogMessages) is composed by adding a subdomain to the domain of its parent **Logger** (p. 195).

A **Logger** (p. 195) also has a threshold. Every **LogMessage** (p. 197) that have a level that is greater than or equal to the threshold is forwarded to any **LogDestination** (p. 192) connected to this **Logger** (p. 195) as well as to the parent **Logger** (p. 195).

Typical usage of the **Logger** (p. 195) class is to declare a global **Logger** (p. 195) object for each library/-module/component to be used by all classes and methods there.

#### 6.154.2 Constructor & Destructor Documentation

##### 6.154.2.1 Arc::Logger::Logger ( **Logger** & *parent*, const std::string & *subdomain* )

Creates a logger.

Creates a logger. The threshold is inherited from its parent **Logger** (p. 195).

#### Parameters

*parent* The parent **Logger** (p. 195) of the new **Logger** (p. 195).

*subdomain* The subdomain of the new logger.

### 6.154.2.2 `Arc::Logger::Logger ( Logger & parent, const std::string & subdomain, LogLevel threshold )`

Creates a logger.

Creates a logger.

#### Parameters

*parent* The parent **Logger** (p. 195) of the new **Logger** (p. 195).

*subdomain* The subdomain of the new logger.

*threshold* The threshold of the new logger.

### 6.154.2.3 `Arc::Logger::~~Logger ( )`

Destroys a logger.

Destructor

## 6.154.3 Member Function Documentation

### 6.154.3.1 `void Arc::Logger::addDestination ( LogDestination & destination )`

Adds a **LogDestination** (p. 192).

Adds a **LogDestination** (p. 192) to which to forward LogMessages sent to this logger (if they pass the threshold). Since LogDestinatoin should not be copied, the new **LogDestination** (p. 192) is passed by reference and a pointer to it is kept for later use. It is therefore important that the **LogDestination** (p. 192) passed to this **Logger** (p. 195) exists at least as long as the **Logger** (p. 195) itself.

### 6.154.3.2 `static Logger& Arc::Logger::getRootLogger ( ) [static]`

The root **Logger** (p. 195).

This is the root **Logger** (p. 195). It is an ancestor of any other **Logger** (p. 195) and always exists.

### 6.154.3.3 `LogLevel Arc::Logger::getThreshold ( ) const`

Returns the threshold.

Returns the threshold.

#### Returns

The threshold of this **Logger** (p. 195).

### 6.154.3.4 `void Arc::Logger::msg ( LogMessage message )`

Sends a **LogMessage** (p. 197).

Sends a **LogMessage** (p. 197).

### Parameters

*The* **LogMessage** (p. 197) to send.

Referenced by `msg()`, and `Arc::stringto()`.

#### 6.154.3.5 void Arc::Logger::msg ( LogLevel *level*, const std::string & *str* ) [inline]

Logs a message text.

Logs a message text string at the specified **LogLevel**. This is a convenience method to save some typing. It simply creates a **LogMessage** (p. 197) and sends it to the other `msg()` (p. 196) method.

### Parameters

*level* The level of the message.

*str* The message text.

References `msg()`.

#### 6.154.3.6 void Arc::Logger::setThreshold ( LogLevel *threshold* )

Sets the threshold.

This method sets the threshold of the **Logger** (p. 195). Any message sent to this **Logger** (p. 195) that has a level below this threshold will be discarded.

### Parameters

*The* threshold

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

- `Logger.h`

## 6.155 Arc::LoggerFormat Struct Reference

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

- `Logger.h`

## 6.156 Arc::LogMessage Class Reference

A class for log messages.

```
#include <Logger.h>
```

### Public Member Functions

- **LogMessage** (LogLevel *level*, const **IString** &*message*)
- **LogMessage** (LogLevel *level*, const **IString** &*message*, const std::string &*identifier*)
- **LogLevel** `getLevel` () const

## Protected Member Functions

- void **setIdentifier** (std::string identifier)

## Friends

- class **Logger**
- std::ostream & **operator**<< (std::ostream &os, const **LogMessage** &message)

### 6.156.1 Detailed Description

A class for log messages. This class is used to represent log messages internally. It contains the time the message was created, its level, from which domain it was sent, an identifier and the message text itself.

### 6.156.2 Constructor & Destructor Documentation

#### 6.156.2.1 Arc::LogMessage::LogMessage ( LogLevel *level*, const IString & *message* )

Creates a **LogMessage** (p. 197) with the specified level and message text.

This constructor creates a **LogMessage** (p. 197) with the specified level and message text. The time is set automatically, the domain is set by the **Logger** (p. 195) to which the **LogMessage** (p. 197) is sent and the identifier is composed from the process ID and the address of the Thread object corresponding to the calling thread.

#### Parameters

- level* The level of the **LogMessage** (p. 197).
- message* The message text.

#### 6.156.2.2 Arc::LogMessage::LogMessage ( LogLevel *level*, const IString & *message*, const std::string & *identifier* )

Creates a **LogMessage** (p. 197) with the specified attributes.

This constructor creates a **LogMessage** (p. 197) with the specified level, message text and identifier. The time is set automatically and the domain is set by the **Logger** (p. 195) to which the **LogMessage** (p. 197) is sent.

#### Parameters

- level* The level of the **LogMessage** (p. 197).
- message* The message text.
- ident* The identifier of the **LogMessage** (p. 197).

### 6.156.3 Member Function Documentation

#### 6.156.3.1 LogLevel Arc::LogMessage::getLevel ( ) const

Returns the level of the **LogMessage** (p. 197).

Returns the level of the **LogMessage** (p. 197).

## Returns

The level of the **LogMessage** (p. 197).

### 6.156.3.2 void Arc::LogMessage::setIdentifier ( std::string *identifier* ) [protected]

Sets the identifier of the **LogMessage** (p. 197).

The purpose of this method is to allow subclasses (in case there are any) to set the identifier of a **LogMessage** (p. 197).

## Parameters

*The* identifier.

## 6.156.4 Friends And Related Function Documentation

### 6.156.4.1 friend class Logger [friend]

The **Logger** (p. 195) class is a friend.

The **Logger** (p. 195) class must have some privileges (e.g. ability to call the setDomain() method), therefore it is a friend.

### 6.156.4.2 std::ostream& operator<< ( std::ostream & *os*, const LogMessage & *message* ) [friend]

Printing of LogMessages to ostreams.

Output operator so that LogMessages can be printed conveniently by LogDestinations.

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

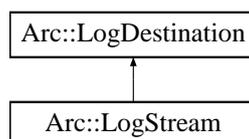
- Logger.h

## 6.157 Arc::LogStream Class Reference

A class for logging to ostreams.

```
#include <Logger.h>
```

Inheritance diagram for Arc::LogStream:



## Public Member Functions

- **LogStream** (std::ostream &destination)

- **LogStream** (std::ostream &destination, const std::string &locale)
- virtual void **log** (const **LogMessage** &message)

### 6.157.1 Detailed Description

A class for logging to ostreams. This class is used for logging to ostreams (cout, cerr, files). It provides synchronization in order to prevent different LogMessages to appear mixed with each other in the stream. In order not to break the synchronization, LogStreams should never be copied. Therefore the copy constructor and assignment operator are private. Furthermore, it is important to keep a **LogStream** (p. 199) object as long as the **Logger** (p. 195) to which it has been registered.

### 6.157.2 Constructor & Destructor Documentation

#### 6.157.2.1 Arc::LogStream::LogStream ( std::ostream & *destination* )

Creates a **LogStream** (p. 199) connected to an ostream.

Creates a **LogStream** (p. 199) connected to the specified ostream. In order not to break synchronization, it is important not to connect more than one **LogStream** (p. 199) object to a certain stream.

#### Parameters

*destination* The ostream to which to erite LogMessages.

#### 6.157.2.2 Arc::LogStream::LogStream ( std::ostream & *destination*, const std::string & *locale* )

Creates a **LogStream** (p. 199) connected to an ostream.

Creates a **LogStream** (p. 199) connected to the specified ostream. The output will be localised to the specified locale.

### 6.157.3 Member Function Documentation

#### 6.157.3.1 virtual void Arc::LogStream::log ( const LogMessage & *message* ) [virtual]

Writes a **LogMessage** (p. 197) to the stream.

This method writes a **LogMessage** (p. 197) to the ostream that is connected to this **LogStream** (p. 199) object. It is synchronized so that not more than one **LogMessage** (p. 197) can be written at a time.

#### Parameters

*message* The **LogMessage** (p. 197) to write.

Implements **Arc::LogDestination** (p. 192).

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

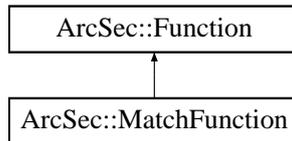
- Logger.h

## 6.158 ArcSec::MatchFunction Class Reference

Evaluate whether `arg1` (value in regular expression) matched `arg0` (lable in regular expression).

```
#include <MatchFunction.h>
```

Inheritance diagram for ArcSec::MatchFunction:



### Public Member Functions

- virtual `AttributeValue * evaluate (AttributeValue *arg0, AttributeValue *arg1, bool check_id=true)`
- virtual `std::list< AttributeValue * > evaluate (std::list< AttributeValue * > args, bool check_id=true)`

### Static Public Member Functions

- static `std::string getFunctionName (std::string datatype)`

#### 6.158.1 Detailed Description

Evaluate whether `arg1` (value in regular expression) matched `arg0` (lable in regular expression).

#### 6.158.2 Member Function Documentation

**6.158.2.1** `virtual AttributeValue* ArcSec::MatchFunction::evaluate ( AttributeValue * arg0, AttributeValue * arg1, bool check_id = true ) [virtual]`

Evaluate two `AttributeValue` (p. 56) objects, and return one `AttributeValue` (p. 56) object

Implements `ArcSec::Function` (p. 167).

**6.158.2.2** `virtual std::list<AttributeValue*> ArcSec::MatchFunction::evaluate ( std::list< AttributeValue * > args, bool check_id = true ) [virtual]`

Evaluate a list of `AttributeValue` (p. 56) objects, and return a list of Attribute objects

Implements `ArcSec::Function` (p. 167).

**6.158.2.3** `static std::string ArcSec::MatchFunction::getFunctionName ( std::string datatype ) [static]`

help function to get the FunctionName

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

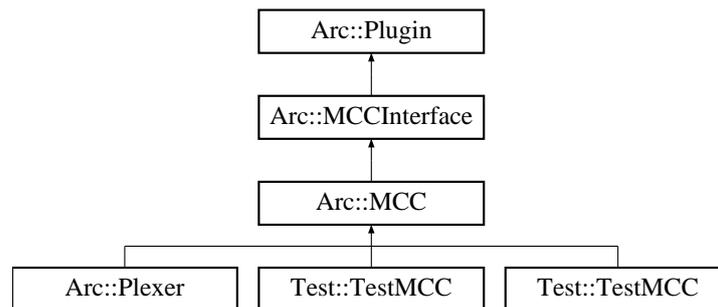
- MatchFunction.h

## 6.159 Arc::MCC Class Reference

**Message** (p. 210) Chain Component - base class for every **MCC** (p. 202) plugin.

```
#include <MCC.h>
```

Inheritance diagram for Arc::MCC:



### Public Member Functions

- **MCC** (**Config** \*)
- virtual void **Next** (**MCCInterface** \*next, const std::string &label="")
- virtual void **AddSecHandler** (**Config** \*cfg, **ArcSec::SecHandler** \*sechandler, const std::string &label="")
- virtual void **Unlink** ()
- virtual **MCC\_Status** process (**Message** &, **Message** &)

### Protected Member Functions

- bool **ProcessSecHandlers** (**Message** &message, const std::string &label="") const

### Protected Attributes

- std::map< std::string, **MCCInterface** \* > **next\_**
- std::map< std::string, std::list< **ArcSec::SecHandler** \* > > **sechandlers\_**

### Static Protected Attributes

- static **Logger** **logger**

#### 6.159.1 Detailed Description

**Message** (p. 210) Chain Component - base class for every **MCC** (p. 202) plugin. This is partially virtual class which defines interface and common functionality for every **MCC** (p. 202) plugin needed for managing of component in a chain.

## 6.159.2 Constructor & Destructor Documentation

### 6.159.2.1 Arc::MCC::MCC ( Config \* ) [inline]

Example constructor - **MCC** (p. 202) takes at least it's configuration subtree

## 6.159.3 Member Function Documentation

### 6.159.3.1 virtual void Arc::MCC::AddSecHandler ( Config \* *cfg*, ArcSec::SecHandler \* *sechandler*, const std::string & *label* = "" ) [virtual]

Add security components/handlers to this **MCC** (p. 202). Security handlers are stacked into a few queues with each queue identified by its label. The queue labelled 'incoming' is executed for every 'request' message after the message is processed by the **MCC** (p. 202) on the service side and before processing on the client side. The queue labelled 'outgoing' is run for response message before it is processed by **MCC** (p. 202) algorithms on the service side and after processing on the client side. Those labels are just a matter of agreement and some MCCs may implement different queues executed at various message processing steps.

### 6.159.3.2 virtual void Arc::MCC::Next ( MCCInterface \* *next*, const std::string & *label* = "" ) [virtual]

Add reference to next **MCC** (p. 202) in chain. This method is called by **Loader** (p. 191) for every potentially labeled link to next component which implements **MCCInterface** (p. 207). If next is NULL corresponding link is removed.

Reimplemented in **Arc::Plexer** (p. 248).

### 6.159.3.3 virtual MCC\_Status Arc::MCC::process ( Message &, Message & ) [inline, virtual]

Dummy **Message** (p. 210) processing method. Just a placeholder.

Implements **Arc::MCCInterface** (p. 207).

Reimplemented in **Arc::Plexer** (p. 248).

### 6.159.3.4 bool Arc::MCC::ProcessSecHandlers ( Message & *message*, const std::string & *label* = "" ) const [protected]

Executes security handlers of specified queue. Returns true if the message is authorized for further processing or if there are no security handlers which implement authorization functionality. This is a convenience method and has to be called by the implementation of the **MCC** (p. 202).

### 6.159.3.5 virtual void Arc::MCC::Unlink ( ) [virtual]

Removing all links. Useful for destroying chains.

## 6.159.4 Field Documentation

### 6.159.4.1 `Logger Arc::MCC::logger` [`static`, `protected`]

A logger for MCCs.

A logger intended to be the parent of loggers in the different MCCs.

Reimplemented in `Arc::Plexer` (p. 248).

### 6.159.4.2 `std::map<std::string, MCCInterface *> Arc::MCC::next_` [`protected`]

Set of labeled "next" components. Each implemented `MCC` (p. 202) must call `process()` (p. 203) method of corresponding `MCCInterface` (p. 207) from this set in own `process()` (p. 203) method.

### 6.159.4.3 `std::map<std::string, std::list<ArcSec::SecHandler *> > Arc::MCC::sechandlers_` [`protected`]

Set of labeled authentication and authorization handlers. `MCC` (p. 202) calls sequence of handlers at specific point depending on associated identifier. In most cases those are "in" and "out" for incoming and outgoing messages correspondingly.

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

- `MCC.h`

## 6.160 `Arc::MCC_Status` Class Reference

A class for communication of `MCC` (p. 202) processing results.

```
#include <MCC_Status.h>
```

### Public Member Functions

- `MCC_Status` (`StatusKind` kind=`STATUS_UNDEFINED`, `const std::string &origin="???", const std::string &explanation="No explanation."`)
- `bool isOk () const`
- `StatusKind getKind () const`
- `const std::string &getOrigin () const`
- `const std::string &getExplanation () const`
- `operator std::string () const`
- `operator bool (void) const`
- `bool operator! (void) const`

### 6.160.1 Detailed Description

A class for communication of `MCC` (p. 202) processing results. This class is used to communicate result status between MCCs. It contains a status kind, a string specifying the origin (`MCC` (p. 202)) of the status object and an explanation.

## 6.160.2 Constructor & Destructor Documentation

**6.160.2.1** `Arc::MCC_Status::MCC_Status ( StatusKind kind = STATUS_UNDEFINED, const std::string & origin = "???", const std::string & explanation = "No explanation." )`

The constructor.

Creates a `MCC_Status` (p. 204) object.

### Parameters

*kind* The StatusKind (default: STATUS\_UNDEFINED)

*origin* The origin MCC (p. 202) (default: "??")

*explanation* An explanation (default: "No explanation.")

## 6.160.3 Member Function Documentation

**6.160.3.1** `const std::string& Arc::MCC_Status::getExplanation ( ) const`

Returns an explanation.

This method returns an explanation of this object.

### Returns

An explanation of this object.

**6.160.3.2** `StatusKind Arc::MCC_Status::getKind ( ) const`

Returns the status kind.

Returns the status kind of this object.

### Returns

The status kind of this object.

**6.160.3.3** `const std::string& Arc::MCC_Status::getOrigin ( ) const`

Returns the origin.

This method returns a string specifying the origin MCC (p. 202) of this object.

### Returns

A string specifying the origin MCC (p. 202) of this object.

**6.160.3.4** `bool Arc::MCC_Status::isOk ( ) const`

Is the status kind ok?

This method returns true if the status kind of this object is STATUS\_OK

**Returns**

true if kind==STATUS\_OK

Referenced by operator bool(), and operator!().

**6.160.3.5 Arc::MCC\_Status::operator bool ( void ) const [inline]**

Is the status kind ok?

This method returns true if the status kind of this object is STATUS\_OK

**Returns**

true if kind==STATUS\_OK

References isOk().

**6.160.3.6 Arc::MCC\_Status::operator std::string ( ) const**

Conversion to string.

This operator converts a **MCC\_Status** (p. 204) object to a string.

**6.160.3.7 bool Arc::MCC\_Status::operator! ( void ) const [inline]**

not operator

Returns true if the status kind is not OK

**Returns**

true if kind!=STATUS\_OK

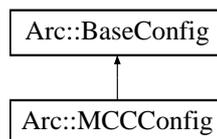
References isOk().

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

- MCC\_Status.h

**6.161 Arc::MCCConfig Class Reference**

Inheritance diagram for Arc::MCCConfig:

**Public Member Functions**

- virtual **XMLNode MakeConfig (XMLNode cfg) const**

## 6.161.1 Member Function Documentation

### 6.161.1.1 virtual XMLNode Arc::MCCConfig::MakeConfig ( XMLNode *cfg* ) const [virtual]

Adds configuration part corresponding to stored information into common configuration tree supplied in 'cfg' argument. Returns reference to XML node representing configuration of **ModuleManager** (p. 218)

Reimplemented from **Arc::BaseConfig** (p. 61).

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

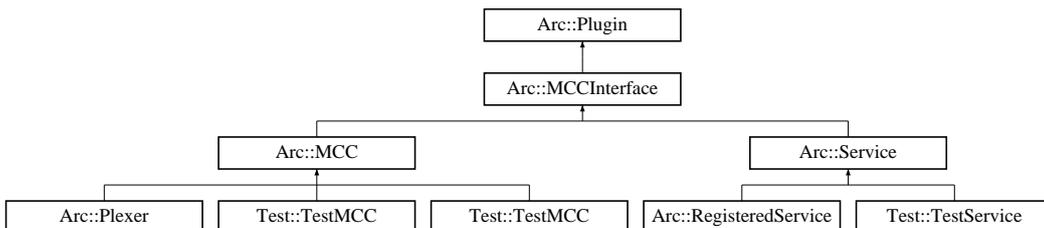
- MCC.h

## 6.162 Arc::MCCInterface Class Reference

Interface for communication between **MCC** (p. 202), **Service** (p. 285) and **Plexer** (p. 247) objects.

```
#include <MCC.h>
```

Inheritance diagram for Arc::MCCInterface:



## Public Member Functions

- virtual **MCC\_Status** process (Message &request, Message &response)=0

### 6.162.1 Detailed Description

Interface for communication between **MCC** (p. 202), **Service** (p. 285) and **Plexer** (p. 247) objects. The Interface consists of the method **process()** (p. 207) which is called by the previous **MCC** (p. 202) in the chain. For memory management policies please read the description of the **Message** (p. 210) class.

### 6.162.2 Member Function Documentation

#### 6.162.2.1 virtual MCC\_Status Arc::MCCInterface::process ( Message & *request*, Message & *response* ) [pure virtual]

Method for processing of requests and responses. This method is called by preceding **MCC** (p. 202) in chain when a request needs to be processed. This method must call similar method of next **MCC** (p. 202) in chain unless any failure happens. Result returned by call to next **MCC** (p. 202) should be processed and passed back to previous **MCC** (p. 202). In case of failure this method is expected to generate valid error response and return it back to previous **MCC** (p. 202) without calling the next one.

**Parameters**

*request* The request that needs to be processed.

*response* A **Message** (p. 210) object that will contain the response of the request when the method returns.

**Returns**

An object representing the status of the call.

Implemented in **Test::TestService** (p. 317), **Arc::MCC** (p. 203), and **Arc::Plexer** (p. 248).

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

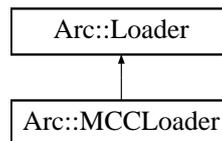
- MCC.h

**6.163 Arc::MCCLoader Class Reference**

Creator of **Message** (p. 210) Component Chains (**MCC** (p. 202)).

```
#include <MCCLoader.h>
```

Inheritance diagram for Arc::MCCLoader:

**Public Member Functions**

- **MCCLoader** (**Config** &cfg)
- **~MCCLoader** ()
- **MCC \* operator[]** (const std::string &id)

**6.163.1 Detailed Description**

Creator of **Message** (p. 210) Component Chains (**MCC** (p. 202)). This class processes XML configuration and creates message chains. Accepted configuration is defined by XML schema mcc.xsd. Supported components are of types **MCC** (p. 202), **Service** (p. 285) and **Plexer** (p. 247). **MCC** (p. 202) and **Service** (p. 285) are loaded from dynamic libraries. For **Plexer** (p. 247) only internal implementation is supported. This object is also a container for loaded componets. All components and chains are destroyed if this object is destroyed. Chains are created in 2 steps. First all components are loaded and corresponding objects are created. Constructors are supplied with corresponding configuration subtrees. During next step components are linked together by calling their Next() methods. Each call creates labeled link to next component in a chain. 2 step method has an advantage over single step because it allows loops in chains and makes loading procedure more simple. But that also means during short period of time components are only partly configured. Components in such state must produce proper error response if **Message** (p. 210) arrives. Note: Current implementation requires all components and links to be labeled. All labels must be unique. Future implementation will be able to assign labels automatically.

## 6.163.2 Constructor & Destructor Documentation

### 6.163.2.1 Arc::MCCLoader::MCCLoader ( Config & *cfg* )

Constructor that takes whole XML configuration and creates component chains

### 6.163.2.2 Arc::MCCLoader::~~MCCLoader ( )

Destructor destroys all components created by constructor

## 6.163.3 Member Function Documentation

### 6.163.3.1 MCC\* Arc::MCCLoader::operator[] ( const std::string & *id* )

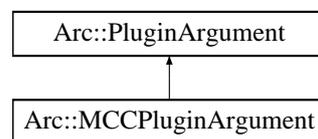
Access entry MCCs in chains. Those are components exposed for external access using 'entry' attribute

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

- MCCLoader.h

## 6.164 Arc::MCCPluginArgument Class Reference

Inheritance diagram for Arc::MCCPluginArgument:



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

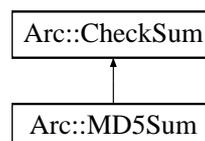
- MCC.h

## 6.165 Arc::MD5Sum Class Reference

Implementation of MD5 checksum.

```
#include <Checksum.h>
```

Inheritance diagram for Arc::MD5Sum:



### 6.165.1 Detailed Description

Implementation of MD5 checksum.

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

- CheckSum.h

### 6.166 Arc::MemoryAllocationException Class Reference

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

- ByteArray.h

### 6.167 Arc::Message Class Reference

Object being passed through chain of MCCs.

```
#include <Message.h>
```

#### Public Member Functions

- **Message** (void)
- **Message** (**Message** &msg)
- **Message** (long msg\_ptr\_addr)
- **~Message** (void)
- **Message** & **operator=** (**Message** &msg)
- **MessagePayload** \* **Payload** (void)
- **MessagePayload** \* **Payload** (**MessagePayload** \*payload)
- **MessageAttributes** \* **Attributes** (void)
- **MessageAuth** \* **Auth** (void)
- **MessageContext** \* **Context** (void)
- **MessageAuthContext** \* **AuthContext** (void)
- void **Context** (**MessageContext** \*ctx)
- void **AuthContext** (**MessageAuthContext** \*auth\_ctx)

#### 6.167.1 Detailed Description

Object being passed through chain of MCCs. An instance of this class refers to objects with main content (**MessagePayload** (p. 218)), authentication/authorization information (**MessageAuth** (p. 215)) and common purpose attributes (**MessageAttributes** (p. 212)). **Message** (p. 210) class does not manage pointers to objects and their content. It only serves for grouping those objects. **Message** (p. 210) objects are supposed to be processed by MCCs and Services implementing **MCCInterface** (p. 207) method process(). All objects constituting content of **Message** (p. 210) object are subject to following policies:

1. All objects created inside call to process() method using new command must be explicitly destroyed within same call using delete command with following exceptions. a) Objects which are assigned to 'response' **Message** (p. 210). b) Objects whose management is completely acquired by objects assigned to 'response' **Message** (p. 210).

2. All objects not created inside call to process() method are not explicitly destroyed within that call with following exception. a) Objects which are part of 'response' Method returned from call to next's process() method. Unless those objects are passed further to calling process(), of course.
3. It is not allowed to make 'response' point to same objects as 'request' does on entry to process() method. That is needed to avoid double destruction of same object. (Note: if in a future such need arises it may be solved by storing additional flags in **Message** (p. 210) object).
4. It is allowed to change content of pointers of 'request' **Message** (p. 210). Calling process() method must not rely on that object to stay intact.
5. Called process() method should either fill 'response' **Message** (p. 210) with pointers to valid objects or to keep them intact. This makes it possible for calling process() to preload 'response' with valid error message.

## 6.167.2 Constructor & Destructor Documentation

### 6.167.2.1 Arc::Message::Message ( void ) [inline]

true if auth\_ctx\_ was created internally Dummy constructor

### 6.167.2.2 Arc::Message::Message ( Message & msg ) [inline]

Copy constructor. Ensures shallow copy.

### 6.167.2.3 Arc::Message::Message ( long msg\_ptr\_addr )

Copy constructor. Used by language bindings

### 6.167.2.4 Arc::Message::~~Message ( void ) [inline]

Destructor does not affect referred objects except those created internally

## 6.167.3 Member Function Documentation

### 6.167.3.1 MessageAttributes\* Arc::Message::Attributes ( void ) [inline]

Returns a pointer to the current attributes object or creates it if no attributes object has been assigned.

### 6.167.3.2 MessageAuth\* Arc::Message::Auth ( void ) [inline]

Returns a pointer to the current authentication/authorization object or creates it if no object has been assigned.

### 6.167.3.3 MessageAuthContext\* Arc::Message::AuthContext ( void ) [inline]

Returns a pointer to the current auth\* context object or creates it if no object has been assigned.

**6.167.3.4 void Arc::Message::AuthContext ( MessageAuthContext \* *auth\_ctx* ) [inline]**

Assigns auth\* context object

**6.167.3.5 void Arc::Message::Context ( MessageContext \* *ctx* ) [inline]**

Assigns message context object

**6.167.3.6 MessageContext\* Arc::Message::Context ( void ) [inline]**

Returns a pointer to the current context object or creates it if no object has been assigned. Last case should happen only if first MCC (p. 202) in a chain is connectionless like one implementing UDP protocol.

**6.167.3.7 Message& Arc::Message::operator= ( Message & *msg* ) [inline]**

Assignment. Ensures shallow copy.

**6.167.3.8 MessagePayload\* Arc::Message::Payload ( void ) [inline]**

Returns pointer to current payload or NULL if no payload assigned.

**6.167.3.9 MessagePayload\* Arc::Message::Payload ( MessagePayload \* *payload* ) [inline]**

Replaces payload with new one. Returns the old one.

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

- Message.h

## 6.168 Arc::MessageAttributes Class Reference

A class for storage of attribute values.

```
#include <MessageAttributes.h>
```

### Public Member Functions

- **MessageAttributes** ()
- void **set** (const std::string &key, const std::string &value)
- void **add** (const std::string &key, const std::string &value)
- void **removeAll** (const std::string &key)
- void **remove** (const std::string &key, const std::string &value)
- int **count** (const std::string &key) const
- const std::string & **get** (const std::string &key) const
- **AttributeIterator** **getAll** (const std::string &key) const
- **AttributeIterator** **getAll** (void) const

## Protected Attributes

- `AttrMap attributes_`

### 6.168.1 Detailed Description

A class for storage of attribute values. This class is used to store attributes of messages. All attribute keys and their corresponding values are stored as strings. Any key or value that is not a string must thus be represented as a string during storage. Furthermore, an attribute is usually a key-value pair with a unique key, but there may also be multiple such pairs with equal keys.

The key of an attribute is composed by the name of the **Message** (p. 210) Chain Component (**MCC** (p. 202)) which produce it and the name of the attribute itself with a colon (:) in between, i.e. `MCC_Name:Attribute_Name`. For example, the key of the "Content-Length" attribute of the HTTP **MCC** (p. 202) is thus "HTTP:Content-Length".

There are also "global attributes", which may be produced by different **MCCs** depending on the configuration. The keys of such attributes are NOT prefixed by the name of the producing **MCC** (p. 202). Before any new global attribute is introduced, it must be agreed upon by the core development team and added below. The global attributes decided so far are:

- `Request-URI` Identifies the service to which the message shall be sent. This attribute is produced by e.g. the HTTP **MCC** (p. 202) and used by the plexer for routing the message to the appropriate service.

### 6.168.2 Constructor & Destructor Documentation

#### 6.168.2.1 `Arc::MessageAttributes::MessageAttributes ( )`

The default constructor.

This is the default constructor of the **MessageAttributes** (p. 212) class. It constructs an empty object that initially contains no attributes.

### 6.168.3 Member Function Documentation

#### 6.168.3.1 `void Arc::MessageAttributes::add ( const std::string & key, const std::string & value )`

Adds a value to an attribute.

This method adds a new value to an attribute. Any previous value will be preserved, i.e. the attribute may become multiple valued.

#### Parameters

- key* The key of the attribute.
- value* The (new) value of the attribute.

#### 6.168.3.2 `int Arc::MessageAttributes::count ( const std::string & key ) const`

Returns the number of values of an attribute.

Returns the number of values of an attribute that matches a certain key.

**Parameters**

*key* The key of the attribute for which to count values.

**Returns**

The number of values that corresponds to the key.

**6.168.3.3** `const std::string& Arc::MessageAttributes::get ( const std::string & key ) const`

Returns the value of a single-valued attribute.

This method returns the value of a single-valued attribute. If the attribute is not single valued (i.e. there is no such attribute or it is a multiple-valued attribute) an empty string is returned.

**Parameters**

*key* The key of the attribute for which to return the value.

**Returns**

The value of the attribute.

**6.168.3.4** `AttributeIterator Arc::MessageAttributes::getAll ( const std::string & key ) const`

Access the value(s) of an attribute.

This method returns an **AttributeIterator** (p. 53) that can be used to access the values of an attribute.

**Parameters**

*key* The key of the attribute for which to return the values.

**Returns**

An **AttributeIterator** (p. 53) for access of the values of the attribute.

**6.168.3.5** `void Arc::MessageAttributes::remove ( const std::string & key, const std::string & value )`

Removes one value of an attribute.

This method removes a certain value from the attribute that matches a certain key.

**Parameters**

*key* The key of the attribute from which the value shall be removed.

*value* The value to remove.

**6.168.3.6** `void Arc::MessageAttributes::removeAll ( const std::string & key )`

Removes all attributes with a certain key.

This method removes all attributes that match a certain key.

**Parameters**

*key* The key of the attributes to remove.

**6.168.3.7 void Arc::MessageAttributes::set ( const std::string & key, const std::string & value )**

Sets a unique value of an attribute.

This method removes any previous value of an attribute and sets the new value as the only value.

**Parameters**

*key* The key of the attribute.

*value* The (new) value of the attribute.

**6.168.4 Field Documentation****6.168.4.1 AttrMap Arc::MessageAttributes::attributes\_ [protected]**

Internal storage of attributes.

An AttrMap (multimap) in which all attributes (key-value pairs) are stored.

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

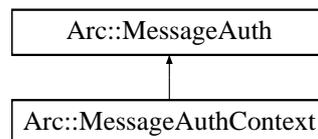
- MessageAttributes.h

**6.169 Arc::MessageAuth Class Reference**

Contains authenticity information, authorization tokens and decisions.

```
#include <MessageAuth.h>
```

Inheritance diagram for Arc::MessageAuth:

**Public Member Functions**

- void **set** (const std::string &key, **SecAttr** \*value)
- void **remove** (const std::string &key)
- **SecAttr** \* **get** (const std::string &key)
- **SecAttr** \* **operator**[ ] (const std::string &key)
- bool **Export** (**SecAttrFormat** format, **XMLNode** &val) const
- **MessageAuth** \* **Filter** (const std::list< std::string > &selected\_keys, const std::list< std::string > &rejected\_keys)

### 6.169.1 Detailed Description

Contains authenticity information, authorization tokens and decisions. This class only supports string keys and **SecAttr** (p. 280) values.

### 6.169.2 Member Function Documentation

#### 6.169.2.1 `bool Arc::MessageAuth::Export ( SecAttrFormat format, XMLNode & val ) const`

Returns properly catenated attributes in specified format.

Content of XML node at is replaced with generated information if XML tree is empty. If tree at is not empty then **Export()** (p. 216) tries to merge generated information to already existing like everything would be generated inside same **Export()** (p. 216) method. If does not represent valid node then new XML tree is created.

#### 6.169.2.2 `MessageAuth* Arc::MessageAuth::Filter ( const std::list< std::string > & selected_keys, const std::list< std::string > & rejected_keys )`

Creates new instance of **MessageAuth** (p. 215) with attributes filtered.

In new instance all attributes with keys listed in are removed. If is not empty only corresponding attributes are transferred to new instance. Created instance does not own refered attributes. Hence parent instance must not be deleted as long as this one is in use.

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

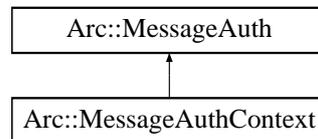
- MessageAuth.h

## 6.170 Arc::MessageAuthContext Class Reference

Handler for content of message auth\* context.

```
#include <Message.h>
```

Inheritance diagram for Arc::MessageAuthContext:



### 6.170.1 Detailed Description

Handler for content of message auth\* context. This class is a container for authorization and authentication information. It gets associated with **Message** (p. 210) object usually by first **MCC** (p. 202) in a chain and is kept as long as connection persists.

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

- Message.h

## 6.171 Arc::MessageContext Class Reference

Handler for content of message context.

```
#include <Message.h>
```

### Public Member Functions

- void **Add** (const std::string &name, **MessageContextElement** \*element)

#### 6.171.1 Detailed Description

Handler for content of message context. This class is a container for objects derived from **MessageContextElement** (p. 217). It gets associated with **Message** (p. 210) object usually by first **MCC** (p. 202) in a chain and is kept as long as connection persists.

#### 6.171.2 Member Function Documentation

##### 6.171.2.1 void Arc::MessageContext::Add ( const std::string & *name*, MessageContextElement \* *element* )

Provided element is taken over by this class. It is remembered by it and destroyed when this class is destroyed.

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

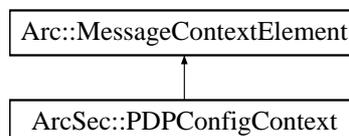
- Message.h

## 6.172 Arc::MessageContextElement Class Reference

Top class for elements contained in message context.

```
#include <Message.h>
```

Inheritance diagram for Arc::MessageContextElement:



#### 6.172.1 Detailed Description

Top class for elements contained in message context. Objects of classes inherited with this one may be stored in **MessageContext** (p. 217) container.

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

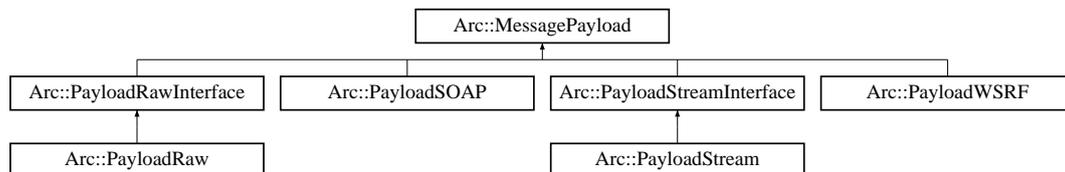
- Message.h

## 6.173 Arc::MessagePayload Class Reference

Base class for content of message passed through chain.

```
#include <Message.h>
```

Inheritance diagram for Arc::MessagePayload:



### 6.173.1 Detailed Description

Base class for content of message passed through chain. It's not intended to be used directly. Instead functional classes must be derived from it.

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

- Message.h

## 6.174 Arc::ModuleDesc Class Reference

Description of loadable module.

```
#include <Plugin.h>
```

### 6.174.1 Detailed Description

Description of loadable module. This class is used for reports

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

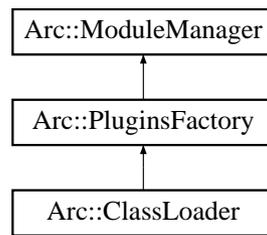
- Plugin.h

## 6.175 Arc::ModuleManager Class Reference

Manager of shared libraries.

```
#include <ModuleManager.h>
```

Inheritance diagram for Arc::ModuleManager:



## Public Member Functions

- **ModuleManager** (**XMLNode** cfg)
- **Glib::Module \* load** (const std::string &name, bool probe=false)
- std::string **find** (const std::string &name)
- **Glib::Module \* reload** (Glib::Module \*module)
- void **unload** (Glib::Module \*module)
- void **unload** (const std::string &name)
- std::string **findLocation** (const std::string &name)
- bool **makePersistent** (Glib::Module \*module)
- bool **makePersistent** (const std::string &name)
- void **setCfg** (**XMLNode** cfg)

### 6.175.1 Detailed Description

Manager of shared libraries. This class loads shared libraries/modules. There supposed to be created one instance of it per executable. In such circumstances it would cache handles to loaded modules and not load them multiple times.

### 6.175.2 Constructor & Destructor Documentation

#### 6.175.2.1 Arc::ModuleManager::ModuleManager ( XMLNode cfg )

Cache of handles of loaded modules Constructor. It is supposed to process corresponding configuration subtree and tune module loading parameters accordingly.

### 6.175.3 Member Function Documentation

#### 6.175.3.1 std::string Arc::ModuleManager::find ( const std::string & name )

Finds loadable module by 'name' looking in same places as **load()** (p. 219) does, but does not load it.

#### 6.175.3.2 std::string Arc::ModuleManager::findLocation ( const std::string & name )

Finds shared library corresponding to module 'name' and returns path to it

#### 6.175.3.3 Glib::Module\* Arc::ModuleManager::load ( const std::string & name, bool probe = false )

Finds module 'name' in cache or loads corresponding loadable module

#### 6.175.3.4 `bool Arc::ModuleManager::makePersistent ( const std::string & name )`

Make sure this module is never unloaded. Even if `unload()` (p. 220) is called.

#### 6.175.3.5 `bool Arc::ModuleManager::makePersistent ( Glib::Module * module )`

Make sure this module is never unloaded. Even if `unload()` (p. 220) is called.

#### 6.175.3.6 `Glib::Module* Arc::ModuleManager::reload ( Glib::Module * module )`

Reload module previously loaded in probe mode. New module is loaded with all symbols resolved and old module handler is unloaded. In case of error old module is not unloaded.

#### 6.175.3.7 `void Arc::ModuleManager::setCfg ( XMLNode cfg )`

Input the configuration subtree, and trigger the module loading (do almost the same as the Constructor); It is function designed for `ClassLoader` (p. 69) to adopt the singleton pattern

#### 6.175.3.8 `void Arc::ModuleManager::unload ( const std::string & name )`

Unload module by its name

#### 6.175.3.9 `void Arc::ModuleManager::unload ( Glib::Module * module )`

Unload module by its identifier

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

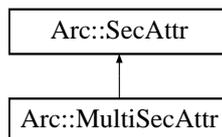
- ModuleManager.h

## 6.176 `Arc::MultiSecAttr` Class Reference

Container of multiple `SecAttr` (p. 280) attributes.

```
#include <SecAttr.h>
```

Inheritance diagram for `Arc::MultiSecAttr`:



### Public Member Functions

- virtual **operator bool** () const
- virtual **bool Export** (`SecAttrFormat` format, `XMLNode &val`) const

### 6.176.1 Detailed Description

Container of multiple **SecAttr** (p. 280) attributes. This class combines multiple attributes. It's export/import methods catenate results of underlying objects. Primary meaning of this class is to serve as base for classes implementing multi level hierarchical tree-like descriptions of user identity. It may also be used for collecting information of same source or kind. Like all information extracted from X509 certificate.

### 6.176.2 Member Function Documentation

#### 6.176.2.1 virtual bool Arc::MultiSecAttr::Export ( SecAttrFormat *format*, XMLNode & *val* ) const [virtual]

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute. XML node referenced by is turned into top level element of specified format.

Reimplemented from **Arc::SecAttr** (p. 281).

#### 6.176.2.2 virtual Arc::MultiSecAttr::operator bool ( ) const [virtual]

This function should return false if the value is to be considered null, e.g. if it hasn't been set or initialized. In other cases it should return true.

Reimplemented from **Arc::SecAttr** (p. 281).

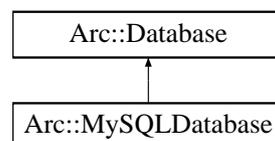
The documentation for this class was generated from the following file:

- SecAttr.h

## 6.177 Arc::MySQLDatabase Class Reference

```
#include <MysqlWrapper.h>
```

Inheritance diagram for Arc::MySQLDatabase:



### Public Member Functions

- virtual bool **connect** (std::string &dbname, std::string &user, std::string &password)
- virtual bool **isconnected** () const
- virtual void **close** ()
- virtual bool **enable\_ssl** (const std::string keyfile="", const std::string certfile="", const std::string cafile="", const std::string capath="")
- virtual bool **shutdown** ()

### 6.177.1 Detailed Description

Implement the database accessing interface in **DBInterface.h** (p. ??) by using mysql client library for accessing mysql database

### 6.177.2 Member Function Documentation

#### 6.177.2.1 virtual void Arc::MySQLDatabase::close ( ) [virtual]

Close the connection with database server

Implements **Arc::Database** (p. 99).

#### 6.177.2.2 virtual bool Arc::MySQLDatabase::connect ( std::string & *dbname*, std::string & *user*, std::string & *password* ) [virtual]

Do connection with database server

##### Parameters

*dbname* The database name which will be used.

*user* The username which will be used to access database.

*password* The password which will be used to access database.

Implements **Arc::Database** (p. 99).

#### 6.177.2.3 virtual bool Arc::MySQLDatabase::enable\_ssl ( const std::string *keyfile* = "", const std::string *certfile* = "", const std::string *cafile* = "", const std::string *capath* = "" ) [virtual]

Enable ssl communication for the connection

##### Parameters

*keyfile* The location of key file.

*certfile* The location of certificate file.

*cafile* The location of ca file.

*capath* The location of ca directory

Implements **Arc::Database** (p. 100).

#### 6.177.2.4 virtual bool Arc::MySQLDatabase::isconnected ( ) const [inline, virtual]

Get the connection status

Implements **Arc::Database** (p. 100).

#### 6.177.2.5 virtual bool Arc::MySQLDatabase::shutdown ( ) [virtual]

Ask database server to shutdown

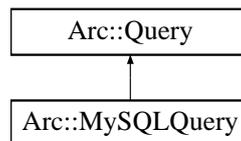
Implements **Arc::Database** (p. 100).

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

- MysqlWrapper.h

## 6.178 Arc::MySQLQuery Class Reference

Inheritance diagram for Arc::MySQLQuery:



### Public Member Functions

- virtual int **get\_num\_columns** ()
- virtual int **get\_num\_rows** ()
- virtual bool **execute** (const std::string &sqlstr)
- virtual QueryRowResult **get\_row** (int row\_number) const
- virtual QueryRowResult **get\_row** () const
- virtual std::string **get\_row\_field** (int row\_number, std::string &field\_name)
- virtual bool **get\_array** (std::string &sqlstr, QueryArrayResult &result, std::vector< std::string > &arguments)

### 6.178.1 Member Function Documentation

#### 6.178.1.1 virtual bool Arc::MySQLQuery::execute ( const std::string & *sqlstr* ) [virtual]

Execute the query

##### Parameters

*sqlstr* The sql sentence used to query

Implements **Arc::Query** (p. 260).

#### 6.178.1.2 virtual bool Arc::MySQLQuery::get\_array ( std::string & *sqlstr*, QueryArrayResult & *result*, std::vector< std::string > & *arguments* ) [virtual]

**Query** (p. 259) the database by using some parameters into sql sentence e.g. "select table.value from table where table.name = ?"

##### Parameters

*sqlstr* The sql sentence with some parameters marked with "?".

*result* The result in an array which includes all of the value in query result.

*arguments* The argument list which should exactly correspond with the parametes in sql sentence.

Implements **Arc::Query** (p. 260).

**6.178.1.3** `virtual int Arc::MySQLQuery::get_num_columns ( ) [virtual]`

Get the column number in the query result

Implements `Arc::Query` (p. 260).

**6.178.1.4** `virtual int Arc::MySQLQuery::get_num_rows ( ) [virtual]`

Get the row number in the query result

Implements `Arc::Query` (p. 260).

**6.178.1.5** `virtual QueryRowResult Arc::MySQLQuery::get_row ( int row_number ) const [virtual]`

Get the value of one row in the query result

**Parameters**

*row\_number* The number of the row

**Returns**

A vector includes all the values in the row

Implements `Arc::Query` (p. 260).

**6.178.1.6** `virtual QueryRowResult Arc::MySQLQuery::get_row ( ) const [virtual]`

Get the value of one row in the query result, the row number will be automatically increased each time the method is called

Implements `Arc::Query` (p. 261).

**6.178.1.7** `virtual std::string Arc::MySQLQuery::get_row_field ( int row_number, std::string & field_name ) [virtual]`

Get the value of one specific field in one specific row

**Parameters**

*row\_number* The row number inside the query result

*field\_name* The field name for the value which will be return

**Returns**

The value of the specified field in the specified row

Implements `Arc::Query` (p. 261).

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

- `MysqlWrapper.h`

## 6.179 Arc::NotificationType Class Reference

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

- JobDescription.h

## 6.180 Arc::NS Class Reference

### Public Member Functions

- **NS** (void)
- **NS** (const char \*prefix, const char \*uri)
- **NS** (const char \*nslst[ ][2])

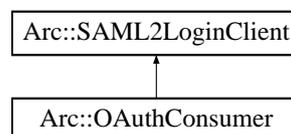
The documentation for this class was generated from the following file:

- XMLNode.h

## 6.181 Arc::OAuthConsumer Class Reference

```
#include <OAuthConsumer.h>
```

Inheritance diagram for Arc::OAuthConsumer:



### Public Member Functions

- **OAuthConsumer** (const **MCCConfig** cfg, const **URL** url, std::list< std::string > idp\_stack)
- **MCC\_Status parseDN** (std::string \*dn)
- **MCC\_Status approveCSR** (const std::string approve\_page)
- **MCC\_Status pushCSR** (const std::string b64\_pub\_key, const std::string pub\_key\_hash, std::string \*approve\_page)
- **MCC\_Status storeCert** (const std::string cert\_path, const std::string auth\_token, const std::string b64\_dn)

### Protected Member Functions

- **MCC\_Status processLogin** (const std::string username="", const std::string password="")

### 6.181.1 Detailed Description

The OAuth functionality depends on the availability of the liboauth C-bindings library

## 6.181.2 Constructor & Destructor Documentation

### 6.181.2.1 `Arc::OAuthConsumer::OAuthConsumer ( const MCCConfig cfg, const URL url, std::list< std::string > idp_stack )`

Construct an OAuth consumer with *url* as service provider. *idp\_name* is currently ignored, since the idp to which the SAML2 redirect will take place is presently a hardcoded value on the SAML2 SP side. This is expected to change in the future.

## 6.181.3 Member Function Documentation

### 6.181.3.1 `MCC_Status Arc::OAuthConsumer::approveCSR ( const std::string approve_page ) [virtual]`

Unsupported placeholder function until Confusa supports OAuth.

Implements `Arc::SAML2LoginClient` (p. 274).

### 6.181.3.2 `MCC_Status Arc::OAuthConsumer::parseDN ( std::string * dn ) [virtual]`

Unsupported placeholder function until Confusa supports OAuth.

Implements `Arc::SAML2LoginClient` (p. 274).

### 6.181.3.3 `MCC_Status Arc::OAuthConsumer::processLogin ( const std::string username = "", const std::string password = "" ) [protected, virtual]`

Main function performing all the OAuth login steps. Username and password will be ignored.

Implements `Arc::SAML2LoginClient` (p. 275).

### 6.181.3.4 `MCC_Status Arc::OAuthConsumer::pushCSR ( const std::string b64_pub_key, const std::string pub_key_hash, std::string * approve_page ) [virtual]`

Unsupported placeholder function until Confusa supports OAuth.

Implements `Arc::SAML2LoginClient` (p. 274).

### 6.181.3.5 `MCC_Status Arc::OAuthConsumer::storeCert ( const std::string cert_path, const std::string auth_token, const std::string b64_dn ) [virtual]`

Unsupported placeholder function until Confusa supports OAuth.

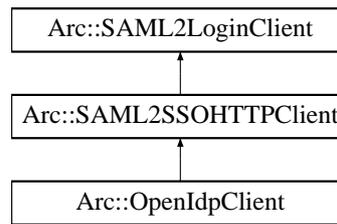
Implements `Arc::SAML2LoginClient` (p. 274).

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

- `OAuthConsumer.h`

## 6.182 `Arc::OpenIdpClient` Class Reference

Inheritance diagram for `Arc::OpenIdpClient`:



## Protected Member Functions

- **MCC\_Status processIdPLogin** (const std::string username, const std::string password)
- **MCC\_Status processConsent** ()
- **MCC\_Status processIdP2Confusa** ()

### 6.182.1 Member Function Documentation

#### 6.182.1.1 MCC\_Status Arc::OpenIdpClient::processConsent ( ) [protected, virtual]

If the IdP has a consent module and the user has not saved her consent, this method will ask the user for consent to transmission of her data to Confusa

Implements **Arc::SAML2SSOHTTPClient** (p. 276).

#### 6.182.1.2 MCC\_Status Arc::OpenIdpClient::processIdP2Confusa ( ) [protected, virtual]

Redirects the user back from identity provider to the Confusa SP

Implements **Arc::SAML2SSOHTTPClient** (p. 276).

#### 6.182.1.3 MCC\_Status Arc::OpenIdpClient::processIdPLogin ( const std::string *username*, const std::string *password* ) [protected, virtual]

Actual identity provider parsers for next three methods implemented in subdirectory idp/

Parse identity provider login page and submit username and password in the previsioned way

Implements **Arc::SAML2SSOHTTPClient** (p. 276).

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

- OpenIdpClient.h

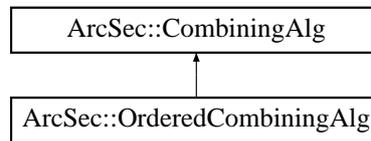
## 6.183 Arc::OptionParser Class Reference

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

- OptionParser.h

## 6.184 ArcSec::OrderedCombiningAlg Class Reference

Inheritance diagram for ArcSec::OrderedCombiningAlg:



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

- OrderedAlg.h

## 6.185 passwd Struct Reference

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

- win32.h

## 6.186 Arc::PathIterator Class Reference

Class to iterate through elements of path.

```
#include <URL.h>
```

### Public Member Functions

- **PathIterator** (const std::string &path, bool end=false)
- **PathIterator & operator++** ()
- **PathIterator & operator--** ()
- **operator bool** () const
- std::string **operator\*** () const
- std::string **Rest** () const

### 6.186.1 Detailed Description

Class to iterate through elements of path.

### 6.186.2 Constructor & Destructor Documentation

#### 6.186.2.1 Arc::PathIterator::PathIterator ( const std::string & path, bool end = false )

Constructor accepts path and stores it internally. If end is set to false iterator is pointing at first element in path. Otherwise selected element is one before last.

### 6.186.3 Member Function Documentation

#### 6.186.3.1 Arc::PathIterator::operator bool ( ) const

Return false when iterator moved outside path elements

#### 6.186.3.2 std::string Arc::PathIterator::operator\* ( ) const

Returns part of initial path from first till and including current

#### 6.186.3.3 PathIterator& Arc::PathIterator::operator++ ( )

Advances iterator to point at next path element

#### 6.186.3.4 PathIterator& Arc::PathIterator::operator-- ( )

Moves iterator to element before current

#### 6.186.3.5 std::string Arc::PathIterator::Rest ( ) const

Returns part of initial path from one after current till end

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

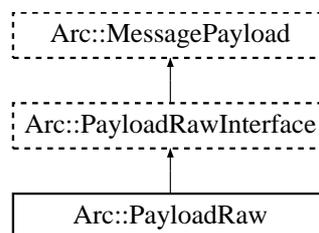
- URL.h

## 6.187 Arc::PayloadRaw Class Reference

Raw byte multi-buffer.

```
#include <PayloadRaw.h>
```

Inheritance diagram for Arc::PayloadRaw:



### Public Member Functions

- **PayloadRaw** (void)
- virtual **~PayloadRaw** (void)
- virtual char **operator[]** (Size\_t pos) const
- virtual char \* **Content** (Size\_t pos=-1)

- virtual `Size_t` **Size** (void) const
- virtual `char *` **Insert** (`Size_t` pos=0, `Size_t` size=0)
- virtual `char *` **Insert** (const `char *`s, `Size_t` pos=0, `Size_t` size=-1)
- virtual `char *` **Buffer** (unsigned int num=0)
- virtual `Size_t` **BufferSize** (unsigned int num=0) const
- virtual `Size_t` **BufferPos** (unsigned int num=0) const
- virtual bool **Truncate** (`Size_t` size)

### 6.187.1 Detailed Description

Raw byte multi-buffer. This is implementation of **PayloadRawInterface** (p. 232). Buffers are memory blocks logically placed one after another.

### 6.187.2 Constructor & Destructor Documentation

#### 6.187.2.1 `Arc::PayloadRaw::PayloadRaw ( void ) [inline]`

List of handled buffers. Constructor. Created object contains no buffers.

#### 6.187.2.2 `virtual Arc::PayloadRaw::~~PayloadRaw ( void ) [virtual]`

Destructor. Frees allocated buffers.

### 6.187.3 Member Function Documentation

#### 6.187.3.1 `virtual char* Arc::PayloadRaw::Buffer ( unsigned int num = 0 ) [virtual]`

Returns pointer to num'th buffer

Implements **Arc::PayloadRawInterface** (p. 233).

#### 6.187.3.2 `virtual Size_t Arc::PayloadRaw::BufferPos ( unsigned int num = 0 ) const [virtual]`

Returns position of num'th buffer

Implements **Arc::PayloadRawInterface** (p. 233).

#### 6.187.3.3 `virtual Size_t Arc::PayloadRaw::BufferSize ( unsigned int num = 0 ) const [virtual]`

Returns length of num'th buffer

Implements **Arc::PayloadRawInterface** (p. 233).

#### 6.187.3.4 `virtual char* Arc::PayloadRaw::Content ( Size_t pos = -1 ) [virtual]`

Get pointer to buffer content at global position 'pos'. By default to beginning of main buffer whatever that means.

Implements **Arc::PayloadRawInterface** (p. 233).

#### 6.187.3.5 `virtual char* Arc::PayloadRaw::Insert ( Size_t pos = 0, Size_t size = 0 ) [virtual]`

Create new buffer at global position 'pos' of size 'size'.

Implements **Arc::PayloadRawInterface** (p. 233).

#### 6.187.3.6 `virtual char* Arc::PayloadRaw::Insert ( const char * s, Size_t pos = 0, Size_t size = -1 ) [virtual]`

Create new buffer at global position 'pos' of size 'size'. Created buffer is filled with content of memory at 's'. If 'size' is negative content at 's' is expected to be null-terminated.

Implements **Arc::PayloadRawInterface** (p. 233).

#### 6.187.3.7 `virtual char Arc::PayloadRaw::operator[] ( Size_t pos ) const [virtual]`

Returns content of byte at specified position. Specified position 'pos' is treated as global one and goes through all buffers placed one after another.

Implements **Arc::PayloadRawInterface** (p. 234).

#### 6.187.3.8 `virtual Size_t Arc::PayloadRaw::Size ( void ) const [virtual]`

Returns logical size of whole structure.

Implements **Arc::PayloadRawInterface** (p. 234).

#### 6.187.3.9 `virtual bool Arc::PayloadRaw::Truncate ( Size_t size ) [virtual]`

Change size of stored information. If size exceeds end of allocated buffer, buffers are not re-allocated, only logical size is extended. Buffers with location behind new size are deallocated.

Implements **Arc::PayloadRawInterface** (p. 234).

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

- PayloadRaw.h

## 6.188 Arc::PayloadRawBuf Struct Reference

### Data Fields

- `int size`
- `int length`
- `bool allocated`

## 6.188.1 Field Documentation

### 6.188.1.1 `bool Arc::PayloadRawBuf::allocated`

size of used memory - size of buffer

### 6.188.1.2 `int Arc::PayloadRawBuf::length`

size of allocated memory

### 6.188.1.3 `int Arc::PayloadRawBuf::size`

pointer to buffer in memory

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

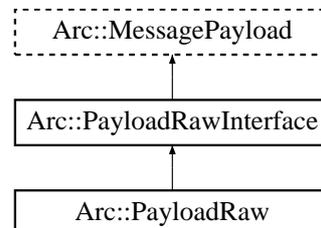
- PayloadRaw.h

## 6.189 `Arc::PayloadRawInterface` Class Reference

Random Access Payload for `Message` (p. 210) objects.

```
#include <PayloadRaw.h>
```

Inheritance diagram for `Arc::PayloadRawInterface`:



### Public Member Functions

- virtual char **operator[]** (Size\_t pos) const =0
- virtual char \* **Content** (Size\_t pos=-1)=0
- virtual Size\_t **Size** (void) const =0
- virtual char \* **Insert** (Size\_t pos=0, Size\_t size=0)=0
- virtual char \* **Insert** (const char \*s, Size\_t pos=0, Size\_t size=-1)=0
- virtual char \* **Buffer** (unsigned int num)=0
- virtual Size\_t **BufferSize** (unsigned int num) const =0
- virtual Size\_t **BufferPos** (unsigned int num) const =0
- virtual bool **Truncate** (Size\_t size)=0

## 6.189.1 Detailed Description

Random Access Payload for **Message** (p. 210) objects. This class is a virtual interface for managing **Message** (p. 210) payload with arbitrarily accessible content. Inheriting classes are supposed to implement memory-resident or memory-mapped content made of optionally multiple chunks/buffers. Every buffer has own size and offset. This class is purely virtual.

## 6.189.2 Member Function Documentation

### 6.189.2.1 `virtual char* Arc::PayloadRawInterface::Buffer ( unsigned int num ) [pure virtual]`

Returns pointer to *num*'th buffer

Implemented in **Arc::PayloadRaw** (p. 230).

### 6.189.2.2 `virtual Size_t Arc::PayloadRawInterface::BufferPos ( unsigned int num ) const [pure virtual]`

Returns position of *num*'th buffer

Implemented in **Arc::PayloadRaw** (p. 230).

### 6.189.2.3 `virtual Size_t Arc::PayloadRawInterface::BufferSize ( unsigned int num ) const [pure virtual]`

Returns length of *num*'th buffer

Implemented in **Arc::PayloadRaw** (p. 230).

### 6.189.2.4 `virtual char* Arc::PayloadRawInterface::Content ( Size_t pos = -1 ) [pure virtual]`

Get pointer to buffer content at global position '*pos*'. By default to beginning of main buffer whatever that means.

Implemented in **Arc::PayloadRaw** (p. 230).

### 6.189.2.5 `virtual char* Arc::PayloadRawInterface::Insert ( Size_t pos = 0, Size_t size = 0 ) [pure virtual]`

Create new buffer at global position '*pos*' of size '*size*'.

Implemented in **Arc::PayloadRaw** (p. 231).

### 6.189.2.6 `virtual char* Arc::PayloadRawInterface::Insert ( const char * s, Size_t pos = 0, Size_t size = -1 ) [pure virtual]`

Create new buffer at global position '*pos*' of size '*size*'. Created buffer is filled with content of memory at '*s*'. If '*size*' is negative content at '*s*' is expected to be null-terminated.

Implemented in **Arc::PayloadRaw** (p. 231).

### 6.189.2.7 `virtual char Arc::PayloadRawInterface::operator[] ( Size_t pos ) const [pure virtual]`

Returns content of byte at specified position. Specified position 'pos' is treated as global one and goes through all buffers placed one after another.

Implemented in `Arc::PayloadRaw` (p. 231).

### 6.189.2.8 `virtual Size_t Arc::PayloadRawInterface::Size ( void ) const [pure virtual]`

Returns logical size of whole structure.

Implemented in `Arc::PayloadRaw` (p. 231).

### 6.189.2.9 `virtual bool Arc::PayloadRawInterface::Truncate ( Size_t size ) [pure virtual]`

Change size of stored information. If size exceeds end of allocated buffer, buffers are not re-allocated, only logical size is extended. Buffers with location behind new size are deallocated.

Implemented in `Arc::PayloadRaw` (p. 231).

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

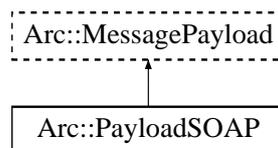
- PayloadRaw.h

## 6.190 Arc::PayloadSOAP Class Reference

Payload of `Message` (p. 210) with SOAP content.

```
#include <PayloadSOAP.h>
```

Inheritance diagram for `Arc::PayloadSOAP`:



### Public Member Functions

- `PayloadSOAP` (const NS &ns, bool fault=false)
- `PayloadSOAP` (const SOAPEnvelope &soap)
- `PayloadSOAP` (const `MessagePayload` &source)

#### 6.190.1 Detailed Description

Payload of `Message` (p. 210) with SOAP content. This class combines `MessagePayload` (p. 218) with `SOAPEnvelope` to make it possible to pass SOAP messages through `MCC` (p. 202) chain.

## 6.190.2 Constructor & Destructor Documentation

### 6.190.2.1 Arc::PayloadSOAP::PayloadSOAP ( const NS & ns, bool fault = false )

Constructor - creates new **Message** (p. 210) payload

### 6.190.2.2 Arc::PayloadSOAP::PayloadSOAP ( const SOAPEnvelope & soap )

Constructor - creates **Message** (p. 210) payload from SOAP document. Provided SOAP document is copied to new object.

### 6.190.2.3 Arc::PayloadSOAP::PayloadSOAP ( const MessagePayload & source )

Constructor - creates SOAP message from payload. **PayloadRawInterface** (p. 232) and derived classes are supported.

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

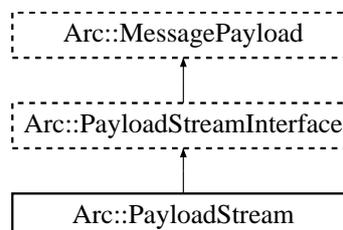
- PayloadSOAP.h

## 6.191 Arc::PayloadStream Class Reference

POSIX handle as Payload.

```
#include <PayloadStream.h>
```

Inheritance diagram for Arc::PayloadStream:



### Public Member Functions

- **PayloadStream** (int h=-1)
- virtual **~PayloadStream** (void)
- virtual bool **Get** (char \*buf, int &size)
- virtual bool **Get** (std::string &buf)
- virtual std::string **Get** (void)
- virtual bool **Put** (const char \*buf, Size\_t size)
- virtual bool **Put** (const std::string &buf)
- virtual bool **Put** (const char \*buf)
- virtual **operator bool** (void)
- virtual bool **operator!** (void)
- virtual int **Timeout** (void) const

- virtual void **Timeout** (int to)
- virtual Size\_t **Pos** (void) const
- virtual Size\_t **Size** (void) const
- virtual Size\_t **Limit** (void) const

## Protected Attributes

- int **handle\_**
- bool **seekable\_**

### 6.191.1 Detailed Description

POSIX handle as Payload. This is an implemetation of **PayloadStreamInterface** (p.238) for generic POSIX handle.

### 6.191.2 Constructor & Destructor Documentation

#### 6.191.2.1 Arc::PayloadStream::PayloadStream ( int *h* = -1 )

true if lseek operation is applicable to open handle Constructor. Attaches to already open handle. Handle is not managed by this class and must be closed by external code.

#### 6.191.2.2 virtual Arc::PayloadStream::~~PayloadStream ( void ) [inline, virtual]

Destructor.

### 6.191.3 Member Function Documentation

#### 6.191.3.1 virtual bool Arc::PayloadStream::Get ( char \* *buf*, int & *size* ) [virtual]

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implements **Arc::PayloadStreamInterface** (p.239).

#### 6.191.3.2 virtual bool Arc::PayloadStream::Get ( std::string & *buf* ) [virtual]

Read as many as possible (sane amount) of bytes into buf.

Implements **Arc::PayloadStreamInterface** (p.239).

#### 6.191.3.3 virtual std::string Arc::PayloadStream::Get ( void ) [inline, virtual]

Read as many as possible (sane amount) of bytes.

Implements **Arc::PayloadStreamInterface** (p.239).

References Get().

Referenced by Get().

**6.191.3.4 virtual Size\_t Arc::PayloadStream::Limit ( void ) const [inline, virtual]**

Returns position at which stream reading will stop if supported. That may be not same as **Size()** (p. 238) if instance is meant to provide access to only part of underlying object.

Implements **Arc::PayloadStreamInterface** (p. 239).

**6.191.3.5 virtual Arc::PayloadStream::operator bool ( void ) [inline, virtual]**

Returns true if stream is valid.

Implements **Arc::PayloadStreamInterface** (p. 240).

References `handle_`.

**6.191.3.6 virtual bool Arc::PayloadStream::operator! ( void ) [inline, virtual]**

Returns true if stream is invalid.

Implements **Arc::PayloadStreamInterface** (p. 240).

References `handle_`.

**6.191.3.7 virtual Size\_t Arc::PayloadStream::Pos ( void ) const [inline, virtual]**

Returns current position in stream if supported.

Implements **Arc::PayloadStreamInterface** (p. 240).

**6.191.3.8 virtual bool Arc::PayloadStream::Put ( const char \* buf, Size\_t size ) [virtual]**

Push 'size' bytes from 'buf' into stream. Returns true on success.

Implements **Arc::PayloadStreamInterface** (p. 240).

**6.191.3.9 virtual bool Arc::PayloadStream::Put ( const char \* buf ) [inline, virtual]**

Push null terminated information from 'buf' into stream. Returns true on success.

Implements **Arc::PayloadStreamInterface** (p. 240).

References `Put()`.

Referenced by `Put()`.

**6.191.3.10 virtual bool Arc::PayloadStream::Put ( const std::string & buf ) [inline, virtual]**

Push information from 'buf' into stream. Returns true on success.

Implements **Arc::PayloadStreamInterface** (p. 240).

References `Put()`.

Referenced by `Put()`.

**6.191.3.11** `virtual Size_t Arc::PayloadStream::Size ( void ) const [inline, virtual]`

Returns size of underlying object if supported.

Implements `Arc::PayloadStreamInterface` (p. 240).

**6.191.3.12** `virtual int Arc::PayloadStream::Timeout ( void ) const [inline, virtual]`

**Query** (p. 259) current timeout for `Get()` (p. 236) and `Put()` (p. 237) operations.

Implements `Arc::PayloadStreamInterface` (p. 240).

**6.191.3.13** `virtual void Arc::PayloadStream::Timeout ( int to ) [inline, virtual]`

Set current timeout for `Get()` (p. 236) and `Put()` (p. 237) operations.

Implements `Arc::PayloadStreamInterface` (p. 241).

**6.191.4 Field Documentation****6.191.4.1** `int Arc::PayloadStream::handle_ [protected]`

Timeout for read/write operations

Referenced by operator `bool()`, and operator `!()`.

**6.191.4.2** `bool Arc::PayloadStream::seekable_ [protected]`

Handle for operations

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

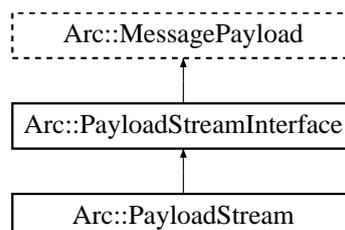
- `PayloadStream.h`

**6.192 Arc::PayloadStreamInterface Class Reference**

Stream-like Payload for `Message` (p. 210) object.

```
#include <PayloadStream.h>
```

Inheritance diagram for `Arc::PayloadStreamInterface`:



## Public Member Functions

- virtual bool **Get** (char \*buf, int &size)=0
- virtual bool **Get** (std::string &buf)=0
- virtual std::string **Get** (void)=0
- virtual bool **Put** (const char \*buf, Size\_t size)=0
- virtual bool **Put** (const std::string &buf)=0
- virtual bool **Put** (const char \*buf)=0
- virtual **operator bool** (void)=0
- virtual bool **operator!** (void)=0
- virtual int **Timeout** (void) const =0
- virtual void **Timeout** (int to)=0
- virtual Size\_t **Pos** (void) const =0
- virtual Size\_t **Size** (void) const =0
- virtual Size\_t **Limit** (void) const =0

### 6.192.1 Detailed Description

Stream-like Payload for **Message** (p. 210) object. This class is a virtual interface for managing stream-like source and destination. It's supposed to be passed through **MCC** (p. 202) chain as payload of **Message** (p. 210). It must be treated by MCCs and Services as dynamic payload. This class is purely virtual.

### 6.192.2 Member Function Documentation

#### 6.192.2.1 virtual bool Arc::PayloadStreamInterface::Get ( char \* buf, int & size ) [pure virtual]

Extracts information from stream up to 'size' bytes. 'size' contains number of read bytes on exit. Returns true in case of success.

Implemented in **Arc::PayloadStream** (p. 236).

#### 6.192.2.2 virtual bool Arc::PayloadStreamInterface::Get ( std::string & buf ) [pure virtual]

Read as many as possible (sane amount) of bytes into buf.

Implemented in **Arc::PayloadStream** (p. 236).

#### 6.192.2.3 virtual std::string Arc::PayloadStreamInterface::Get ( void ) [pure virtual]

Read as many as possible (sane amount) of bytes.

Implemented in **Arc::PayloadStream** (p. 236).

#### 6.192.2.4 virtual Size\_t Arc::PayloadStreamInterface::Limit ( void ) const [pure virtual]

Returns position at which stream reading will stop if supported. That may be not same as **Size()** (p. 240) if instance is meant to provide access to only part of underlying object.

Implemented in **Arc::PayloadStream** (p. 237).

**6.192.2.5 virtual Arc::PayloadStreamInterface::operator bool ( void ) [pure virtual]**

Returns true if stream is valid.

Implemented in **Arc::PayloadStream** (p. 237).

**6.192.2.6 virtual bool Arc::PayloadStreamInterface::operator! ( void ) [pure virtual]**

Returns true if stream is invalid.

Implemented in **Arc::PayloadStream** (p. 237).

**6.192.2.7 virtual Size\_t Arc::PayloadStreamInterface::Pos ( void ) const [pure virtual]**

Returns current position in stream if supported.

Implemented in **Arc::PayloadStream** (p. 237).

**6.192.2.8 virtual bool Arc::PayloadStreamInterface::Put ( const char \* buf, Size\_t size ) [pure virtual]**

Push 'size' bytes from 'buf' into stream. Returns true on success.

Implemented in **Arc::PayloadStream** (p. 237).

**6.192.2.9 virtual bool Arc::PayloadStreamInterface::Put ( const char \* buf ) [pure virtual]**

Push null terminated information from 'buf' into stream. Returns true on success.

Implemented in **Arc::PayloadStream** (p. 237).

**6.192.2.10 virtual bool Arc::PayloadStreamInterface::Put ( const std::string & buf ) [pure virtual]**

Push information from 'buf' into stream. Returns true on success.

Implemented in **Arc::PayloadStream** (p. 237).

**6.192.2.11 virtual Size\_t Arc::PayloadStreamInterface::Size ( void ) const [pure virtual]**

Returns size of underlying object if supported.

Implemented in **Arc::PayloadStream** (p. 238).

**6.192.2.12 virtual int Arc::PayloadStreamInterface::Timeout ( void ) const [pure virtual]**

**Query** (p. 259) current timeout for **Get()** (p. 239) and **Put()** (p. 240) operations.

Implemented in **Arc::PayloadStream** (p. 238).

**6.192.2.13 virtual void Arc::PayloadStreamInterface::Timeout ( int to ) [pure virtual]**

Set current timeout for **Get()** (p. 239) and **Put()** (p. 240) operations.

Implemented in **Arc::PayloadStream** (p. 238).

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

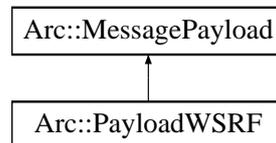
- PayloadStream.h

**6.193 Arc::PayloadWSRF Class Reference**

This class combines **MessagePayload** (p. 218) with **WSRF** (p. 369).

```
#include <PayloadWSRF.h>
```

Inheritance diagram for Arc::PayloadWSRF:

**Public Member Functions**

- **PayloadWSRF** (const SOAPEnvelope &soap)
- **PayloadWSRF** (WSRF &wsrp)
- **PayloadWSRF** (const **MessagePayload** &source)

**6.193.1 Detailed Description**

This class combines **MessagePayload** (p. 218) with **WSRF** (p. 369). It's intention is to make it possible to pass **WSRF** (p. 369) messages through **MCC** (p. 202) chain as one more Payload type.

**6.193.2 Constructor & Destructor Documentation****6.193.2.1 Arc::PayloadWSRF::PayloadWSRF ( const SOAPEnvelope & soap )**

Constructor - creates **Message** (p. 210) payload from SOAP message. Returns invalid **WSRF** (p. 369) if SOAP does not represent WS-ResourceProperties

**6.193.2.2 Arc::PayloadWSRF::PayloadWSRF ( WSRF & wsrp )**

Constructor - creates **Message** (p. 210) payload with acquired **WSRF** (p. 369) message. **WSRF** (p. 369) message will be destroyed by destructor of this object.

### 6.193.2.3 Arc::PayloadWSRF::PayloadWSRF ( const MessagePayload & source )

Constructor - creates **WSRF** (p. 369) message from payload. All classes derived from SOAPEnvelope are supported.

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

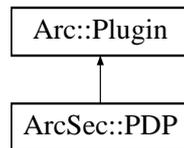
- PayloadWSRF.h

## 6.194 ArcSec::PDP Class Reference

Base class for **Policy** (p. 254) Decision Point plugins.

```
#include <PDP.h>
```

Inheritance diagram for ArcSec::PDP:



### 6.194.1 Detailed Description

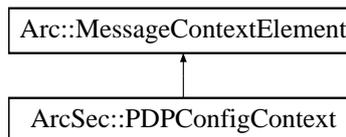
Base class for **Policy** (p. 254) Decision Point plugins. This virtual class defines method isPermitted() which processes security related information/attributes in Message and makes security decision - permit (true) or deny (false). Configuration of **PDP** (p. 242) is consumed during creation of instance through XML subtree fed to constructor.

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

- PDP.h

## 6.195 ArcSec::PDPCfgContext Class Reference

Inheritance diagram for ArcSec::PDPCfgContext:

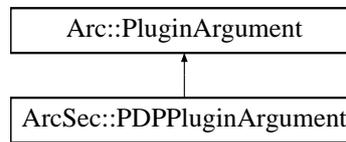


The documentation for this class was generated from the following file:

- PDP.h

## 6.196 ArcSec::PDPPluginArgument Class Reference

Inheritance diagram for ArcSec::PDPPluginArgument:



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

- PDP.h

## 6.197 Arc::Period Class Reference

### Public Member Functions

- **Period** ()
- **Period** (time\_t)
- **Period** (time\_t seconds, uint32\_t nanoseconds)
- **Period** (const std::string &, PeriodBase base=PeriodSeconds)
- **Period** & **operator=** (time\_t)
- **Period** & **operator=** (const **Period** &)
- void **SetPeriod** (time\_t)
- time\_t **GetPeriod** () const
- const sigc::slot< const char \* > \* **istr** () const
- **operator std::string** () const
- bool **operator**< (const **Period** &) const
- bool **operator**> (const **Period** &) const
- bool **operator**<= (const **Period** &) const
- bool **operator**>= (const **Period** &) const
- bool **operator**== (const **Period** &) const
- bool **operator**!= (const **Period** &) const

### 6.197.1 Constructor & Destructor Documentation

#### 6.197.1.1 Arc::Period::Period ( )

Default constructor. The period is set to 0 length.

#### 6.197.1.2 Arc::Period::Period ( time\_t )

Constructor that takes a time\_t variable and stores it.

#### 6.197.1.3 Arc::Period::Period ( time\_t seconds, uint32\_t nanoseconds )

Constructor that takes seconds and nanoseconds and stores them.

**6.197.1.4** `Arc::Period::Period ( const std::string & , PeriodBase base = PeriodSeconds )`

Constructor that tries to convert a string.

**6.197.2 Member Function Documentation****6.197.2.1** `time_t Arc::Period::GetPeriod ( ) const`

gets the period

**6.197.2.2** `const sigc::slot<const char*>* Arc::Period::istr ( ) const`

For use with `IString` (p. 184)

**6.197.2.3** `Arc::Period::operator std::string ( ) const`

Returns a string representation of the period.

**6.197.2.4** `bool Arc::Period::operator!= ( const Period & ) const`

Comparing two `Period` (p. 243) objects.

**6.197.2.5** `bool Arc::Period::operator< ( const Period & ) const`

Comparing two `Period` (p. 243) objects.

**6.197.2.6** `bool Arc::Period::operator<= ( const Period & ) const`

Comparing two `Period` (p. 243) objects.

**6.197.2.7** `Period& Arc::Period::operator= ( time_t )`

Assignment operator from a `time_t`.

**6.197.2.8** `Period& Arc::Period::operator= ( const Period & )`

Assignment operator from a `Period` (p. 243).

**6.197.2.9** `bool Arc::Period::operator== ( const Period & ) const`

Comparing two `Period` (p. 243) objects.

**6.197.2.10** `bool Arc::Period::operator> ( const Period & ) const`

Comparing two `Period` (p. 243) objects.

**6.197.2.11** `bool ArcSec::Period::operator>= ( const Period & ) const`

Comparing two **Period** (p. 243) objects.

**6.197.2.12** `void ArcSec::Period::SetPeriod ( time_t )`

sets the period

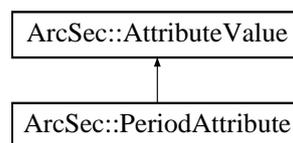
The documentation for this class was generated from the following file:

- DateTime.h

**6.198 ArcSec::PeriodAttribute Class Reference**

```
#include <DateTimeAttribute.h>
```

Inheritance diagram for ArcSec::PeriodAttribute:

**Public Member Functions**

- virtual `bool equal ( AttributeValue *other, bool check_id=true)`
- virtual `std::string encode ()`
- virtual `std::string getType ()`
- virtual `std::string getId ()`

**6.198.1 Detailed Description**

Format: `datetime"/"duration datetime"/"datetime duration"/"datetime`

**6.198.2 Member Function Documentation****6.198.2.1** `virtual std::string ArcSec::PeriodAttribute::encode ( ) [virtual]`

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 57).

**6.198.2.2** `virtual bool ArcSec::PeriodAttribute::equal ( AttributeValue * value, bool check_id = true ) [virtual]`

Evaluate whether "this" equals to the parameter value

Implements **ArcSec::AttributeValue** (p. 58).

### 6.198.2.3 virtual std::string ArcSec::PeriodAttribute::getId ( ) [inline, virtual]

Get the AttributeId of the <Attribute>

Implements ArcSec::AttributeValue (p. 58).

### 6.198.2.4 virtual std::string ArcSec::PeriodAttribute::getType ( ) [inline, virtual]

Get the DataType of the <Attribute>

Implements ArcSec::AttributeValue (p. 58).

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

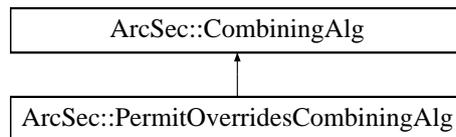
- DateTimeAttribute.h

## 6.199 ArcSec::PermitOverridesCombiningAlg Class Reference

Implement the "Permit-Overrides" algorithm.

```
#include <PermitOverridesAlg.h>
```

Inheritance diagram for ArcSec::PermitOverridesCombiningAlg:



### Public Member Functions

- virtual Result **combine** (EvaluationCtx \*ctx, std::list< Policy \* > policies)
- virtual const std::string & **getalgId** (void) const

### 6.199.1 Detailed Description

Implement the "Permit-Overrides" algorithm. Permit-Overrides, scans the policy set which is given as the parameters of "combine" method, if gets "permit" result from any policy, then stops scanning and gives "permit" as result, otherwise gives "deny".

### 6.199.2 Member Function Documentation

#### 6.199.2.1 virtual Result ArcSec::PermitOverridesCombiningAlg::combine ( EvaluationCtx \* ctx, std::list< Policy \* > policies ) [virtual]

If there is one policy which return positive evaluation result, then omit the other policies and return DECISION\_PERMIT

#### Parameters

- ctx* This object contains request information which will be used to evaluated against policy.

*policiies* This is a container which contains policy objects.

### Returns

The combined result according to the algorithm.

Implements `ArcSec::CombiningAlg` (p. 76).

**6.199.2.2** `virtual const std::string& ArcSec::PermitOverridesCombiningAlg::getalgId ( void )`  
`const [inline, virtual]`

Get the identifier

Implements `ArcSec::CombiningAlg` (p. 76).

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

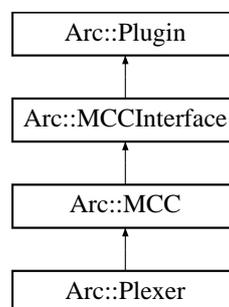
- PermitOverridesAlg.h

## 6.200 Arc::Plexer Class Reference

The **Plexer** (p. 247) class, used for routing messages to services.

```
#include <Plexer.h>
```

Inheritance diagram for Arc::Plexer:



### Public Member Functions

- **Plexer** (`Config *cfg`)
- virtual `~Plexer` ()
- virtual void **Next** (`MCCInterface *next`, `const std::string &label`)
- virtual `MCC_Status process` (`Message &request`, `Message &response`)

### Static Public Attributes

- static **Logger** `logger`

### 6.200.1 Detailed Description

The **Plexer** (p. 247) class, used for routing messages to services. This is the **Plexer** (p. 247) class. Its purpose is to route incoming messages to appropriate Services and **MCC** (p. 202) chains.

### 6.200.2 Constructor & Destructor Documentation

#### 6.200.2.1 `Arc::Plexer::Plexer ( Config * cfg )`

The constructor.

This is the constructor. Since all member variables are instances of "well-behaving" STL classes, nothing needs to be done.

#### 6.200.2.2 `virtual Arc::Plexer::~~Plexer ( ) [virtual]`

The destructor.

This is the destructor. Since all member variables are instances of "well-behaving" STL classes, nothing needs to be done.

### 6.200.3 Member Function Documentation

#### 6.200.3.1 `virtual void Arc::Plexer::Next ( MCCInterface * next, const std::string & label ) [virtual]`

Add reference to next **MCC** (p. 202) in chain.

This method is called by **Loader** (p. 191) for every potentially labeled link to next component which implements **MCCInterface** (p. 207). If next is set NULL corresponding link is removed.

Reimplemented from **Arc::MCC** (p. 203).

#### 6.200.3.2 `virtual MCC_Status Arc::Plexer::process ( Message & request, Message & response ) [virtual]`

Route request messages to appropriate services.

Routes the request message to the appropriate service. Routing is based on the path part of value of the **ENDPOINT** attribute. Routed message is assigned following attributes: **PLEXER:PATTERN** - matched pattern, **PLEXER:EXTENSION** - last unmatched part of **ENDPOINT** path.

Reimplemented from **Arc::MCC** (p. 203).

### 6.200.4 Field Documentation

#### 6.200.4.1 `Logger Arc::Plexer::logger [static]`

A logger for MCCs.

A logger intended to be the parent of loggers in the different MCCs.

Reimplemented from **Arc::MCC** (p. 204).

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

- Plexer.h

## 6.201 Arc::PlexerEntry Class Reference

A pair of label (regex) and pointer to MCC (p. 202).

```
#include <Plexer.h>
```

### 6.201.1 Detailed Description

A pair of label (regex) and pointer to MCC (p. 202). A helper class that stores a label (regex) and a pointer to a service.

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

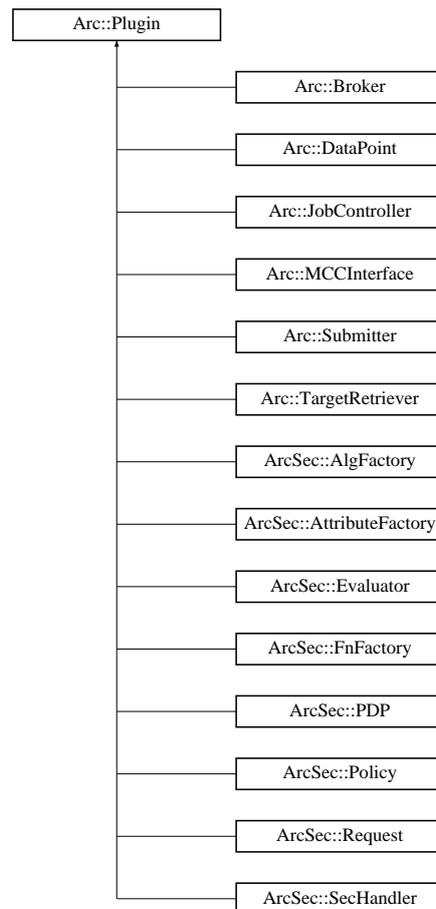
- Plexer.h

## 6.202 Arc::Plugin Class Reference

Base class for loadable ARC components.

```
#include <Plugin.h>
```

Inheritance diagram for Arc::Plugin:



### 6.202.1 Detailed Description

Base class for loadable ARC components. All classes representing loadable ARC components must be either descendants of this class or be wrapped by its offspring.

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

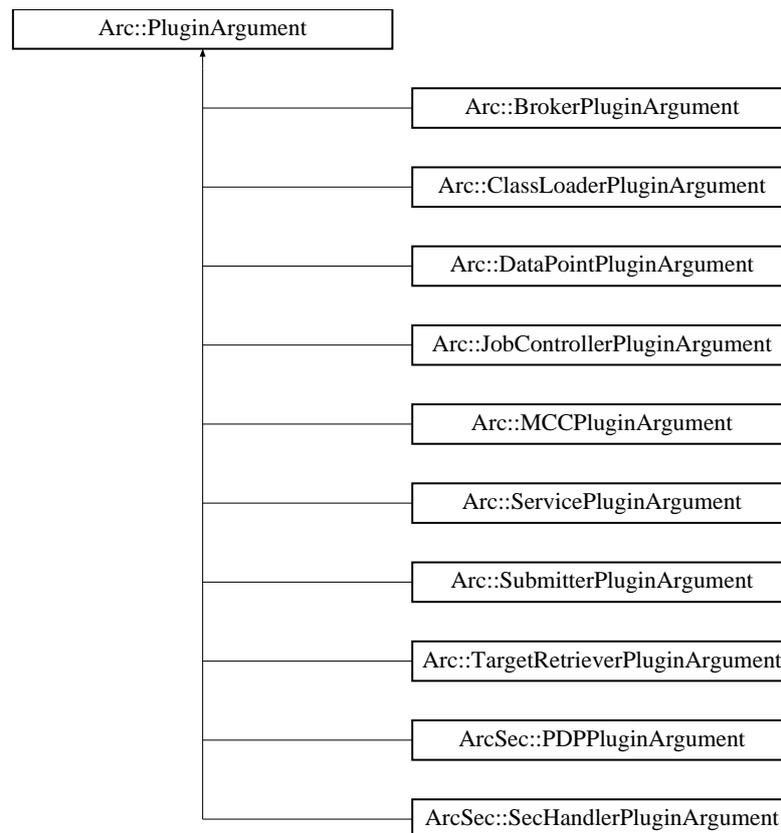
- `Plugin.h`

### 6.203 `Arc::PluginArgument` Class Reference

Base class for passing arguments to loadable ARC components.

```
#include <Plugin.h>
```

Inheritance diagram for `Arc::PluginArgument`:



## Public Member Functions

- **PluginsFactory** \* **get\_factory** (void)
- **Glib::Module** \* **get\_module** (void)

### 6.203.1 Detailed Description

Base class for passing arguments to loadable ARC components. During its creation constructor function of ARC loadable component expects instance of class inherited from this one or wrapped in it. Then dynamic type casting is used for obtaining class of expected kind.

### 6.203.2 Member Function Documentation

#### 6.203.2.1 PluginsFactory\* Arc::PluginArgument::get\_factory ( void )

Returns pointer to factory which instantiated plugin.

Because factory usually destroys/unloads plugins in its destructor it should be safe to keep this pointer inside plugin for later use. But one must always check.

#### 6.203.2.2 Glib::Module\* Arc::PluginArgument::get\_module ( void )

Returns pointer to loadable module/library which contains plugin.

Corresponding factory keeps list of modules till itself is destroyed. So it should be safe to keep that pointer. But care must be taken if module contains persistent plugins. Such modules stay in memory after factory is destroyed. So it is advisable to use obtained pointer only in constructor function of plugin.

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

- Plugin.h

## 6.204 Arc::PluginDesc Class Reference

Description of plugin.

```
#include <Plugin.h>
```

### 6.204.1 Detailed Description

Description of plugin. This class is used for reports

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

- Plugin.h

## 6.205 Arc::PluginDescriptor Struct Reference

Description of ARC lodable component.

```
#include <Plugin.h>
```

### 6.205.1 Detailed Description

Description of ARC lodable component.

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

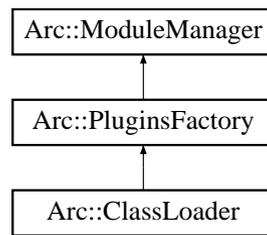
- Plugin.h

## 6.206 Arc::PluginsFactory Class Reference

Generic ARC plugins loader.

```
#include <Plugin.h>
```

Inheritance diagram for Arc::PluginsFactory:



## Public Member Functions

- **PluginsFactory** (`XMLNode cfg`)
- `void TryLoad` (`bool v=true`)
- `bool load` (`const std::string &name`)
- `bool scan` (`const std::string &name`, `ModuleDesc &desc`)
- `void report` (`std::list< ModuleDesc > &descs`)

### 6.206.1 Detailed Description

Generic ARC plugins loader. The instance of this class provides functionality of loading pluggable ARC components stored in shared libraries. For more information please check HED documentation. This class is thread-safe - its methods are protected from simultaneous use from multiple threads. Current thread protection implementation is suboptimal and will be revised in future.

### 6.206.2 Constructor & Destructor Documentation

#### 6.206.2.1 Arc::PluginsFactory::PluginsFactory ( XMLNode *cfg* )

Constructor - accepts configuration (not yet used) meant to tune loading of modules.

### 6.206.3 Member Function Documentation

#### 6.206.3.1 bool Arc::PluginsFactory::load ( const std::string & *name* )

These methods load module named lib'*name*' and check if it contains ARC plugin(s) of specified '*kind*' and '*name*'. If there are no specified plugins or module does not contain any ARC plugins it is unloaded. All loaded plugins are also registered in internal list of this instance of **PluginsFactory** (p.252) class. Returns true if any plugin was loaded.

#### 6.206.3.2 void Arc::PluginsFactory::report ( std::list< ModuleDesc > & *descs* )

Provides information about currently loaded modules and plugins.

#### 6.206.3.3 bool Arc::PluginsFactory::scan ( const std::string & *name*, ModuleDesc & *desc* )

Collect information about plugins stored in module(s) with specified names. Returns true if any of specified modules has plugins.

### 6.206.3.4 void Arc::PluginsFactory::TryLoad ( bool v = true ) [inline]

These methods load module named lib' name', locate plugin constructor functions of specified 'kind' and 'name' (if specified) and call it. Supplied argument affects way plugin instance is created in plugin-specific way. If name of plugin is not specified then all plugins of specified kind are tried with supplied argument till valid instance is created. All loaded plugins are also registered in internal list of this instance of **PluginsFactory** (p. 252) class. If search is set to false then no attempt is made to find plugins in loadable modules. Only plugins already loaded with previous calls to get\_instance() and load() are checked. Returns created instance or NULL if failed. Specifies if loadable module may be loaded while looking for analyzing its content. If set to false only \*.apd files are checked. Modules without corresponding \*.apd will be ignored. Default is true;

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

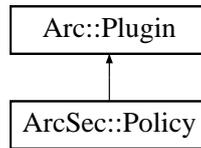
- Plugin.h

## 6.207 ArcSec::Policy Class Reference

Interface for containing and processing different types of policy.

```
#include <Policy.h>
```

Inheritance diagram for ArcSec::Policy:



### Public Member Functions

- **Policy** ()
- **Policy** (const Arc::XMLNode)
- **Policy** (const Arc::XMLNode, EvaluatorContext \*)
- virtual **operator bool** (void) const =0
- virtual MatchResult **match** (EvaluationCtx \*)=0
- virtual Result **eval** (EvaluationCtx \*)=0
- virtual void **addPolicy** (Policy \*pl)
- virtual void **setEvaluatorContext** (EvaluatorContext \*)
- virtual void **make\_policy** ()
- virtual std::string **getEffect** () const =0
- virtual EvalResult & **getEvalResult** ()=0
- virtual void **setEvalResult** (EvalResult &res)=0
- virtual const char \* **getEvalName** () const =0
- virtual const char \* **getName** () const =0

## 6.207.1 Detailed Description

Interface for containing and processing different types of policy. Basically, each policy object is a container which includes a few elements e.g., ArcPolicySet objects includes a few ArcPolicy objects; ArcPolicy object includes a few ArcRule objects. There is logical relationship between ArcRules or ArcPolicies, which is called combining algorithm. According to algorithm, evaluation results from the elements are combined, and then the combined evaluation result is returned to the up-level.

## 6.207.2 Constructor & Destructor Documentation

### 6.207.2.1 ArcSec::Policy::Policy ( const Arc::XMLNode ) [inline]

Template constructor - creates policy based on XML document.

If XML document is empty then empty policy is created. If it is not empty then it must be valid policy document - otherwise created object should be invalid.

### 6.207.2.2 ArcSec::Policy::Policy ( const Arc::XMLNode, EvaluatorContext \* ) [inline]

Template constructor - creates policy based on XML document.

If XML document is empty then empty policy is created. If it is not empty then it must be valid policy document - otherwise created object should be invalid. This constructor is based on the policy node and i the **EvaluatorContext** (p. 152) which includes the factory objects for combining algorithm and function

## 6.207.3 Member Function Documentation

### 6.207.3.1 virtual void ArcSec::Policy::addPolicy ( Policy \* pl ) [inline, virtual]

Add a policy element to into "this" object

### 6.207.3.2 virtual Result ArcSec::Policy::eval ( EvaluationCtx \* ) [pure virtual]

Evaluate policy For the <Rule> of **Arc** (p.23), only get the "Effect" from rules; For the <Policy> of **Arc** (p. 23), combine the evaluation result from <Rule>; For the <Rule> of XACML, evaluate the <Condition> node by using information from request, and use the "Effect" attribute of <Rule>; For the <Policy> of XACML, combine the evaluation result from <Rule>

### 6.207.3.3 virtual std::string ArcSec::Policy::getEffect ( ) const [pure virtual]

Get the "Effect" attribute

### 6.207.3.4 virtual const char\* ArcSec::Policy::getEvalName ( ) const [pure virtual]

Get the name of **Evaluator** (p. 150) which can evaluate this policy

### 6.207.3.5 virtual EvalResult& ArcSec::Policy::getEvalResult ( ) [pure virtual]

Get evaluation result

**6.207.3.6** `virtual const char* ArcSec::Policy::getName ( ) const [pure virtual]`

Get the name of this policy

**6.207.3.7** `virtual void ArcSec::Policy::make_policy ( ) [inline, virtual]`

Parse XMLNode, and construct the low-level Rule object

**6.207.3.8** `virtual void ArcSec::Policy::setEvalResult ( EvalResult & res ) [pure virtual]`

Set evaluation result

**6.207.3.9** `virtual void ArcSec::Policy::setEvaluatorContext ( EvaluatorContext * ) [inline, virtual]`

Set **Evaluator** (p. 150) Context for the usage in creating low-level policy object

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

- Policy.h

## 6.208 ArcSec::PolicyStore::PolicyElement Class Reference

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

- PolicyStore.h

## 6.209 ArcSec::PolicyParser Class Reference

A interface which will isolate the policy object from actual policy storage (files, urls, database).

```
#include <PolicyParser.h>
```

### Public Member Functions

- virtual **Policy** \* **parsePolicy** (const **Source** &source, std::string policyclassname, **EvaluatorContext** \*ctx)

### 6.209.1 Detailed Description

A interface which will isolate the policy object from actual policy storage (files, urls, database). Parse the policy from policy source (e.g. files, urls, database, etc.).

## 6.209.2 Member Function Documentation

### 6.209.2.1 virtual Policy\* ArcSec::PolicyParser::parsePolicy ( const Source & *source*, std::string *policyclassname*, EvaluatorContext \* *ctx* ) [virtual]

Parse policy

#### Parameters

*source* location of the policy

*policyclassname* name of the policy for ClassLoader

*ctx* EvaluatorContext (p. 152) which includes the \*\*Factory

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

- PolicyParser.h

## 6.210 ArcSec::PolicyStore Class Reference

Storage place for policy objects.

```
#include <PolicyStore.h>
```

### Data Structures

- class PolicyElement

### Public Member Functions

- PolicyStore (const std::string &alg, const std::string &policyclassname, EvaluatorContext \*ctx)

### 6.210.1 Detailed Description

Storage place for policy objects.

### 6.210.2 Constructor & Destructor Documentation

#### 6.210.2.1 ArcSec::PolicyStore::PolicyStore ( const std::string & *alg*, const std::string & *policyclassname*, EvaluatorContext \* *ctx* )

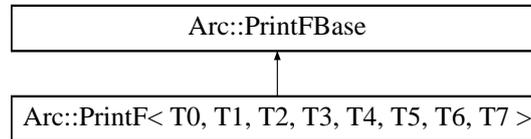
Creates policy store with specified combining algorithm (alg - not used yet), policy name (policyclassname) and context (ctx)

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

- PolicyStore.h

## 6.211 Arc::Printf< T0, T1, T2, T3, T4, T5, T6, T7 > Class Template Reference

Inheritance diagram for Arc::Printf< T0, T1, T2, T3, T4, T5, T6, T7 >:



```

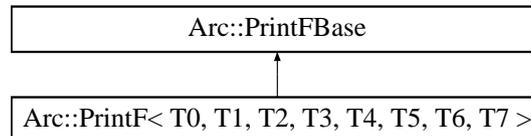
template<class T0 = int, class T1 = int, class T2 = int, class T3 = int, class T4 = int, class T5 = int,
class T6 = int, class T7 = int> class Arc::Printf< T0, T1, T2, T3, T4, T5, T6, T7 >
  
```

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

- IString.h

## 6.212 Arc::PrintfBase Class Reference

Inheritance diagram for Arc::PrintfBase:

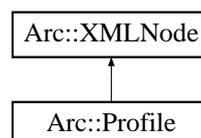


The documentation for this class was generated from the following file:

- IString.h

## 6.213 Arc::Profile Class Reference

Inheritance diagram for Arc::Profile:



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

- Profile.h

## 6.214 ArcCredential::PROXYCERTINFO\_st Struct Reference

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

- Proxycertinfo.h

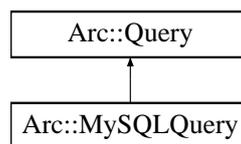
## 6.215 ArcCredential::PROXYPOLICY\_st Struct Reference

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

- Proxycertinfo.h

## 6.216 Arc::Query Class Reference

Inheritance diagram for Arc::Query:



### Public Member Functions

- **Query** ()
- **Query** (Database \*db)
- virtual ~**Query** ()
- virtual int **get\_num\_columns** ()=0
- virtual int **get\_num\_rows** ()=0
- virtual bool **execute** (const std::string &sqlstr)=0
- virtual QueryRowResult **get\_row** (int row\_number) const =0
- virtual QueryRowResult **get\_row** () const =0
- virtual std::string **get\_row\_field** (int row\_number, std::string &field\_name)=0
- virtual bool **get\_array** (std::string &sqlstr, QueryArrayResult &result, std::vector< std::string > &arguments)=0

### 6.216.1 Constructor & Destructor Documentation

#### 6.216.1.1 Arc::Query::Query ( ) [inline]

Default constructor

#### 6.216.1.2 Arc::Query::Query ( Database \* db ) [inline]

Constructor

**Parameters**

*db* The database object which will be used by **Query** (p. 259) class to get the database connection

**6.216.1.3 virtual Arc::Query::~~Query ( ) [inline, virtual]**

Deconstructor

**6.216.2 Member Function Documentation****6.216.2.1 virtual bool Arc::Query::execute ( const std::string & *sqlstr* ) [pure virtual]**

Execute the query

**Parameters**

*sqlstr* The sql sentence used to query

Implemented in **Arc::MySQLQuery** (p. 223).

**6.216.2.2 virtual bool Arc::Query::get\_array ( std::string & *sqlstr*, QueryArrayResult & *result*, std::vector< std::string > & *arguments* ) [pure virtual]**

**Query** (p. 259) the database by using some parameters into sql sentence e.g. "select table.value from table where table.name = ?"

**Parameters**

*sqlstr* The sql sentence with some parameters marked with "?".

*result* The result in an array which includes all of the value in query result.

*arguments* The argument list which should exactly correspond with the parametes in sql sentence.

Implemented in **Arc::MySQLQuery** (p. 223).

**6.216.2.3 virtual int Arc::Query::get\_num\_columns ( ) [pure virtual]**

Get the colum number in the query result

Implemented in **Arc::MySQLQuery** (p. 224).

**6.216.2.4 virtual int Arc::Query::get\_num\_rows ( ) [pure virtual]**

Get the row number in the query result

Implemented in **Arc::MySQLQuery** (p. 224).

**6.216.2.5 virtual QueryRowResult Arc::Query::get\_row ( int *row\_number* ) const [pure virtual]**

Get the value of one row in the query result

**Parameters**

*row\_number* The number of the row

**Returns**

A vector includes all the values in the row

Implemented in `Arc::MySQLQuery` (p. 224).

**6.216.2.6 virtual QueryRowResult Arc::Query::get\_row ( ) const [pure virtual]**

Get the value of one row in the query result, the row number will be automatically increased each time the method is called

Implemented in `Arc::MySQLQuery` (p. 224).

**6.216.2.7 virtual std::string Arc::Query::get\_row\_field ( int row\_number, std::string & field\_name ) [pure virtual]**

Get the value of one specific field in one specific row

**Parameters**

*row\_number* The row number inside the query result

*field\_name* The field name for the value which will be return

**Returns**

The value of the specified filed in the specified row

Implemented in `Arc::MySQLQuery` (p. 224).

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

- DBInterface.h

**6.217 Arc::Range< T > Class Template Reference**

```
template<class T> class Arc::Range< T >
```

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

- JobDescription.h

**6.218 Arc::Register\_Info\_Type Struct Reference**

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

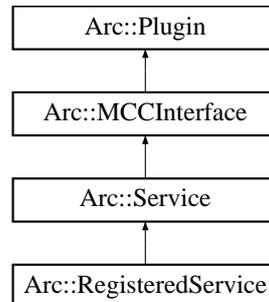
- InfoRegister.h

## 6.219 Arc::RegisteredService Class Reference

**RegisteredService** (p. 262) - extension of **Service** (p. 285) performing self-registration.

```
#include <RegisteredService.h>
```

Inheritance diagram for Arc::RegisteredService:



### Public Member Functions

- **RegisteredService** (**Config** \*)

#### 6.219.1 Detailed Description

**RegisteredService** (p. 262) - extension of **Service** (p. 285) performing self-registration.

#### 6.219.2 Constructor & Destructor Documentation

##### 6.219.2.1 Arc::RegisteredService::RegisteredService ( **Config** \* )

Example constructor - Server takes at least it's configuration subtree

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

- RegisteredService.h

## 6.220 Arc::RegularExpression Class Reference

A regular expression class.

```
#include <ArcRegex.h>
```

### Public Member Functions

- **RegularExpression** ()
- **RegularExpression** (std::string pattern)
- **RegularExpression** (const **RegularExpression** &regex)
- **~RegularExpression** ()
- const **RegularExpression** & **operator=** (const **RegularExpression** &regex)

- bool **isOk** ()
- bool **hasPattern** (std::string str)
- bool **match** (const std::string &str) const
- bool **match** (const std::string &str, std::list< std::string > &unmatched, std::list< std::string > &matched) const
- std::string **getPattern** () const

### 6.220.1 Detailed Description

A regular expression class. This class is a wrapper around the functions provided in regex.h.

### 6.220.2 Member Function Documentation

#### 6.220.2.1 bool Arc::RegularExpression::match ( const std::string & *str*, std::list< std::string > & *unmatched*, std::list< std::string > & *matched* ) const

Returns true if this regex matches the string provided.

Unmatched parts of the string are stored in 'unmatched'. Matched parts of the string are stored in 'matched'. The first entry in matched is the string that matched the regex, and the following entries are parenthesised elements of the regex

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

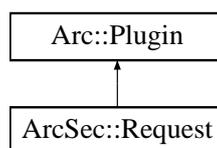
- ArcRegex.h

## 6.221 ArcSec::Request Class Reference

Base class/Interface for request, includes a container for RequestItems and some operations.

```
#include <Request.h>
```

Inheritance diagram for ArcSec::Request:



### Public Member Functions

- virtual ReqItemList **getRequestItems** () const
- virtual void **setRequestItems** (ReqItemList)
- virtual void **addRequestItem** (Attrs &, Attrs &, Attrs &, Attrs &)
- virtual void **setAttributeFactory** (AttributeFactory \*attributefactory)=0
- virtual void **make\_request** ()=0
- virtual const char \* **getEvalName** () const =0
- virtual const char \* **getName** () const =0
- **Request** ()
- **Request** (const Source &)

### 6.221.1 Detailed Description

Base class/Interface for request, includes a container for RequestItems and some operations. A **Request** (p. 263) object can has a few <subjects, actions, objects> tuples, i.e. **RequestItem** (p. 266) The **Request** (p. 263) class and any customized class which inherit from it, should be loadable, which means these classes can be dynamically loaded according to the configuration information, see the example configuration below: <Service name="pdp.service" id="pdp\_service"> <pdp:PDPCConfig> <.....> <pdp:Request (p. 263) name="arc.request" /> <.....> </pdp:PDPCConfig> </Service>

There can be different types of subclass which inherit **Request** (p. 263), such like XACMLRequest, ArcRequest, GACLRequest

### 6.221.2 Constructor & Destructor Documentation

#### 6.221.2.1 ArcSec::Request::Request ( ) [inline]

Default constructor

#### 6.221.2.2 ArcSec::Request::Request ( const Source & ) [inline]

Constructor: Parse request information from a xml stucture in memory

### 6.221.3 Member Function Documentation

#### 6.221.3.1 virtual void ArcSec::Request::addItem ( Attrs &, Attrs &, Attrs &, Attrs & ) [inline, virtual]

Add request tuple from non-XMLNode

#### 6.221.3.2 virtual const char\* ArcSec::Request::getEvalName ( ) const [pure virtual]

Get the name of corresponding evaluator

#### 6.221.3.3 virtual const char\* ArcSec::Request::getName ( ) const [pure virtual]

Get the name of this request

#### 6.221.3.4 virtual ReqItemList ArcSec::Request::getRequestItems ( ) const [inline, virtual]

Get all the **RequestItem** (p. 266) inside **RequestItem** (p. 266) container

#### 6.221.3.5 virtual void ArcSec::Request::make\_request ( ) [pure virtual]

Create the objects included in **Request** (p. 263) according to the node attached to the **Request** (p. 263) object

**6.221.3.6** virtual void ArcSec::Request::setAttributeFactory ( AttributeFactory \* *attributefactory* ) [pure virtual]

Set the attribute factory for the usage of **Request** (p. 263)

**6.221.3.7** virtual void ArcSec::Request::setRequestItems ( ReqItemList ) [inline, virtual]

Set the content of the container

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

- Request.h

## 6.222 ArcSec::RequestAttribute Class Reference

Wrapper which includes **AttributeValue** (p. 56) object which is generated according to date type of one specific node in Request.xml.

```
#include <RequestAttribute.h>
```

### Public Member Functions

- **RequestAttribute** (Arc::XMLNode &node, AttributeFactory \*attrfactory)
- **RequestAttribute & duplicate** (RequestAttribute &)

### 6.222.1 Detailed Description

Wrapper which includes **AttributeValue** (p. 56) object which is generated according to date type of one specific node in Request.xml.

### 6.222.2 Constructor & Destructor Documentation

**6.222.2.1** ArcSec::RequestAttribute::RequestAttribute ( Arc::XMLNode & *node*, AttributeFactory \* *atrfactory* )

Constructor - create attribute value object according to the "Type" in the node <Attribute attributeid="urn:arc:subject:voms-attribute" type="string">urn:mace:shibboleth:examples</Attribute>

### 6.222.3 Member Function Documentation

**6.222.3.1** RequestAttribute& ArcSec::RequestAttribute::duplicate ( RequestAttribute & )

Duplicate the parameter into "this"

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

- RequestAttribute.h

## 6.223 ArcSec::RequestItem Class Reference

Interface for request item container, <subjects, actions, objects, ctxs> tuple.

```
#include <RequestItem.h>
```

### Public Member Functions

- **RequestItem** (Arc::XMLNode &, AttributeFactory \*)

### 6.223.1 Detailed Description

Interface for request item container, <subjects, actions, objects, ctxs> tuple.

### 6.223.2 Constructor & Destructor Documentation

#### 6.223.2.1 ArcSec::RequestItem::RequestItem ( Arc::XMLNode &, AttributeFactory \* ) [inline]

Constructor

#### Parameters

- node* The XMLNode structure of the request item
- attributefactory* The **AttributeFactory** (p.52) which will be used to generate **RequestAttribute** (p.265)

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

- RequestItem.h

## 6.224 ArcSec::RequestTuple Class Reference

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

- EvaluationCtx.h

## 6.225 Arc::ResourceSlotType Class Reference

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

- JobDescription.h

## 6.226 Arc::ResourcesType Class Reference

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

- JobDescription.h

## 6.227 Arc::ResourceTargetType Class Reference

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

- JobDescription.h

## 6.228 ArcSec::Response Class Reference

Container for the evaluation results.

```
#include <Response.h>
```

### 6.228.1 Detailed Description

Container for the evaluation results.

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

- Response.h

## 6.229 ArcSec::ResponseItem Class Reference

Evaluation result concerning one **RequestTuple** (p. 266).

```
#include <Response.h>
```

### 6.229.1 Detailed Description

Evaluation result concerning one **RequestTuple** (p. 266). Include the **RequestTuple** (p. 266), related XMLNode, the set of policy objects which give positive evaluation result, and the related XMLNode

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

- Response.h

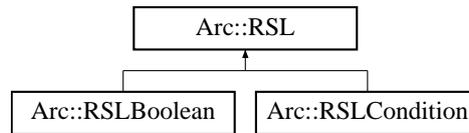
## 6.230 ArcSec::ResponseList Class Reference

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

- Response.h

## 6.231 Arc::RSL Class Reference

Inheritance diagram for Arc::RSL:

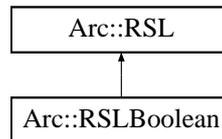


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.232 Arc::RSLBoolean Class Reference

Inheritance diagram for `Arc::RSLBoolean`:

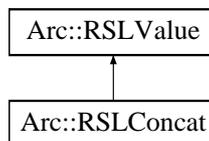


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.233 Arc::RSLConcat Class Reference

Inheritance diagram for `Arc::RSLConcat`:

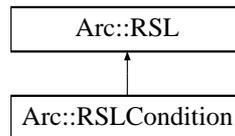


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.234 Arc::RSLCondition Class Reference

Inheritance diagram for `Arc::RSLCondition`:

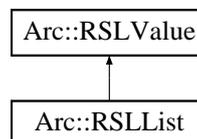


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.235 Arc::RSLList Class Reference

Inheritance diagram for Arc::RSLList:

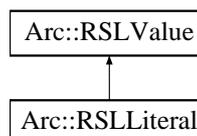


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.236 Arc::RSLLiteral Class Reference

Inheritance diagram for Arc::RSLLiteral:



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

- RSLParser.h

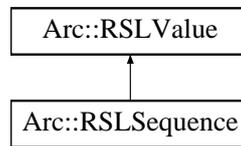
## 6.237 Arc::RSLParser Class Reference

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

- RSLParser.h

## 6.238 Arc::RSLSequence Class Reference

Inheritance diagram for Arc::RSLSequence:

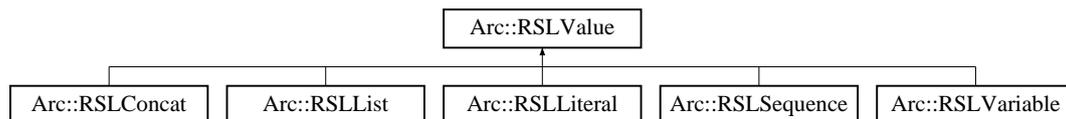


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.239 Arc::RSLValue Class Reference

Inheritance diagram for Arc::RSLValue:

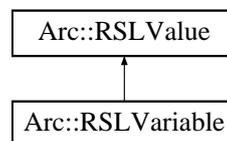


The documentation for this class was generated from the following file:

- RSLParser.h

## 6.240 Arc::RSLVariable Class Reference

Inheritance diagram for Arc::RSLVariable:



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

- RSLParser.h

## 6.241 Arc::Run Class Reference

```
#include <Run.h>
```

## Public Member Functions

- **Run** (const std::string &cmdline)
- **Run** (const std::list< std::string > &argv)
- **~Run** (void)
- **operator bool** (void)
- **bool operator!** (void)
- **bool Start** (void)
- **bool Wait** (int timeout)
- **bool Wait** (void)
- **int Result** (void)
- **bool Running** (void)
- **int ReadStdout** (int timeout, char \*buf, int size)
- **int ReadStderr** (int timeout, char \*buf, int size)
- **int WriteStdin** (int timeout, const char \*buf, int size)
- **void AssignStdout** (std::string &str)
- **void AssignStderr** (std::string &str)
- **void AssignStdin** (std::string &str)
- **void KeepStdout** (bool keep=true)
- **void KeepStderr** (bool keep=true)
- **void KeepStdin** (bool keep=true)
- **void CloseStdout** (void)
- **void CloseStderr** (void)
- **void CloseStdin** (void)
- **void AssignWorkingDirectory** (std::string &wd)
- **void Kill** (int timeout)
- **void Abandon** (void)

## Static Public Member Functions

- static void **AfterFork** (void)

### 6.241.1 Detailed Description

This class runs external executable. It is possible to read/write it's standard handles or to redirect them to std::string elements.

### 6.241.2 Constructor & Destructor Documentation

#### 6.241.2.1 Arc::Run::Run ( const std::string & *cmdline* )

Constructor preapres object to run cmdline

#### 6.241.2.2 Arc::Run::Run ( const std::list< std::string > & *argv* )

Constructor preapres object to run executable and arguments specified in argv

### 6.241.2.3 Arc::Run::~~Run ( void )

Destructor kills running executable and releases associated resources

## 6.241.3 Member Function Documentation

### 6.241.3.1 void Arc::Run::Abandon ( void )

Detach this object from running process. After calling this method instance is not associated with external process anymore. As result destructor will not kill process.

### 6.241.3.2 static void Arc::Run::AfterFork ( void ) [static]

Call this method after fork() in child cporocess. It will reinitialize internal structures for new environment. Do not call it in any other case than defined.

### 6.241.3.3 void Arc::Run::AssignStderr ( std::string & str )

Associate stderr handle of executable with string. This method must be called before **Start()** (p.274). str object must be valid as long as this object exists.

### 6.241.3.4 void Arc::Run::AssignStdin ( std::string & str )

Associate stdin handle of executable with string. This method must be called before **Start()** (p.274). str object must be valid as long as this object exists.

### 6.241.3.5 void Arc::Run::AssignStdout ( std::string & str )

Associate stdout handle of executable with string. This method must be called before **Start()** (p.274). str object must be valid as long as this object exists.

### 6.241.3.6 void Arc::Run::AssignWorkingDirectory ( std::string & wd ) [inline]

Assign working direcotry of the running process

### 6.241.3.7 void Arc::Run::CloseStderr ( void )

Closes pipe associated with stderr handle

### 6.241.3.8 void Arc::Run::CloseStdin ( void )

Closes pipe associated with stdin handle

### 6.241.3.9 void Arc::Run::CloseStdout ( void )

Closes pipe associated with stdout handle

**6.241.3.10 void Arc::Run::KeepStderr ( bool *keep* = true )**

Keep stderr same as parent's if keep = true

**6.241.3.11 void Arc::Run::KeepStdin ( bool *keep* = true )**

Keep stdin same as parent's if keep = true

**6.241.3.12 void Arc::Run::KeepStdout ( bool *keep* = true )**

Keep stdout same as parent's if keep = true

**6.241.3.13 void Arc::Run::Kill ( int *timeout* )**

Kill running executable. First soft kill signal (SIGTERM) is sent to executable. If after timeout seconds executable is still running it's killed completely. Currently this method does not work for Windows OS

**6.241.3.14 Arc::Run::operator bool ( void ) [inline]**

Returns true if object is valid

**6.241.3.15 bool Arc::Run::operator! ( void ) [inline]**

Returns true if object is invalid

**6.241.3.16 int Arc::Run::ReadStderr ( int *timeout*, char \* *buf*, int *size* )**

Read from stderr handle of running executable. Parameter timeout specifies upper limit for which method will block in milliseconds. Negative means infinite. This method may be used while stderr is directed to string. But result is unpredictable. Returns number of read bytes.

**6.241.3.17 int Arc::Run::ReadStdout ( int *timeout*, char \* *buf*, int *size* )**

Read from stdout handle of running executable. Parameter timeout specifies upper limit for which method will block in milliseconds. Negative means infinite. This method may be used while stdout is directed to string. But result is unpredictable. Returns number of read bytes.

**6.241.3.18 int Arc::Run::Result ( void ) [inline]**

Returns exit code of execution.

**6.241.3.19 bool Arc::Run::Running ( void )**

Return true if execution is going on.

### 6.241.3.20 `bool Arc::Run::Start ( void )`

Starts running executable. This method may be called only once.

### 6.241.3.21 `bool Arc::Run::Wait ( int timeout )`

Wait till execution finished or till timeout seconds expires. Returns true if execution is complete.

### 6.241.3.22 `bool Arc::Run::Wait ( void )`

Wait till execution finished

### 6.241.3.23 `int Arc::Run::WriteStdin ( int timeout, const char * buf, int size )`

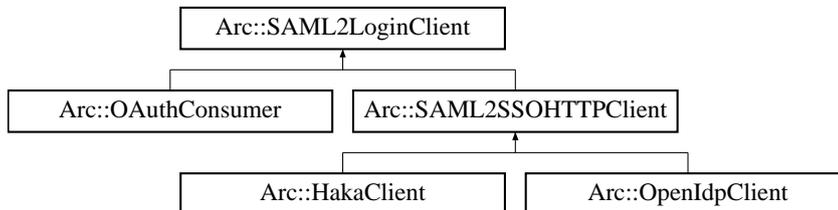
Write to stdin handle of running executable. Parameter timeout specifies upper limit for which method will block in milliseconds. Negative means infinite. This method may be used while stdin is directed to string. But result is unpredictable. Returns number of written bytes.

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

- Run.h

## 6.242 Arc::SAML2LoginClient Class Reference

Inheritance diagram for Arc::SAML2LoginClient:



### Public Member Functions

- `SAML2LoginClient` (const `MCCConfig` cfg, const `URL` url, `std::list< std::string >` idp\_stack)
- virtual `MCC_Status processLogin` (const `std::string` username="", const `std::string` password="")=0
- `MCC_Status findSimpleSAMLInstallation` ()

### 6.242.1 Constructor & Destructor Documentation

#### 6.242.1.1 `Arc::SAML2LoginClient::SAML2LoginClient ( const MCCConfig cfg, const URL url, std::list< std::string > idp_stack )`

list with the idp for nested wayf For example, Confusa can use betawayf.wayf.dk as an identity provider, which is itself only a wayf and shares the metadata with concrete service providers or even further nested

ways. Since due to mutual authentication with metadata, we are obliged to follow the SSO redirects from WAYF to WAYF, the WAYFs are stored in a list.

## 6.242.2 Member Function Documentation

### 6.242.2.1 MCC\_Status Arc::SAML2LoginClient::findSimpleSAMLInstallation ( )

find the location of the simplesamlphp installation on the SP side Will be stored in (\*sso\_pages)[SimpleSAML]

### 6.242.2.2 virtual MCC\_Status Arc::SAML2LoginClient::processLogin ( const std::string username = "", const std::string password = "" ) [pure virtual]

Base interface for all login procedures

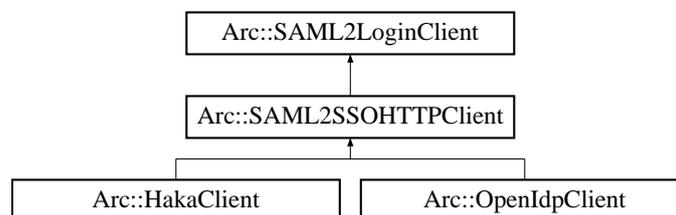
Implemented in [Arc::OAuthConsumer](#) (p. 226), and [Arc::SAML2SSOHTTPClient](#) (p. 276).

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

- SAML2LoginClient.h

## 6.243 Arc::SAML2SSOHTTPClient Class Reference

Inheritance diagram for Arc::SAML2SSOHTTPClient:



### Public Member Functions

- **MCC\_Status processLogin** (const std::string username, const std::string password)
- **MCC\_Status parseDN** (std::string \*dn)
- **MCC\_Status approveCSR** (const std::string approve\_page)
- **MCC\_Status pushCSR** (const std::string b64\_pub\_key, const std::string pub\_key\_hash, std::string \*approve\_page)
- **MCC\_Status storeCert** (const std::string cert\_path, const std::string auth\_token, const std::string b64\_dn)

### Protected Member Functions

- virtual **MCC\_Status processIdPLogin** (const std::string username, const std::string password)=0
- virtual **MCC\_Status processConsent** ()=0
- virtual **MCC\_Status processIdP2Confusa** ()=0

## 6.243.1 Member Function Documentation

### 6.243.1.1 MCC\_Status Arc::SAML2SSOHTTPClient::approveCSR ( const std::string *approve\_page* ) [virtual]

Simulate click on the approve cert signing request link

Implements Arc::SAML2LoginClient (p. 274).

### 6.243.1.2 MCC\_Status Arc::SAML2SSOHTTPClient::parseDN ( std::string \* *dn* ) [virtual]

Parse the used DN from the Confusa about\_you page

Implements Arc::SAML2LoginClient (p. 274).

### 6.243.1.3 virtual MCC\_Status Arc::SAML2SSOHTTPClient::processConsent ( ) [protected, pure virtual]

If the IdP has a consent module and the user has not saved her consent, this method will ask the user for consent to transmission of her data to Confusa

Implemented in Arc::HakaClient (p. 169), and Arc::OpenIdpClient (p. 227).

### 6.243.1.4 virtual MCC\_Status Arc::SAML2SSOHTTPClient::processIdP2Confusa ( ) [protected, pure virtual]

Redirects the user back from identity provider to the Confusa SP

Implemented in Arc::HakaClient (p. 169), and Arc::OpenIdpClient (p. 227).

### 6.243.1.5 virtual MCC\_Status Arc::SAML2SSOHTTPClient::processIdPLogin ( const std::string *username*, const std::string *password* ) [protected, pure virtual]

Actual identity provider parsers for next three methods implemented in subdirectory idp/

Parse identity provider login page and submit username and password in the previsioned way

Implemented in Arc::HakaClient (p. 169), and Arc::OpenIdpClient (p. 227).

### 6.243.1.6 MCC\_Status Arc::SAML2SSOHTTPClient::processLogin ( const std::string *username*, const std::string *password* ) [virtual]

Models complete SAML2 WebSSO authN flow with start -> WAYF -> Login -> (consent) -> start

Implements Arc::SAML2LoginClient (p. 275).

### 6.243.1.7 MCC\_Status Arc::SAML2SSOHTTPClient::pushCSR ( const std::string *b64\_pub\_key*, const std::string *pub\_key\_hash*, std::string \* *approve\_page* ) [virtual]

Send the cert signing request to Confusa for signing

Implements Arc::SAML2LoginClient (p. 274).

### 6.243.1.8 MCC\_Status Arc::SAML2SSOHTTPClient::storeCert ( const std::string cert\_path, const std::string auth\_token, const std::string b64\_dn ) [virtual]

Download the signed certificate from Confusa and store it locally

Implements Arc::SAML2LoginClient (p. 274).

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

- SAML2LoginClient.h

## 6.244 Arc::SAMLToken Class Reference

Class for manipulating SAML Token **Profile** (p. 258).

```
#include <SAMLToken.h>
```

### Public Types

- enum SAMLVersion

### Public Member Functions

- SAMLToken (SOAPEnvelope &soap)
- SAMLToken (SOAPEnvelope &soap, const std::string &certfile, const std::string &keyfile, SAMLVersion saml\_version=SAML2, XMLNode saml\_assertion=XMLNode())
- ~SAMLToken (void)
- operator bool (void)
- bool Authenticate (const std::string &cafile, const std::string &capath)
- bool Authenticate (void)

### 6.244.1 Detailed Description

Class for manipulating SAML Token **Profile** (p. 258). This class is for generating/consuming SAML Token profile. See WS-Security SAML Token **Profile** (p. 258) v1.1 ([www.oasis-open.org/committees/wss](http://www.oasis-open.org/committees/wss)) Currently this class is used by samltoken handler (will appears in src/hed/pdc/samltokensh/) It is not a must to directly called this class. If we need to use SAML Token functionality, we only need to configure the samltoken handler into service and client. Currently, only a minor part of the specification has been implemented.

About how to identify and reference security token for signing message, currently, only the "SAML Assertion Referenced from KeyInfo" (part 3.4.2 of WS-Security SAML Token **Profile** (p. 258) v1.1 specification) is supported, which means the implementation can only process SAML assertion "referenced from Key-Info", and also can only generate SAML Token with SAML assertion "referenced from KeyInfo". More complete support need to implement.

About subject confirmation method, the implementation can process "hold-of-key" (part 3.5.1 of WS-Security SAML Token **Profile** (p. 258) v1.1 specification) subject subject confirmation method.

About SAML version, the implementation can process SAML assertion with SAML version 1.1 and 2.0; can only generate SAML assertion with SAML version 2.0.

In the SAML Token profile, for the hold-of-key subject confirmation method, there are three interaction parts: the attesting entity, the relying party and the issuing authority. In the hold-of-key subject confirmation method, it is the attesting entity's subject identity which will be inserted into the SAML assertion.

Firstly the attesting entity authenticates to issuing authority by using some authentication scheme such as WSS x509 Token profile (Alternatively the username/password authentication scheme or other different authentication scheme can also be used, unless the issuing authority can retrieve the key from a trusted certificate server after firmly establishing the subject's identity under the username/password scheme). So then issuing authority is able to make a definitive statement (sign a SAML assertion) about an act of authentication that has already taken place.

The attesting entity gets the SAML assertion and then signs the soap message together with the assertion by using its private key (the relevant certificate has been authenticated by issuing authority, and its relevant public key has been put into SubjectConfirmation element under saml assertion by issuing authority. Only the actual owner of the saml assertion can do this, as only the subject possesses the private key paired with the public key in the assertion. This establishes an irrefutable connection between the author of the SOAP message and the assertion describing an authentication event.)

The relying party is supposed to trust the issuing authority. When it receives a message from the asserting entity, it will check the saml assertion based on its predetermined trust relationship with the SAML issuing authority, and check the signature of the soap message based on the public key in the saml assertion without directly trust relationship with attesting entity (subject owner).

## 6.244.2 Member Enumeration Documentation

### 6.244.2.1 enum Arc::SAMLToken::SAMLVersion

Since the specification SAMLVersion is for distinguishing two types of saml version. It is used as the parameter of constructor.

## 6.244.3 Constructor & Destructor Documentation

### 6.244.3.1 Arc::SAMLToken::SAMLToken ( SOAPEnvelope & soap )

Constructor. Parse SAML Token information from SOAP header. SAML Token related information is extracted from SOAP header and stored in class variables. And then it the **SAMLToken** (p. 277) object will be used for authentication.

#### Parameters

*soap* The SOAP message which contains the **SAMLToken** (p. 277) in the soap header

### 6.244.3.2 Arc::SAMLToken::SAMLToken ( SOAPEnvelope & soap, const std::string & certfile, const std::string & keyfile, SAMLVersion saml\_version = SAML2, XMLNode saml\_assertion = XMLNode () )

Constructor. Add SAML Token information into the SOAP header. Generated token contains elements SAML token and signature, and is meant to be used for authentication on the consuming side. This constructor is for a specific SAML Token profile usage, in which the attesting entity signs the SAML assertion for itself (self-sign). This usage implicitly requires that the relying party trust the attesting entity. More general (requires issuing authority) usage will be provided by other constructor. And the under-developing SAML service will be used as the issuing authority.

**Parameters**

*soap* The SOAP message to which the SAML Token will be inserted.

*certfile* The certificate file.

*keyfile* The key file which will be used to create signature.

*samlversion* The SAML version, only SAML2 is supported currently.

*samlassertion* The SAML assertion got from 3rd party, and used for protecting the SOAP message; If not present, then self-signed assertion will be generated.

**6.244.3.3 Arc::SAMLToken::~~SAMLToken ( void )**

Deconstructor. Nothing to be done except finalizing the xmlsec library.

**6.244.4 Member Function Documentation****6.244.4.1 bool Arc::SAMLToken::Authenticate ( const std::string & *cafile*, const std::string & *capath* )**

Check signature by using the trusted certificates It is used by relying parting after calling **SAMLToken(SOAPEnvelope& soap)** (p. 278) This method will check the SAML assertion based on the trusted certificated specified as parameter *cafile* or *capath*; and also check the signature to soap message (the signature is generated by attesting entity by signing soap body together with SAML assertion) by using the public key inside SAML assestion.

**Parameters**

*cafile* ca file

*capath* ca directory

**6.244.4.2 bool Arc::SAMLToken::Authenticate ( void )**

Check signature by using the cert information in soap message

**6.244.4.3 Arc::SAMLToken::operator bool ( void )**

Returns true of constructor succeeded

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

- SAMLToken.h

**6.245 Arc::ScalableTime< T > Class Template Reference**

**template<class T> class Arc::ScalableTime< T >**

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

- JobDescription.h

## 6.246 Arc::ScalableTime< int > Class Template Reference

```
template<> class Arc::ScalableTime< int >
```

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

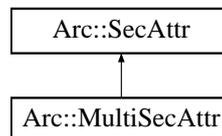
- JobDescription.h

## 6.247 Arc::SecAttr Class Reference

This is an abstract interface to a security attribute.

```
#include <SecAttr.h>
```

Inheritance diagram for Arc::SecAttr:



### Public Member Functions

- **SecAttr** ()
- bool **operator==** (const **SecAttr** &b) const
- bool **operator!=** (const **SecAttr** &b) const
- virtual **operator bool** () const
- virtual bool **Export** (**SecAttrFormat** format, std::string &val) const
- virtual bool **Export** (**SecAttrFormat** format, **XMLNode** &val) const
- virtual bool **Import** (**SecAttrFormat** format, const std::string &val)

### Static Public Attributes

- static **SecAttrFormat** **ARCAuth**
- static **SecAttrFormat** **XACML**
- static **SecAttrFormat** **SAML**
- static **SecAttrFormat** **GACL**

### 6.247.1 Detailed Description

This is an abstract interface to a security attribute. This class is meant to be inherited to implement security attributes. Depending on what data it needs to store inheriting classes may need to implement constructor and destructor. They must however override the equality and the boolean operators. The equality is meant to compare security attributes. The prototype implies that all attributes are comparable to all others. This behaviour should be modified as needed by using `dynamic_cast` operations. The boolean cast operation is meant to embody "nullness" if that is applicable to the particular type.

## 6.247.2 Member Function Documentation

### 6.247.2.1 virtual bool Arc::SecAttr::Export ( SecAttrFormat *format*, std::string & *val* ) const [virtual]

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute.

### 6.247.2.2 virtual bool Arc::SecAttr::Export ( SecAttrFormat *format*, XMLNode & *val* ) const [virtual]

Convert internal structure into specified format. Returns false if format is not supported/suitable for this attribute. XML node referenced by is turned into top level element of specified format.

Reimplemented in **Arc::MultiSecAttr** (p. 221).

### 6.247.2.3 virtual bool Arc::SecAttr::Import ( SecAttrFormat *format*, const std::string & *val* ) [virtual]

Fills internal structure from external object of specified format. Returns false if failed to do. The usage pattern for this method is not defined and it is provided only to make class symmetric. Hence it's implementation is not required yet.

### 6.247.2.4 virtual Arc::SecAttr::operator bool ( ) const [virtual]

This function should return false if the value is to be considered null, e.g. if it hasn't been set or initialized. In other cases it should return true.

Reimplemented in **Arc::MultiSecAttr** (p. 221).

### 6.247.2.5 bool Arc::SecAttr::operator!= ( const SecAttr & *b* ) const [inline]

This is a convenience function to allow the usage of "not equal" conditions and need not be overridden.

### 6.247.2.6 bool Arc::SecAttr::operator== ( const SecAttr & *b* ) const [inline]

This function should (in inheriting classes) return true if this and *b* are considered to represent same content. Identifying and restricting the type of *b* should be done using `dynamic_cast` operations. Currently it is not defined how comparison methods to be used. Hence their implementation is not required.

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

- SecAttr.h

## 6.248 Arc::SecAttrFormat Class Reference

Export/import format.

```
#include <SecAttr.h>
```

### 6.248.1 Detailed Description

Export/import format. Format is identified by textual identity string. Class description includes basic formats only. That list may be extended.

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

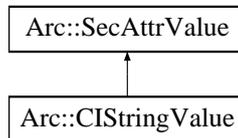
- SecAttr.h

## 6.249 Arc::SecAttrValue Class Reference

This is an abstract interface to a security attribute.

```
#include <SecAttrValue.h>
```

Inheritance diagram for Arc::SecAttrValue:



### Public Member Functions

- `bool operator==(SecAttrValue &b)`
- `bool operator!=(SecAttrValue &b)`
- `virtual operator bool ()`

### 6.249.1 Detailed Description

This is an abstract interface to a security attribute. This class is meant to be inherited to implement security attributes. Depending on what data it needs to store inheriting classes may need to implement constructor and destructor. They must however override the equality and the boolean operators. The equality is meant to compare security attributes. The prototype implies that all attributes are comparable to all others. This behaviour should be modified as needed by using `dynamic_cast` operations. The boolean cast operation is meant to embody "nullness" if that is applicable to the particular type.

### 6.249.2 Member Function Documentation

#### 6.249.2.1 virtual Arc::SecAttrValue::operator bool ( ) [virtual]

This function should return false if the value is to be considered null, e g if it hasn't been set or initialized. In other cases it should return true.

Reimplemented in `Arc::CIStrngValue` (p. 69).

#### 6.249.2.2 bool Arc::SecAttrValue::operator!=( SecAttrValue & b )

This is a convenience function to allow the usage of "not equal" conditions and need not be overridden.

### 6.249.2.3 bool Arc::SecAttrValue::operator==( SecAttrValue & b )

This function should (in inheriting classes) return true if this and b are considered to be the same. Identifying and restricting the type of b should be done using dynamic\_cast operations.

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

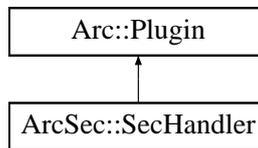
- SecAttrValue.h

## 6.250 ArcSec::SecHandler Class Reference

Base class for simple security handling plugins.

```
#include <SecHandler.h>
```

Inheritance diagram for ArcSec::SecHandler:



### 6.250.1 Detailed Description

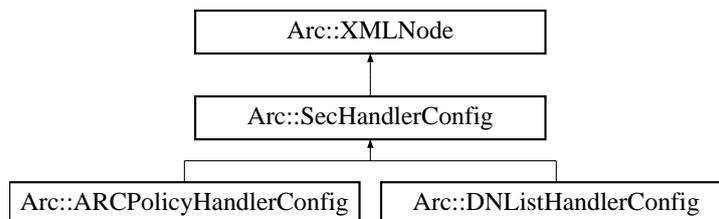
Base class for simple security handling plugins. This virtual class defines method Handle() which processes security related information/attributes in Message and optionally makes security decision. Instances of such classes are normally arranged in chains and are called on incoming and outgoing messages in various MCC and Service plugins. Return value of Handle() defines either processing should continue (true) or stop with error (false). Configuration of **SecHandler** (p. 283) is consumed during creation of instance through XML subtree fed to constructor.

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

- SecHandler.h

## 6.251 Arc::SecHandlerConfig Class Reference

Inheritance diagram for Arc::SecHandlerConfig:



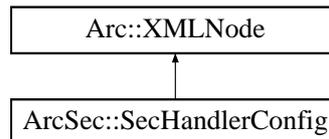
The documentation for this class was generated from the following file:

- ClientInterface.h

## 6.252 ArcSec::SecHandlerConfig Class Reference

```
#include <SecHandler.h>
```

Inheritance diagram for ArcSec::SecHandlerConfig:



### 6.252.1 Detailed Description

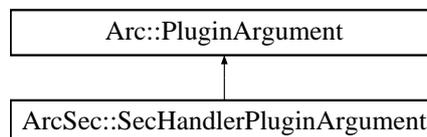
Helper class to create **Security** (p. 284) Handler configuration

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

- SecHandler.h

## 6.253 ArcSec::SecHandlerPluginArgument Class Reference

Inheritance diagram for ArcSec::SecHandlerPluginArgument:



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

- SecHandler.h

## 6.254 ArcSec::Security Class Reference

Common stuff used by security related classes.

```
#include <Security.h>
```

### 6.254.1 Detailed Description

Common stuff used by security related classes. This class is just a place where to put common stuff that is used by security related classes. So far it only contains a logger.

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

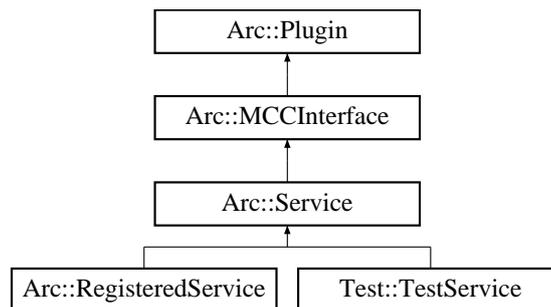
- Security.h

## 6.255 Arc::Service Class Reference

**Service** (p. 285) - last component in a **Message** (p. 210) Chain.

```
#include <Service.h>
```

Inheritance diagram for Arc::Service:



### Public Member Functions

- **Service** (**Config** \*)
- virtual void **AddSecHandler** (**Config** \*cfg, **ArcSec::SecHandler** \*sechandler, const std::string &label="")
- virtual bool **RegistrationCollector** (**XMLNode** &doc)
- virtual std::string **getID** ()

### Protected Member Functions

- bool **ProcessSecHandlers** (**Message** &message, const std::string &label="") const

### Protected Attributes

- std::map< std::string, std::list< **ArcSec::SecHandler** \* > > **sechandlers\_**

### Static Protected Attributes

- static **Logger** **logger**

#### 6.255.1 Detailed Description

**Service** (p. 285) - last component in a **Message** (p. 210) Chain. This class which defines interface and common functionality for every **Service** (p. 285) plugin. Interface is made of method **process()** (p. 207) which is called by **Plexer** (p. 247) or **MCC** (p. 202) class. There is one **Service** (p. 285) object created for every service description processed by **Loader** (p. 191) class objects. Classes derived from **Service** (p. 285) class must implement **process()** (p. 207) method of **MCCInterface** (p. 207). It is up to developer how internal

state of service is stored and communicated to other services and external utilities. **Service** (p. 285) is free to expect any type of payload passed to it and generate any payload as well. Useful types depend on MCCs in chain which leads to that service. For example if service is expected to be linked to SOAP **MCC** (p. 202) it must accept and generate messages with **PayloadSOAP** (p. 234) payload. Method **process()** (p. 207) of class derived from **Service** (p. 285) class may be called concurrently in multiple threads. Developers must take that into account and write thread-safe implementation. Simple example of service is provided in `/src/tests/echo/echo.cpp` of source tree. The way to write client counterpart of corresponding service is undefined yet. For example see `/src/tests/echo/test.cpp`.

## 6.255.2 Constructor & Destructor Documentation

### 6.255.2.1 `Arc::Service::Service ( Config * )`

Example constructor - Server takes at least its configuration subtree

## 6.255.3 Member Function Documentation

### 6.255.3.1 `virtual void Arc::Service::AddSecHandler ( Config * cfg, ArcSec::SecHandler * sechandler, const std::string & label = "" ) [virtual]`

Add security components/handlers to this **MCC** (p. 202). For more information please see description of **MCC::AddSecHandler** (p. 203)

### 6.255.3.2 `virtual std::string Arc::Service::getID ( ) [inline, virtual]`

**Service** (p. 285) may implement own service identifier gathering method. This method returns identifier of service which is used for registering it Information Services.

### 6.255.3.3 `bool Arc::Service::ProcessSecHandlers ( Message & message, const std::string & label = "" ) const [protected]`

Executes security handlers of specified queue. For more information please see description of **MCC::ProcessSecHandlers** (p. 203)

### 6.255.3.4 `virtual bool Arc::Service::RegistrationCollector ( XMLNode & doc ) [virtual]`

**Service** (p. 285) specific registration collector, used for generate service registrations. In implemented service this method should generate GLUE2 document with part of service description which service wishes to advertise to Information Services.

## 6.255.4 Field Documentation

### 6.255.4.1 `Logger Arc::Service::logger [static, protected]`

**Logger** (p. 195) object used to print messages generated by this class.

### 6.255.4.2 std::map<std::string,std::list<ArcSec::SecHandler\*> > Arc::Service::sechandlers\_ [protected]

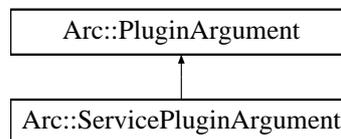
Set of labeled authentication and authorization handlers. MCC (p.202) calls sequence of handlers at specific point depending on associated identifier. in most aces those are "in" and "out" for incoming and outgoing messages correspondingly.

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

- Service.h

## 6.256 Arc::ServicePluginArgument Class Reference

Inheritance diagram for Arc::ServicePluginArgument:



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

- Service.h

## 6.257 Arc::SimpleCondition Class Reference

Helper function to create simple thread.

```
#include <Thread.h>
```

### Public Member Functions

- void **lock** (void)
- void **unlock** (void)
- void **signal** (void)
- void **signal\_nonblock** (void)
- void **broadcast** (void)
- void **wait** (void)
- void **wait\_nonblock** (void)
- bool **wait** (int t)
- void **reset** (void)

### 6.257.1 Detailed Description

Helper function to create simple thread. It takes care of all peculiarities of Glib::Thread API. As result it runs function 'func' with argument 'arg' in a separate thread. The created thread will be joinable. Returns true on success. This function is currently disable because it is not clear if joinability is a needed feature Simple triggered condition. Provides condition and semaphore objects in one element.

## 6.257.2 Member Function Documentation

### 6.257.2.1 void Arc::SimpleCondition::broadcast ( void ) [inline]

Signal about condition to all waiting threads

### 6.257.2.2 void Arc::SimpleCondition::lock ( void ) [inline]

Acquire semaphor

### 6.257.2.3 void Arc::SimpleCondition::reset ( void ) [inline]

Reset object to initial state

### 6.257.2.4 void Arc::SimpleCondition::signal ( void ) [inline]

Signal about condition

### 6.257.2.5 void Arc::SimpleCondition::signal\_nonblock ( void ) [inline]

Signal about condition without using semaphor

### 6.257.2.6 void Arc::SimpleCondition::unlock ( void ) [inline]

Release semaphor

### 6.257.2.7 bool Arc::SimpleCondition::wait ( int t ) [inline]

Wait for condition no longer than t milliseconds

### 6.257.2.8 void Arc::SimpleCondition::wait ( void ) [inline]

Wait for condition

### 6.257.2.9 void Arc::SimpleCondition::wait\_nonblock ( void ) [inline]

Wait for condition without using semaphor

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

- Thread.h

## 6.258 Arc::SimpleCounter Class Reference

### Public Member Functions

- bool wait (int t)

## 6.258.1 Member Function Documentation

### 6.258.1.1 bool Arc::SimpleCounter::wait ( int *t* ) [inline]

Wait for condition no longer than *t* milliseconds

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

- Thread.h

## 6.259 Arc::SOAPMessage Class Reference

**Message** (p. 210) restricted to SOAP payload.

```
#include <SOAPMessage.h>
```

### Public Member Functions

- **SOAPMessage** (void)
- **SOAPMessage** (long *msg\_ptr\_addr*)
- **SOAPMessage** (**Message** &*msg*)
- ~**SOAPMessage** (void)
- SOAPEnvelope \* **Payload** (void)
- void **Payload** (SOAPEnvelope \**new\_payload*)
- **MessageAttributes** \* **Attributes** (void)

### 6.259.1 Detailed Description

**Message** (p. 210) restricted to SOAP payload. This is a special **Message** (p. 210) intended to be used in language bindings for programming languages which are not flexible enough to support all kinds of Payloads. It is passed through chain of MCCs and works like the **Message** (p. 210) but can carry only SOAP content.

### 6.259.2 Constructor & Destructor Documentation

#### 6.259.2.1 Arc::SOAPMessage::SOAPMessage ( void ) [inline]

Dummy constructor

#### 6.259.2.2 Arc::SOAPMessage::SOAPMessage ( long *msg\_ptr\_addr* )

Copy constructor. Used by language bindings

#### 6.259.2.3 Arc::SOAPMessage::SOAPMessage ( Message & *msg* )

Copy constructor. Ensures shallow copy.

#### 6.259.2.4 Arc::SOAPMessage::~~SOAPMessage ( void )

Destructor does not affect refered objects

### 6.259.3 Member Function Documentation

#### 6.259.3.1 MessageAttributes\* Arc::SOAPMessage::Attributes ( void ) [inline]

Returns a pointer to the current attributes object or NULL if no attributes object has been assigned.

#### 6.259.3.2 SOAPEnvelope\* Arc::SOAPMessage::Payload ( void )

Returns pointer to current payload or NULL if no payload assigned.

#### 6.259.3.3 void Arc::SOAPMessage::Payload ( SOAPEnvelope \* new\_payload )

Replace payload with a COPY of new one

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

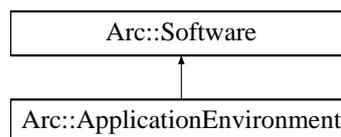
- SOAPMessage.h

## 6.260 Arc::Software Class Reference

Used to represent software (names and version) and comparison.

```
#include <Software.h>
```

Inheritance diagram for Arc::Software:



### Public Types

- enum **ComparisonOperatorEnum** {  
**NOTEQUAL** = 0, **EQUAL** = 1, **GREATERTHAN** = 2, **LESSTHAN** = 3,  
**GREATERTHANOEQUAL** = 4, **LESSTHANOEQUAL** = 5 }
- typedef bool(Software::\* **ComparisonOperator** )(const **Software** &) const

### Public Member Functions

- **Software** ()
- **Software** (const std::string &name\_version)
- **Software** (const std::string &name, const std::string &version)

- **Software** (const std::string &family, const std::string &name, const std::string &version)
- bool **empty** () const
- bool **operator==** (const **Software** &sw) const
- bool **operator!=** (const **Software** &sw) const
- bool **operator>** (const **Software** &sw) const
- bool **operator<** (const **Software** &sw) const
- bool **operator>=** (const **Software** &sw) const
- bool **operator<=** (const **Software** &sw) const
- std::string **operator()** () const
- **operator std::string** (void) const
- const std::string & **getFamily** () const
- const std::string & **getName** () const
- const std::string & **getVersion** () const

### Static Public Member Functions

- static **ComparisonOperator** **convert** (const **ComparisonOperatorEnum** &co)
- static std::string **toString** (**ComparisonOperator** co)

### Static Public Attributes

- static const std::string **VERSIONTOKENS**

### Friends

- std::ostream & **operator<<** (std::ostream &out, const **Software** &sw)

## 6.260.1 Detailed Description

Used to represent software (names and version) and comparison. The **Software** (p. 290) class is used to represent the name of a piece of software internally. Generally software are identified by a name and possibly a version number. Some software can also be categorized by type or family (compilers, operating system, etc.). A software object can be compared to other software objects using the comparison operators contained in this class. The basic usage of this class is to test if some specified software requirement (**SoftwareRequirement** (p. 297)) are fulfilled, by using the comparability of the class.

Internally the **Software** (p. 290) object is represented by a family and name identifier, and the software version is tokenized at the characters defined in **VERSIONTOKENS**, and stored as a list of tokens.

## 6.260.2 Member Typedef Documentation

### 6.260.2.1 typedef bool(Software::\* Arc::Software::ComparisonOperator)(const Software &) const

Definition of a comparison operator method pointer.

This typedef defines a comparison operator method pointer.

**See also**

**operator==** (p. 295),  
**operator!=** (p. 294),  
**operator>** (p. 296),  
**operator<** (p. 295),  
**operator>=** (p. 296),  
**operator<=** (p. 295),  
**ComparisonOperatorEnum** (p. 292).

### 6.260.3 Member Enumeration Documentation

#### 6.260.3.1 enum Arc::Software::ComparisonOperatorEnum

Comparison operator enum.

The **ComparisonOperatorEnum** (p. 292) enumeration is a 1-1 correspondance between the defined comparison method operators (**Software::ComparisonOperator** (p. 291)), and can be used in circumstances where method pointers are not supported.

**Enumerator:**

**NOTEQUAL** see **operator!=** (p. 294)  
**EQUAL** see **operator==** (p. 295)  
**GREATERTHAN** see **operator>** (p. 296)  
**LESSTHAN** see **operator<** (p. 295)  
**GREATERTHANOREQUAL** see **operator>=** (p. 296)  
**LESSTHANOREQUAL** see **operator<=** (p. 295)

### 6.260.4 Constructor & Destructor Documentation

#### 6.260.4.1 Arc::Software::Software ( ) [**inline**]

Dummy constructor.

This constructor creates a empty object.

#### 6.260.4.2 Arc::Software::Software ( const std::string & *name\_version* )

Create a **Software** (p. 290) object.

Create a **Software** (p. 290) object from a single string composed of a name and a version part. The created object will contain a empty family part. The name and version part of the string will be split at the first occurence of a dash (-) which is followed by a digit (0-9). If the string does not contain such a pattern, the passed string will be taken to be the name and version will be empty.

**Parameters**

*name\_version* should be a string composed of the name and version of the software to represent.

### 6.260.4.3 Arc::Software::Software ( const std::string & *name*, const std::string & *version* )

Create a **Software** (p. 290) object.

Create a **Software** (p. 290) object with the specified name and version. The family part will be left empty.

#### Parameters

*name* the software name to represent.

*version* the software version to represent.

### 6.260.4.4 Arc::Software::Software ( const std::string & *family*, const std::string & *name*, const std::string & *version* )

Create a **Software** (p. 290) object.

Create a **Software** (p. 290) object with the specified family, name and version.

#### Parameters

*family* the software family to represent.

*name* the software name to represent.

*version* the software version to represent.

## 6.260.5 Member Function Documentation

### 6.260.5.1 static ComparisonOperator Arc::Software::convert ( const ComparisonOperatorEnum & *co* ) [static]

Convert a **ComparisonOperatorEnum** (p. 292) value to a comparison method pointer.

The passed **ComparisonOperatorEnum** (p. 292) will be converted to a comparison method pointer defined by the **Software::ComparisonOperator** (p. 291) typedef.

This static method is not defined in language bindings created with Swig, since method pointers are not supported by Swig.

#### Parameters

*co* a **ComparisonOperatorEnum** (p. 292) value.

#### Returns

A method pointer to a comparison method is returned.

### 6.260.5.2 bool Arc::Software::empty ( ) const [inline]

Indicates whether the object is empty.

#### Returns

true if the name of this object is empty, otherwise false.

**6.260.5.3** `const std::string& Arc::Software::getFamily ( ) const [inline]`

Get family.

**Returns**

The family the represented software belongs to is returned.

**6.260.5.4** `const std::string& Arc::Software::getName ( ) const [inline]`

Get name.

**Returns**

The name of the represented software is returned.

**6.260.5.5** `const std::string& Arc::Software::getVersion ( ) const [inline]`

Get version.

**Returns**

The version of the represented software is returned.

**6.260.5.6** `Arc::Software::operator std::string ( void ) const [inline]`

Cast to string.

This casting operator behaves exactly as `operator()()` (p. 294) does. The cast is used like `(std::string) <software-object>`.

**See also**

`operator()()` (p. 294).

References `operator()()`.

**6.260.5.7** `bool Arc::Software::operator!= ( const Software & sw ) const [inline]`

Inequality operator (non-trivial behaviour).

The inequality operator should be used to test if two **Software** (p. 290) objects are of different versions but share the same name and family. So it should not be used to test if two **Software** (p. 290) objects differ in either name, version or family. Two **Software** (p. 290) objects are unequal if they share the same name and family but have different versions and the versions are non-empty.

**Parameters**

`sw` is the RHS **Software** (p. 290) object.

**Returns**

`true` when the two objects are unequal, otherwise `false`.

### 6.260.5.8 std::string Arc::Software::operator() ( ) const

Get string representation.

Returns the string representation of this object, which is 'family'-'name'-'version'.

#### Returns

The string representation of this object is returned.

#### See also

operator std::string().

Referenced by operator std::string().

### 6.260.5.9 bool Arc::Software::operator< ( const Software & *sw* ) const [inline]

Less-than operator.

The behaviour of this less-than operator is equivalent to the greater-than operator (**operator>()** (p. 296)) with the LHS and RHS swapped.

#### Parameters

*sw* is the RHS object.

#### Returns

`true` if the LHS is less than the RHS, otherwise `false`.

#### See also

**operator>()** (p. 296).

### 6.260.5.10 bool Arc::Software::operator<= ( const Software & *sw* ) const [inline]

Less-than or equal operator.

The LHS object is greater than or equal to the RHS object if the LHS equal the RHS (**operator==()** (p. 295)) or if the LHS is greater than the RHS (**operator>()** (p. 296)).

#### Parameters

*sw* is the RHS object.

#### Returns

`true` if the LHS is less than or equal the RHS, otherwise `false`.

#### See also

**operator==()** (p. 295),

**operator<()** (p. 295).

### 6.260.5.11 `bool Arc::Software::operator==( const Software & sw ) const [inline]`

Equality operator.

Two **Software** (p. 290) objects are equal if they are of the same family, and if they have the same name. If BOTH objects specifies a version they must also equal, for the objects to be equal. Otherwise the two objects does not equal. This operator can also be represented by the **Software::EQUAL** (p. 292) **ComparisonOperatorEnum** (p. 292) value.

#### Parameters

`sw` is the RHS **Software** (p. 290) object.

#### Returns

`true` when the two objects equals, otherwise `false`.

### 6.260.5.12 `bool Arc::Software::operator>( const Software & sw ) const`

Greater-than operator.

For the LHS object to be greater than the RHS object they must first share the same family and name and have non-empty versions. Then, the first version token of each object is compared and if they are identical, the two next version tokens will be compared. If not identical, the two tokens will be parsed as integers, and if parsing fails the LHS is not greater than the RHS. If parsing succeeds and the integers equals, the two next tokens will be compared, otherwise the comparison is resolved by the integer comparison.

If the LHS contains more version tokens than the RHS, and the comparison have not been resolved at the point of equal number of tokens, then if the additional tokens contains a token which cannot be parsed to a integer the LHS is not greater than the RHS. If the parsed integer is not 0 then the LHS is greater than the RHS. If the rest of the additional tokens are 0, the LHS is not greater than the RHS.

If the RHS contains more version tokens than the LHS and comparison have not been resolved at the point of equal number of tokens, or simply if comparison have not been resolved at the point of equal number of tokens, then the LHS is not greater than the RHS.

#### Parameters

`sw` is the RHS object.

#### Returns

`true` if the LHS is greater than the RHS, otherwise `false`.

### 6.260.5.13 `bool Arc::Software::operator>=( const Software & sw ) const [inline]`

Greater-than or equal operator.

The LHS object is greater than or equal to the RHS object if the LHS equal the RHS (**operator==(** (p. 295)) or if the LHS is greater than the RHS (**operator>()** (p. 296)).

#### Parameters

`sw` is the RHS object.

**Returns**

`true` if the LHS is greater than or equal the RHS, otherwise `false`.

**See also**

`operator==( )` (p. 295),  
`operator>( )` (p. 296).

**6.260.5.14 static std::string Arc::Software::toString ( ComparisonOperator *co* ) [static]**

Convert **Software::ComparisonOperator** (p. 291) to a string.

This method is not available in language bindings created by Swig, since method pointers are not supported by Swig.

**Parameters**

*co* is a **Software::ComparisonOperator** (p. 291).

**Returns**

The string representation of the passed **Software::ComparisonOperator** (p. 291) is returned.

**6.260.6 Friends And Related Function Documentation****6.260.6.1 std::ostream& operator<< ( std::ostream & *out*, const Software & *sw* ) [friend]**

Write **Software** (p. 290) string representation to a `std::ostream`.

Write the string representation of a **Software** (p. 290) object to a `std::ostream`.

**Parameters**

*out* is a `std::ostream` to write the string representation of the **Software** (p. 290) object to.

*sw* is the **Software** (p. 290) object to write to the `std::ostream`.

**Returns**

The passed `std::ostream` *out* is returned.

**6.260.7 Field Documentation****6.260.7.1 const std::string Arc::Software::VERSIONTOKENS [static]**

Tokens used to split version string.

This string constant specifies which tokens will be used to split the version string.

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

- Software.h

## 6.261 Arc::SoftwareRequirement Class Reference

Class used to express and resolve version requirements on software.

```
#include <Software.h>
```

### Public Member Functions

- **SoftwareRequirement** (bool requiresAll=false)
- **SoftwareRequirement** (const **Software** &sw, **Software::ComparisonOperator** swCo-mOp=&Software::operator==, bool requiresAll=false)
- **SoftwareRequirement** (const **Software** &sw, **Software::ComparisonOperatorEnum** co, bool requiresAll=false)
- **SoftwareRequirement** & **operator=** (const **SoftwareRequirement** &sr)
- **SoftwareRequirement** (const **SoftwareRequirement** &sr)
- void **add** (const **Software** &sw, **Software::ComparisonOperator** swCo-mOp=&Software::operator==)
- void **add** (const **Software** &sw, **Software::ComparisonOperatorEnum** co)
- bool **isRequiringAll** () const
- void **setRequirement** (bool all)
- bool **isSatisfied** (const **Software** &sw) const
- bool **isSatisfied** (const std::list< **Software** > &swList) const
- bool **isSatisfied** (const std::list< **ApplicationEnvironment** > &swList) const
- bool **selectSoftware** (const **Software** &sw)
- bool **selectSoftware** (const std::list< **Software** > &swList)
- bool **selectSoftware** (const std::list< **ApplicationEnvironment** > &swList)
- bool **isResolved** () const
- bool **empty** () const
- void **clear** ()
- const std::list< **Software** > & **getSoftwareList** () const
- const std::list< **Software::ComparisonOperator** > & **getComparisonOperatorList** () const

### 6.261.1 Detailed Description

Class used to express and resolve version requirements on software. A requirement in this class is defined as a pair composed of a **Software** (p. 290) object and either a **Software::ComparisonOperator** (p. 291) method pointer or equally a **Software::ComparisonOperatorEnum** (p. 292) enum value. A **SoftwareRequirement** (p. 297) object can contain multiple of such requirements, and then it can specified if all these requirements should be satisfied, or if it is enough to satisfy only one of them. The requirements can be satisfied by a single **Software** (p. 290) object or a list of either **Software** (p. 290) or **ApplicationEnvironment** (p. 50) objects, by using the method **isSatisfied**() (p. 302). This class also contain a number of methods (**selectSoftware**() (p. 304)) to select **Software** (p. 290) objects which are satisfying the requirements, and in this way resolving requirements.

### 6.261.2 Constructor & Destructor Documentation

#### 6.261.2.1 Arc::SoftwareRequirement::SoftwareRequirement ( bool *requiresAll* = *false* ) [inline]

Create a empty **SoftwareRequirement** (p. 297) object.

The created **SoftwareRequirement** (p. 297) object will contain no requirements.

**Parameters**

*requiresAll* indicates whether the all requirements have to be satisfied (`true`) or if only a single one (`false`), the default is that only a single requirement need to be satisfied.

### 6.261.2.2 Arc::SoftwareRequirement::SoftwareRequirement ( const Software & *sw*, Software::ComparisonOperator *swComOp* = &Software::operator==, bool *requiresAll* = false )

Create a **SoftwareRequirement** (p. 297) object.

The created **SoftwareRequirement** (p. 297) object will contain one requirement specified by the **Software** (p. 290) object *sw*, and the **Software::ComparisonOperator** (p. 291) *swComOp*.

This constructor is not available in language bindings created by Swig, since method pointers are not supported by Swig, see **SoftwareRequirement(const Software&, Software::ComparisonOperatorEnum, bool)** (p. 299) instead.

**Parameters**

*sw* is the **Software** (p. 290) object of the requirement to add.

*swComOp* is the **Software::ComparisonOperator** (p. 291) of the requirement to add.

*requiresAll* indicates whether the all requirements have to be satisfied (`true`) or if only a single one (`false`), the default is that only a single requirement need to be satisfied.

### 6.261.2.3 Arc::SoftwareRequirement::SoftwareRequirement ( const Software & *sw*, Software::ComparisonOperatorEnum *co*, bool *requiresAll* = false )

Create a **SoftwareRequirement** (p. 297) object.

The created **SoftwareRequirement** (p. 297) object will contain one requirement specified by the **Software** (p. 290) object *sw*, and the **Software::ComparisonOperatorEnum** (p. 292) *co*.

**Parameters**

*sw* is the **Software** (p. 290) object of the requirement to add.

*co* is the **Software::ComparisonOperatorEnum** (p. 292) of the requirement to add.

*requiresAll* indicates whether the all requirements have to be satisfied (`true`) or if only a single one (`false`), the default is that only a single requirement need to be satisfied.

### 6.261.2.4 Arc::SoftwareRequirement::SoftwareRequirement ( const SoftwareRequirement & *sr* ) [inline]

Copy constructor.

Create a **SoftwareRequirement** (p. 297) object from another **SoftwareRequirement** (p. 297) object.

**Parameters**

*sr* is the **SoftwareRequirement** (p. 297) object to make a copy of.

### 6.261.3 Member Function Documentation

**6.261.3.1** `void Arc::SoftwareRequirement::add ( const Software & sw,  
Software::ComparisonOperator swComOp = &Software::operator== )`

Add a **Software** (p. 290) object a corresponding comparison operator to this object.

Adds software name and version to list of requirements and associates the comparison operator with it (equality by default).

This method is not available in language bindings created by Swig, since method pointers are not supported by Swig, see `add(const Software&, Software::ComparisonOperatorEnum)` (p. 300) instead.

#### Parameters

*sw* is the **Software** (p. 290) object to add as part of a requirement.

*swComOp* is the **Software::ComparisonOperator** (p. 291) method pointer to add as part of a requirement, the default operator will be **Software::operator==()** (p. 295).

**6.261.3.2** `void Arc::SoftwareRequirement::add ( const Software & sw,  
Software::ComparisonOperatorEnum co )`

Add a **Software** (p. 290) object a corresponding comparison operator to this object.

Adds software name and version to list of requirements and associates the comparison operator with it (equality by default).

#### Parameters

*sw* is the **Software** (p. 290) object to add as part of a requirement.

*co* is the **Software::ComparisonOperatorEnum** (p. 292) value to add as part of a requirement, the default enum will be **Software::EQUAL** (p. 292).

**6.261.3.3** `void Arc::SoftwareRequirement::clear ( ) [inline]`

Clear the object.

The requirements in this object will be cleared when invoking this method.

**6.261.3.4** `bool Arc::SoftwareRequirement::empty ( ) const [inline]`

Test if the object is empty.

#### Returns

`true` if this object do no contain any requirements, otherwise `false`.

**6.261.3.5** `const std::list<Software::ComparisonOperator>&  
Arc::SoftwareRequirement::getComparisonOperatorList ( ) const  
[inline]`

Get list of comparison operators.

**Returns**

The list of internally stored comparison operators is returned.

**See also**

**Software::ComparisonOperator** (p. 291),  
**getSoftwareList** (p. 301).

**6.261.3.6** `const std::list<Software>& Arc::SoftwareRequirement::getSoftwareList ( ) const [inline]`

Get list of **Software** (p. 290) objects.

**Returns**

The list of internally stored **Software** (p. 290) objects is returned.

**See also**

**Software** (p. 290),  
**getComparisonOperatorList** (p. 300).

**6.261.3.7** `bool Arc::SoftwareRequirement::isRequiringAll ( ) const [inline]`

Indicates whether all requirements has to be satisfied.

This method returns `true` if all requirements has to be satisfied. If only one requirement has to be satisfied, `false` is returned.

**Returns**

`true` if all requirements has to be satisfied, otherwise `false`.

**See also**

**setRequirement** (p. 304).

**6.261.3.8** `bool Arc::SoftwareRequirement::isResolved ( ) const`

Indicates whether requirements have been resolved or not.

If specified that only one requirement has to be satisfied, then for this object to be resolved it can only contain one requirement and it has use the equal operator (**Software::operator==** (p. 295)).

If specified that all requirements has to be satisfied, then for this object to be resolved each requirement must have a **Software** (p. 290) object with a unique family/name composition, i.e. no other requirements have a **Software** (p. 290) object with the same family/name composition, and each requirement must use the equal operator (**Software::operator==** (p. 295)).

If this object has been resolved then `true` is returned when invoking this method, otherwise `false` is returned.

**Returns**

`true` if this object have been resolved, otherwise `false`.

### 6.261.3.9 `bool Arc::SoftwareRequirement::isSatisfied ( const std::list< ApplicationEnvironment > & swList ) const`

Test if requirements are satisfied.

This method behaves in exactly the same way as the `isSatisfied(const Software&) const` (p. 302) method does.

#### Parameters

`swList` is the list of `ApplicationEnvironment` (p. 50) objects which should be used to try satisfy the requirements.

#### Returns

`true` if requirements are satisfied, otherwise `false`.

#### See also

`isSatisfied(const Software&) const` (p. 302),  
`isSatisfied(const std::list<Software>&) const` (p. 302),  
`selectSoftware(const std::list<ApplicationEnvironment>&)` (p. 304),  
`isResolved() const` (p. 301).

### 6.261.3.10 `bool Arc::SoftwareRequirement::isSatisfied ( const Software & sw ) const [inline]`

Test if requirements are satisfied.

Returns `true` if the requirements are satisfied by the specified `Software` (p. 290) `sw`, otherwise `false` is returned.

#### Parameters

`sw` is the `Software` (p. 290) which should satisfy the requirements.

#### Returns

`true` if requirements are satisfied, otherwise `false`.

#### See also

`isSatisfied(const std::list<Software>&) const` (p. 302),  
`isSatisfied(const std::list<ApplicationEnvironment>&) const` (p. 301),  
`selectSoftware(const Software&)` (p. 304),  
`isResolved() const` (p. 301).

References `isSatisfied()`.

Referenced by `isSatisfied()`.

### 6.261.3.11 `bool Arc::SoftwareRequirement::isSatisfied ( const std::list< Software > & swList ) const`

Test if requirements are satisfied.

Returns `true` if stored requirements are satisfied by software specified in `swList`, otherwise `false` is returned.

Note that if all requirements must be satisfied and multiple requirements exist having identical name and family all these requirements should be satisfied by a single **Software** (p. 290) object.

#### Parameters

`swList` is the list of **Software** (p. 290) objects which should be used to try satisfy the requirements.

#### Returns

`true` if requirements are satisfied, otherwise `false`.

#### See also

`isSatisfied(const Software&) const` (p. 302),  
`isSatisfied(const std::list<ApplicationEnvironment>&) const` (p. 301),  
`selectSoftware(const std::list<Software>&)` (p. 303),  
`isResolved() const` (p. 301).

#### 6.261.3.12 SoftwareRequirement& Arc::SoftwareRequirement::operator= ( const SoftwareRequirement & sr )

Assignment operator.

Set this object equal to that of the passed **SoftwareRequirement** (p. 297) object `sr`.

#### Parameters

`sr` is the **SoftwareRequirement** (p. 297) object to set object equal to.

#### 6.261.3.13 bool Arc::SoftwareRequirement::selectSoftware ( const std::list< Software > & swList )

Select software.

If the passed list of **Software** (p. 290) objects `swList` do not satisfy the requirements `false` is returned and this object is not modified. If however the list of **Software** (p. 290) objects `swList` do satisfy the requirements `true` is returned and the **Software** (p. 290) objects satisfying the requirements will replace these with the equality operator (**Software::operator==** (p. 295)) used as the comparator for the new requirements.

Note that if all requirements must be satisfied and multiple requirements exist having identical name and family all these requirements should be satisfied by a single **Software** (p. 290) object and it will replace all these requirements.

#### Parameters

`swList` is a list of **Software** (p. 290) objects used to satisfy requirements.

#### Returns

`true` if requirements are satisfied, otherwise `false`.

**See also**

**selectSoftware(const Software&)** (p. 304),  
**selectSoftware(const std::list<ApplicationEnvironment>&)** (p. 304),  
**isSatisfied(const std::list<Software>& const)** (p. 302),  
**isResolved() const** (p. 301).

#### 6.261.3.14 **bool Arc::SoftwareRequirement::selectSoftware ( const Software & *sw* ) [inline]**

Select software.

If the passed **Software** (p. 290) *sw* do not satisfy the requirements `false` is returned and this object is not modified. If however the **Software** (p. 290) object *sw* do satisfy the requirements `true` is returned and the requirements are set to equal the *sw* **Software** (p. 290) object.

**Parameters**

*sw* is the **Software** (p. 290) object used to satisfy requirements.

**Returns**

`true` if requirements are satisfied, otherwise `false`.

**See also**

**selectSoftware(const std::list<Software>&)** (p. 303),  
**selectSoftware(const std::list<ApplicationEnvironment>&)** (p. 304),  
**isSatisfied(const Software& const)** (p. 302),  
**isResolved() const** (p. 301).

References `selectSoftware()`.

Referenced by `selectSoftware()`.

#### 6.261.3.15 **bool Arc::SoftwareRequirement::selectSoftware ( const std::list<ApplicationEnvironment > & *swList* )**

Select software.

This method behaves exactly as the **selectSoftware(const std::list<Software>&)** (p. 303) method does.

**Parameters**

*swList* is a list of **ApplicationEnvironment** (p. 50) objects used to satisfy requirements.

**Returns**

`true` if requirements are satisfied, otherwise `false`.

**See also**

**selectSoftware(const Software&)** (p. 304),  
**selectSoftware(const std::list<Software>&)** (p. 303),  
**isSatisfied(const std::list<ApplicationEnvironment>& const)** (p. 301),  
**isResolved() const** (p. 301).

**6.261.3.16 void Arc::SoftwareRequirement::setRequirement ( bool *all* ) [inline]**

Set relation between requirements.

Specifies if all requirements stored need to be satisfied or if it is enough to satisfy only one of them.

**Parameters**

*all* is a boolean specifying if all requirements has to be satisfied.

**See also**

**isRequiringAll()** (p. 301).

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

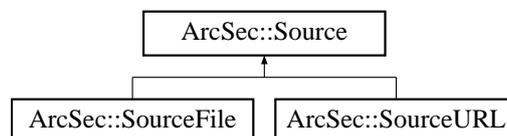
- Software.h

**6.262 ArcSec::Source Class Reference**

Acquires and parses XML document from specified source.

```
#include <Source.h>
```

Inheritance diagram for ArcSec::Source:

**Public Member Functions**

- **Source** (const **Source** &s)
- **Source** (Arc::XMLNode &xml)
- **Source** (std::istream &stream)
- **Source** (Arc::URL &url)
- **Source** (const std::string &str)
- **Arc::XMLNode Get** (void) const
- **operator bool** (void)

**6.262.1 Detailed Description**

Acquires and parses XML document from specified source. This class is to be used to provide easy way to specify different sources for XML Authorization Policies and Requests.

## 6.262.2 Constructor & Destructor Documentation

### 6.262.2.1 `ArcSec::Source::Source ( const Source & s ) [inline]`

Copy constructor.

Use this constructor only for temporary objects. Parsed XML document is still owned by copied source and hence lifetime of create object should not exceed that of copied one.

### 6.262.2.2 `ArcSec::Source::Source ( Arc::URL & url )`

Fetch XML document from specified url and parse it.

This constructor is not implemented yet.

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

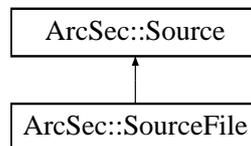
- Source.h

## 6.263 `ArcSec::SourceFile` Class Reference

Convenience class for obtaining XML document from file.

```
#include <Source.h>
```

Inheritance diagram for `ArcSec::SourceFile`:



### Public Member Functions

- `SourceFile (const SourceFile &s)`
- `SourceFile (const char *name)`
- `SourceFile (const std::string &name)`

### 6.263.1 Detailed Description

Convenience class for obtaining XML document from file.

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

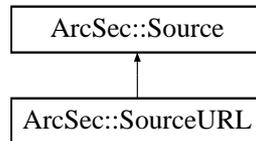
- Source.h

## 6.264 `ArcSec::SourceURL` Class Reference

Convenience class for obtaining XML document from remote URL.

```
#include <Source.h>
```

Inheritance diagram for ArcSec::SourceURL:



## Public Member Functions

- **SourceURL** (const **SourceURL** &s)
- **SourceURL** (const char \*url)
- **SourceURL** (const std::string &url)

### 6.264.1 Detailed Description

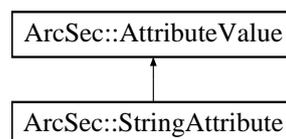
Convenience class for obtaining XML document from remote URL.

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

- Source.h

## 6.265 ArcSec::StringAttribute Class Reference

Inheritance diagram for ArcSec::StringAttribute:



## Public Member Functions

- virtual bool **equal** (**AttributeValue** \*other, bool check\_id=true)
- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.265.1 Member Function Documentation

#### 6.265.1.1 virtual std::string ArcSec::StringAttribute::encode ( ) [inline, virtual]

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 57).

### 6.265.1.2 virtual bool ArcSec::StringAttribute::equal ( AttributeValue \* value, bool check\_id = true ) [virtual]

Evaluate whether "this" equals to the parameter value

Implements ArcSec::AttributeValue (p. 58).

### 6.265.1.3 virtual std::string ArcSec::StringAttribute::getId ( ) [inline, virtual]

Get the AttributeId of the <Attribute>

Implements ArcSec::AttributeValue (p. 58).

### 6.265.1.4 virtual std::string ArcSec::StringAttribute::getType ( ) [inline, virtual]

Get the DataType of the <Attribute>

Implements ArcSec::AttributeValue (p. 58).

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

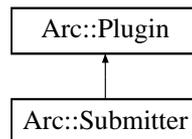
- StringAttribute.h

## 6.266 Arc::Submitter Class Reference

Base class for the Submitters.

```
#include <Submitter.h>
```

Inheritance diagram for Arc::Submitter:



### Public Member Functions

- virtual **URL Submit** (const **JobDescription** &jobdesc, const **ExecutionTarget** &et) const =0
- virtual **URL Migrate** (const **URL** &jobid, const **JobDescription** &jobdesc, const **ExecutionTarget** &et, bool forcemigration) const =0

#### 6.266.1 Detailed Description

Base class for the Submitters. **Submitter** (p. 308) is the base class for Grid middleware specialized **Submitter** (p. 308) objects. The class submits job(s) to the computing resource it represents and uploads (needed by the job) local input files.

## 6.266.2 Member Function Documentation

**6.266.2.1** virtual URL Arc::Submitter::Migrate ( const URL & *jobid*, const JobDescription & *jobdesc*, const ExecutionTarget & *et*, bool *forcemigration* ) const [pure virtual]

This virtual method should be overridden by plugins which should be capable of migrating jobs. The active job which should be migrated is pointed to by the **URL** (p. 323) *jobid*, and is represented by the **JobDescription** (p. 188) *jobdesc*. The *forcemigration* boolean specifies if the migration should succeed if the active job cannot be terminated. The protected method `AddJob` can be used to save job information. This method should return the **URL** (p. 323) of the migrated job. In case migration fails an empty **URL** (p. 323) should be returned.

**6.266.2.2** virtual URL Arc::Submitter::Submit ( const JobDescription & *jobdesc*, const ExecutionTarget & *et* ) const [pure virtual]

This virtual method should be overridden by plugins which should be capable of submitting jobs, defined in the **JobDescription** (p. 188) *jobdesc*, to the **ExecutionTarget** (p. 154) *et*. The protected convenience method `AddJob` can be used to save job information. This method should return the **URL** (p. 323) of the submitted job. In case submission fails an empty **URL** (p. 323) should be returned.

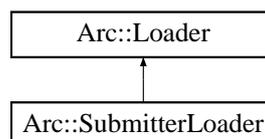
The documentation for this class was generated from the following file:

- Submitter.h

## 6.267 Arc::SubmitterLoader Class Reference

```
#include <Submitter.h>
```

Inheritance diagram for Arc::SubmitterLoader:



### Public Member Functions

- `SubmitterLoader ()`
- `~SubmitterLoader ()`
- `Submitter * load (const std::string &name, const UserConfig &usercfg)`
- `const std::list< Submitter * > & GetSubmitters () const`

### 6.267.1 Detailed Description

Class responsible for loading **Submitter** (p. 308) plugins. The **Submitter** (p. 308) objects returned by a **SubmitterLoader** (p. 309) must not be used after the **SubmitterLoader** (p. 309) goes out of scope.

## 6.267.2 Constructor & Destructor Documentation

### 6.267.2.1 Arc::SubmitterLoader::SubmitterLoader ( )

Constructor Creates a new **SubmitterLoader** (p. 309).

### 6.267.2.2 Arc::SubmitterLoader::~~SubmitterLoader ( )

Destructor Calling the destructor destroys all Submitters loaded by the **SubmitterLoader** (p. 309) instance.

## 6.267.3 Member Function Documentation

### 6.267.3.1 const std::list<Submitter\*> & Arc::SubmitterLoader::GetSubmitters ( ) const [inline]

Retrieve the list of loaded Submitters.

#### Returns

A reference to the list of Submitters.

### 6.267.3.2 Submitter\* Arc::SubmitterLoader::load ( const std::string & *name*, const UserConfig & *usercfg* )

Load a new **Submitter** (p. 308)

#### Parameters

*name* The name of the **Submitter** (p. 308) to load.

*usercfg* The **UserConfig** (p. 333) object for the new **Submitter** (p. 308).

#### Returns

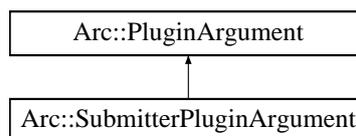
A pointer to the new **Submitter** (p. 308) (NULL on error).

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

- Submitter.h

## 6.268 Arc::SubmitterPluginArgument Class Reference

Inheritance diagram for Arc::SubmitterPluginArgument:



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

- Submitter.h

## 6.269 Arc::TargetGenerator Class Reference

Target generation class

```
#include <TargetGenerator.h>
```

### Public Member Functions

- **TargetGenerator** (const **UserConfig** &usercfg)
- void **GetTargets** (int targetType, int detailLevel)
- const std::list< **ExecutionTarget** > & **FoundTargets** () const
- std::list< **ExecutionTarget** > & **ModifyFoundTargets** ()
- const std::list< **XMLNode** \* > & **FoundJobs** () const
- bool **AddService** (const **URL** &url)
- bool **AddIndexServer** (const **URL** &url)
- void **AddTarget** (const **ExecutionTarget** &target)
- void **AddJob** (const **XMLNode** &job)
- void **PrintTargetInfo** (bool longlist) const
- **SimpleCounter** & **ServiceCounter** (void)

### 6.269.1 Detailed Description

Target generation class The **TargetGenerator** (p. 310) class is the umbrella class for resource discovery and information retrieval (index servers and computing clusters). It can also be used to locate user Grid jobs but does not collect the job details (see the arcsync CLI). The **TargetGenerator** (p. 310) loads **TargetRetriever** (p. 313) plugins (which implements the actual information retrieval) from **URL** (p. 323) objects found in the **UserConfig** (p. 333) object passed to its constructor using the custom **TargetRetrieverLoader** (p. 315). E.g. if an **URL** (p. 323) pointing to an ARC1 computing resource is found in the **UserConfig** (p. 333) object the **TargetRetrieverARC1** is loaded.

### 6.269.2 Constructor & Destructor Documentation

#### 6.269.2.1 Arc::TargetGenerator::TargetGenerator ( const UserConfig & usercfg )

Create a **TargetGenerator** (p. 310) object.

Default constructor to create a **TargetGenerator**. The constructor reads the computing and index service **URL** (p. 323) objects from the passed **UserConfig** (p. 333) object using the **UserConfig** (p. 333):**GetSelectedServices** method. From each **URL** (p. 323) a matching specialized **TargetRetriever** (p. 313) plugin is loaded using the **TargetRetrieverLoader** (p. 315).

#### Parameters

- usercfg* Reference to **UserConfig** (p. 333) object with **URL** (p. 323) objects to computing and/or index services and paths to user credentials.

## 6.269.3 Member Function Documentation

### 6.269.3.1 `bool Arc::TargetGenerator::AddIndexServer ( const URL & url )`

Add a new index server to the foundIndexServers list.

Method to add a new index server to the list of foundIndexServers in a thread secure way. Compares the argument **URL** (p. 323) against the servers returned by **UserConfig::GetRejectedServices** (p. 346) and only allows to add the service if not specifically rejected.

#### Parameters

*url* **URL** (p. 323) pointing to the index server.

### 6.269.3.2 `void Arc::TargetGenerator::AddJob ( const XMLNode & job )`

Add a new **Job** (p. 184) to the foundJobs list.

Method to add a new **Job** (p. 184) (usually discovered by a **TargetRetriever** (p. 313)) to the list of foundJobs in a thread secure way.

#### Parameters

*job* **XMLNode** (p. 392) describing the job.

### 6.269.3.3 `bool Arc::TargetGenerator::AddService ( const URL & url )`

Add a new computing service to the foundServices list.

Method to add a new service to the list of foundServices in a thread secure way. Compares the argument **URL** (p. 323) against the services returned by **UserConfig::GetRejectedServices** (p. 346) and only allows to add the service if not specifically rejected.

#### Parameters

*url* **URL** (p. 323) pointing to the information system of the computing service.

### 6.269.3.4 `void Arc::TargetGenerator::AddTarget ( const ExecutionTarget & target )`

Add a new **ExecutionTarget** (p. 154) to the foundTargets list.

Method to add a new **ExecutionTarget** (p. 154) (usually discovered by a **TargetRetriever** (p. 313)) to the list of foundTargets in a thread secure way.

#### Parameters

*target* **ExecutionTarget** (p. 154) to be added.

### 6.269.3.5 `const std::list<XMLNode*>& Arc::TargetGenerator::FoundJobs ( ) const`

Return Grid jobs found by GetTargets.

Method to return the list of Grid jobs found by a call to the GetTargets method.

**6.269.3.6 const std::list<ExecutionTarget>& Arc::TargetGenerator::FoundTargets ( ) const**

Return targets found by GetTargets.

Method to return a const list of **ExecutionTarget** (p. 154) objects (currently only supported Target type) found by the GetTarget method.

**6.269.3.7 void Arc::TargetGenerator::GetTargets ( int targetType, int detailLevel )**

Find available targets.

Method to prepare a list of chosen Targets with a specified detail level. Current implementation supports finding computing clusters (**ExecutionTarget** (p. 154)) with full detail level and Grid jobs with limited detail level.

**Parameters**

*targetType* 0 = **ExecutionTarget** (p. 154), 1 = Grid jobs

*detailLevel* 1 = All details, 2 = Limited details (not implemented)

**6.269.3.8 std::list<ExecutionTarget>& Arc::TargetGenerator::ModifyFoundTargets ( )**

Return targets found by GetTargets.

Method to return the list of **ExecutionTarget** (p. 154) objects (currently only supported Target type) found by the GetTarget method.

**6.269.3.9 void Arc::TargetGenerator::PrintTargetInfo ( bool longlist ) const**

Prints target information.

Method to print information of the found targets to std::cout.

**Parameters**

*longlist* false for minimal information, true for detailed information

**6.269.3.10 SimpleCounter& Arc::TargetGenerator::ServiceCounter ( void )**

Returns reference to worker counter.

This method returns reference to counter which keeps amount of started worker threads communicating with services asynchronously. The counter must be incremented for every thread started and decremented when thread exits. Main thread will then wait till counters drops to zero.

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

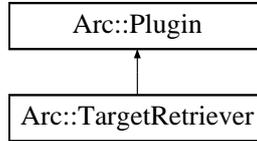
- TargetGenerator.h

## 6.270 Arc::TargetRetriever Class Reference

TargetRetriever base class

```
#include <TargetRetriever.h>
```

Inheritance diagram for Arc::TargetRetriever:



## Public Member Functions

- virtual void **GetTargets** (**TargetGenerator** &mom, int targetType, int detailLevel)=0

## Protected Member Functions

- **TargetRetriever** (const **UserConfig** &usercfg, const **URL** &url, ServiceType st, const std::string &flavour)

### 6.270.1 Detailed Description

TargetRetriever base class The **TargetRetriever** (p. 313) class is a pure virtual base class to be used for grid flavour specializations. It is designed to work in conjunction with the **TargetGenerator** (p. 310).

### 6.270.2 Constructor & Destructor Documentation

#### 6.270.2.1 Arc::TargetRetriever::TargetRetriever ( const UserConfig & usercfg, const URL & url, ServiceType st, const std::string & flavour ) [protected]

**TargetRetriever** (p. 313) constructor.

Default constructor to create a TargeGenerator. The constructor reads the computing and index service **URL** (p. 323) objects from the

#### Parameters

*usercfg*

*url*

*st*

*flavour*

### 6.270.3 Member Function Documentation

#### 6.270.3.1 virtual void Arc::TargetRetriever::GetTargets ( TargetGenerator & mom, int targetType, int detailLevel ) [pure virtual]

Method for collecting targets.

Pure virtual method for collecting targets. Implementation depends on the Grid middleware in question and is thus left to the specialized class.

**Parameters**

*mom* is the reference to the **TargetGenerator** (p. 310) which has loaded the **TargetRetriever** (p. 313)

*targetType* is the identificaion of targets to find (0=ExecutionTargets, 1=Grid Jobs)

*detailLevel* is the required level of details (1 = All details, 2 = Limited details)

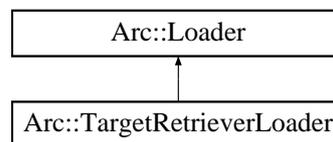
The documentation for this class was generated from the following file:

- TargetRetriever.h

**6.271 Arc::TargetRetrieverLoader Class Reference**

```
#include <TargetRetriever.h>
```

Inheritance diagram for Arc::TargetRetrieverLoader:

**Public Member Functions**

- **TargetRetrieverLoader** ()
- **~TargetRetrieverLoader** ()
- **TargetRetriever \* load** (const std::string &name, const **UserConfig** &usercfg, const **URL** &url, const ServiceType &st)
- const std::list< **TargetRetriever** \* > & **GetTargetRetrievers** () const

**6.271.1 Detailed Description**

Class responsible for loading **TargetRetriever** (p. 313) plugins The **TargetRetriever** (p. 313) objects returned by a **TargetRetrieverLoader** (p. 315) must not be used after the **TargetRetrieverLoader** (p. 315) goes out of scope.

**6.271.2 Constructor & Destructor Documentation****6.271.2.1 Arc::TargetRetrieverLoader::TargetRetrieverLoader ( )**

Constructor Creates a new **TargetRetrieverLoader** (p. 315).

**6.271.2.2 Arc::TargetRetrieverLoader::~~TargetRetrieverLoader ( )**

Destructor Calling the destructor destroys all **TargetRetrievers** loaded by the **TargetRetrieverLoader** (p. 315) instance.

### 6.271.3 Member Function Documentation

#### 6.271.3.1 `const std::list<TargetRetriever*> & Arc::TargetRetrieverLoader::GetTargetRetrievers ( ) const [inline]`

Retrieve the list of loaded TargetRetrievers.

#### Returns

A reference to the list of TargetRetrievers.

#### 6.271.3.2 `TargetRetriever* Arc::TargetRetrieverLoader::load ( const std::string & name, const UserConfig & usercfg, const URL & url, const ServiceType & st )`

Load a new **TargetRetriever** (p. 313)

#### Parameters

*name* The name of the **TargetRetriever** (p. 313) to load.

*usercfg* The **UserConfig** (p. 333) object for the new **TargetRetriever** (p. 313).

*url* The **URL** (p. 323) used to contact the target.

*st* specifies service type of the target.

#### Returns

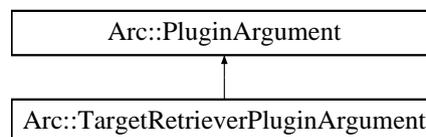
A pointer to the new **TargetRetriever** (p. 313) (NULL on error).

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

- TargetRetriever.h

## 6.272 Arc::TargetRetrieverPluginArgument Class Reference

Inheritance diagram for Arc::TargetRetrieverPluginArgument:

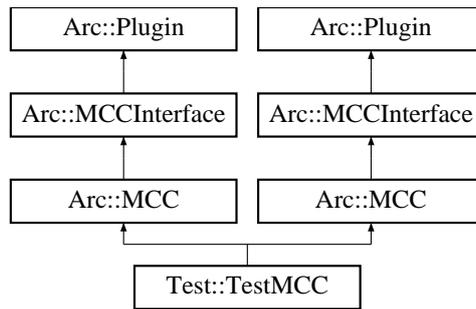


The documentation for this class was generated from the following file:

- TargetRetriever.h

## 6.273 Test::TestMCC Class Reference

Inheritance diagram for Test::TestMCC:

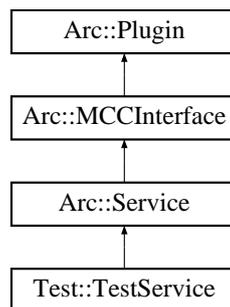


The documentation for this class was generated from the following files:

- loader/TestMCC.h
- message/TestMCC.h

## 6.274 Test::TestService Class Reference

Inheritance diagram for Test::TestService:



### Public Member Functions

- virtual `Arc::MCC_Status process (Arc::Message &request, Arc::Message &response)`

#### 6.274.1 Member Function Documentation

##### 6.274.1.1 virtual `Arc::MCC_Status Test::TestService::process ( Arc::Message & request, Arc::Message & response ) [virtual]`

Method for processing of requests and responses. This method is called by preceding MCC in chain when a request needs to be processed. This method must call similar method of next MCC in chain unless any failure happens. Result returned by call to next MCC should be processed and passed back to previous MCC. In case of failure this method is expected to generate valid error response and return it back to previous MCC without calling the next one.

#### Parameters

*request* The request that needs to be processed.

*response* A Message object that will contain the response of the request when the method returns.

### Returns

An object representing the status of the call.

Implements **Arc::MCCInterface** (p. 207).

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

- TestService.h

## 6.275 Arc::ThreadInitializer Class Reference

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

- Thread.h

## 6.276 Arc::ThreadRegistry Class Reference

```
#include <Thread.h>
```

### Public Member Functions

- void **RegisterThread** (void)
- void **UnregisterThread** (void)
- bool **WaitOrCancel** (int timeout)
- bool **WaitForExit** (int timeout=-1)

### 6.276.1 Detailed Description

This class is a set of conditions, mutexes, etc. conveniently exposed to monitor running child threads and to wait till they exit. There are no protections against race conditions. So use it carefully.

### 6.276.2 Member Function Documentation

#### 6.276.2.1 bool Arc::ThreadRegistry::WaitForExit ( int *timeout* = -1 )

Wait for registered threads to exit. Leave after timeout miliseconds if failed. Returns true if all registered threads reported their exit.

#### 6.276.2.2 bool Arc::ThreadRegistry::WaitOrCancel ( int *timeout* )

Wait for timeout milliseconds or cancel request. Returns true if cancel request received.

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

- Thread.h

## 6.277 Arc::Time Class Reference

A class for storing and manipulating times.

```
#include <DateTime.h>
```

### Public Member Functions

- **Time** ()
- **Time** (time\_t)
- **Time** (time\_t time, uint32\_t nanosec)
- **Time** (const std::string &)
- **Time & operator=** (time\_t)
- **Time & operator=** (const **Time** &)
- **Time & operator=** (const char \*)
- **Time & operator=** (const std::string &)
- void **SetTime** (time\_t)
- void **SetTime** (time\_t time, uint32\_t nanosec)
- time\_t **GetTime** () const
- **operator std::string** () const
- std::string **str** (const **TimeFormat** &=time\_format) const
- bool **operator<** (const **Time** &) const
- bool **operator>** (const **Time** &) const
- bool **operator<=** (const **Time** &) const
- bool **operator>=** (const **Time** &) const
- bool **operator==** (const **Time** &) const
- bool **operator!=** (const **Time** &) const
- **Time operator+** (const **Period** &) const
- **Time operator-** (const **Period** &) const
- **Period operator-** (const **Time** &) const

### Static Public Member Functions

- static void **SetFormat** (const **TimeFormat** &)
- static **TimeFormat GetFormat** ()

#### 6.277.1 Detailed Description

A class for storing and manipulating times.

#### 6.277.2 Constructor & Destructor Documentation

##### 6.277.2.1 Arc::Time::Time ( )

Default constructor. The time is put equal the current time.

##### 6.277.2.2 Arc::Time::Time ( time\_t )

Constructor that takes a time\_t variable and stores it.

**6.277.2.3 Arc::Time::Time ( time\_t time, uint32\_t nanosec )**

Constructor that takes a fine grained time variables and stores them.

**6.277.2.4 Arc::Time::Time ( const std::string & )**

Constructor that tries to convert a string into a time\_t.

**6.277.3 Member Function Documentation****6.277.3.1 static TimeFormat Arc::Time::GetFormat ( ) [static]**

Gets the default format for time strings.

**6.277.3.2 time\_t Arc::Time::GetTime ( ) const**

gets the time

**6.277.3.3 Arc::Time::operator std::string ( ) const**

Returns a string representation of the time, using the default format.

**6.277.3.4 bool Arc::Time::operator!= ( const Time & ) const**

Comparing two **Time** (p. 318) objects.

**6.277.3.5 Time Arc::Time::operator+ ( const Period & ) const**

Adding **Time** (p. 318) object with **Period** (p. 243) object.

**6.277.3.6 Time Arc::Time::operator- ( const Period & ) const**

Subtracting **Period** (p. 243) object from **Time** (p. 318) object.

**6.277.3.7 Period Arc::Time::operator- ( const Time & ) const**

Subtracting **Time** (p. 318) object from the other **Time** (p. 318) object.

**6.277.3.8 bool Arc::Time::operator< ( const Time & ) const**

Comparing two **Time** (p. 318) objects.

**6.277.3.9 bool Arc::Time::operator<= ( const Time & ) const**

Comparing two **Time** (p. 318) objects.

**6.277.3.10 Time& Arc::Time::operator= ( const char \* )**

Assignment operator from a char pointer.

**6.277.3.11 Time& Arc::Time::operator= ( const std::string & )**

Assignment operator from a string.

**6.277.3.12 Time& Arc::Time::operator= ( const Time & )**

Assignment operator from a **Time** (p. 318).

**6.277.3.13 Time& Arc::Time::operator= ( time\_t )**

Assignment operator from a time\_t.

**6.277.3.14 bool Arc::Time::operator== ( const Time & ) const**

Comparing two **Time** (p. 318) objects.

**6.277.3.15 bool Arc::Time::operator> ( const Time & ) const**

Comparing two **Time** (p. 318) objects.

**6.277.3.16 bool Arc::Time::operator>= ( const Time & ) const**

Comparing two **Time** (p. 318) objects.

**6.277.3.17 static void Arc::Time::SetFormat ( const TimeFormat & ) [static]**

Sets the default format for time strings.

**6.277.3.18 void Arc::Time::SetTime ( time\_t )**

sets the time

**6.277.3.19 void Arc::Time::SetTime ( time\_t time, uint32\_t nanosec )**

sets the fine grained time

**6.277.3.20 std::string Arc::Time::str ( const TimeFormat & = *time\_format* ) const**

Returns a string representation of the time, using the specified format.

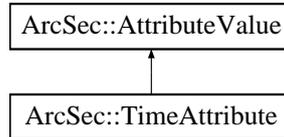
The documentation for this class was generated from the following file:

- DateTime.h

## 6.278 ArcSec::TimeAttribute Class Reference

```
#include <DateTimeAttribute.h>
```

Inheritance diagram for ArcSec::TimeAttribute:



### Public Member Functions

- virtual bool **equal** (**AttributeValue** \*other, bool check\_id=true)
- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

#### 6.278.1 Detailed Description

Format: HHMMSSZ HH:MM:SS HH:MM:SS+HH:MM HH:MM:SSZ

#### 6.278.2 Member Function Documentation

##### 6.278.2.1 virtual std::string ArcSec::TimeAttribute::encode ( ) [virtual]

encode the value in a string format

Implements **ArcSec::AttributeValue** (p. 57).

##### 6.278.2.2 virtual bool ArcSec::TimeAttribute::equal ( AttributeValue \* value, bool check\_id = true ) [virtual]

Evaluate whether "this" equal to the parameter value

Implements **ArcSec::AttributeValue** (p. 58).

##### 6.278.2.3 virtual std::string ArcSec::TimeAttribute::getId ( ) [inline, virtual]

Get the AttributeId of the <Attribute>

Implements **ArcSec::AttributeValue** (p. 58).

##### 6.278.2.4 virtual std::string ArcSec::TimeAttribute::getType ( ) [inline, virtual]

Get the DataType of the <Attribute>

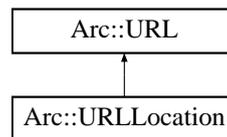
Implements **ArcSec::AttributeValue** (p. 58).

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

- DateTimeAttribute.h

## 6.279 Arc::URL Class Reference

Inheritance diagram for Arc::URL:



### Public Types

- enum **Scope**

### Public Member Functions

- **URL** ()
- **URL** (const std::string &url)
- virtual ~**URL** ()
- const std::string & **Protocol** () const
- void **ChangeProtocol** (const std::string &newprot)
- bool **IsSecureProtocol** () const
- const std::string & **Username** () const
- const std::string & **Passwd** () const
- const std::string & **Host** () const
- void **ChangeHost** (const std::string &newhost)
- int **Port** () const
- void **ChangePort** (int newport)
- const std::string & **Path** () const
- std::string **FullPath** () const
- void **ChangePath** (const std::string &newpath)
- const std::map< std::string, std::string > & **HTTPOptions** () const
- const std::string & **HTTPOption** (const std::string &option, const std::string &undefined="") const
- const std::list< std::string > & **LDAPAttributes** () const
- void **AddLDAPAttribute** (const std::string &attribute)
- **Scope** **LDAPScope** () const
- void **ChangeLDAPScope** (const **Scope** newscope)
- const std::string & **LDAPFilter** () const
- void **ChangeLDAPFilter** (const std::string &newfilter)
- const std::map< std::string, std::string > & **Options** () const
- const std::string & **Option** (const std::string &option, const std::string &undefined="") const
- const std::map< std::string, std::string > & **MetaDataOptions** () const
- const std::string & **MetaDataOption** (const std::string &option, const std::string &undefined="") const
- void **AddOption** (const std::string &option, const std::string &value, bool overwrite=true)
- void **AddMetaDataOption** (const std::string &option, const std::string &value, bool overwrite=true)

- `const std::list< URLLocation > & Locations () const`
- `const std::map< std::string, std::string > & CommonLocOptions () const`
- `const std::string & CommonLocOption (const std::string &option, const std::string &undefined="") const`
- `virtual std::string str () const`
- `virtual std::string plainstr () const`
- `virtual std::string fullstr () const`
- `virtual std::string ConnectionURL () const`
- `bool operator< (const URL &url) const`
- `bool operator== (const URL &url) const`
- `operator bool () const`
- `std::map< std::string, std::string > ParseOptions (const std::string &optstring, char separator)`

### Static Public Member Functions

- `static std::string OptionString (const std::map< std::string, std::string > &options, char separator)`

### Static Protected Member Functions

- `static std::string BaseDN2Path (const std::string &)`
- `static std::string Path2BaseDN (const std::string &)`

### Protected Attributes

- `std::string protocol`
- `std::string username`
- `std::string passwd`
- `std::string host`
- `bool ip6addr`
- `int port`
- `std::string path`
- `std::map< std::string, std::string > httpoptions`
- `std::map< std::string, std::string > metadataoptions`
- `std::list< std::string > ldattributes`
- `Scope ldapscope`
- `std::string ldapfilter`
- `std::map< std::string, std::string > urloptions`
- `std::list< URLLocation > locations`
- `std::map< std::string, std::string > commonlocoptions`
- `bool valid`

### Friends

- `std::ostream & operator<< (std::ostream &out, const URL &u)`

## 6.279.1 Member Enumeration Documentation

### 6.279.1.1 enum Arc::URL::Scope

Scope for LDAP URLs

## 6.279.2 Constructor & Destructor Documentation

### 6.279.2.1 Arc::URL::URL ( )

Empty constructor. Necessary when the class is part of another class and the like.

### 6.279.2.2 Arc::URL::URL ( const std::string & *url* )

Constructs a new URL (p. 323) from a string representation.

### 6.279.2.3 virtual Arc::URL::~~URL ( ) [virtual]

URL (p. 323) Destructor

## 6.279.3 Member Function Documentation

### 6.279.3.1 void Arc::URL::AddLDAPAttribute ( const std::string & *attribute* )

Adds an LDAP attribute.

### 6.279.3.2 void Arc::URL::AddMetaDataOption ( const std::string & *option*, const std::string & *value*, bool *overwrite* = *true* )

Adds a metadata option

### 6.279.3.3 void Arc::URL::AddOption ( const std::string & *option*, const std::string & *value*, bool *overwrite* = *true* )

Adds a URL (p. 323) option.

### 6.279.3.4 static std::string Arc::URL::BaseDN2Path ( const std::string & ) [static, protected]

a private method that converts an ldap basedn to a path.

### 6.279.3.5 void Arc::URL::ChangeHost ( const std::string & *newhost* )

Changes the hostname of the URL (p. 323).

**6.279.3.6 void Arc::URL::ChangeLDAPFilter ( const std::string & *newfilter* )**

Changes the LDAP filter.

**6.279.3.7 void Arc::URL::ChangeLDAPScope ( const Scope *newscope* )**

Changes the LDAP scope.

**6.279.3.8 void Arc::URL::ChangePath ( const std::string & *newpath* )**

Changes the path of the **URL** (p. 323).

**6.279.3.9 void Arc::URL::ChangePort ( int *newport* )**

Changes the port of the **URL** (p. 323).

**6.279.3.10 void Arc::URL::ChangeProtocol ( const std::string & *newprot* )**

Changes the protocol of the **URL** (p. 323).

**6.279.3.11 const std::string& Arc::URL::CommonLocOption ( const std::string & *option*, const std::string & *undefined* = "" ) const**

Returns the value of a common location option.

**Parameters**

*option* The option whose value is returned.

*undefined* This value is returned if the common location option is not defined.

**6.279.3.12 const std::map<std::string, std::string>& Arc::URL::CommonLocOptions ( ) const**

Returns the common location options if any.

**6.279.3.13 virtual std::string Arc::URL::ConnectionURL ( ) const [virtual]**

Returns a string representation with protocol, host and port only

**6.279.3.14 std::string Arc::URL::FullPath ( ) const**

Returns the path of the **URL** (p. 323) with all options attached.

**6.279.3.15 virtual std::string Arc::URL::fullstr ( ) const [virtual]**

Returns a string representation including options and locations

Reimplemented in **Arc::URLLocation** (p. 332).

**6.279.3.16** `const std::string& Arc::URL::Host ( ) const`

Returns the hostname of the **URL** (p. 323).

**6.279.3.17** `const std::string& Arc::URL::HTTPOption ( const std::string & option, const std::string & undefined = "" ) const`

Returns the value of an HTTP option.

**Parameters**

*option* The option whose value is returned.

*undefined* This value is returned if the HTTP option is not defined.

**6.279.3.18** `const std::map<std::string, std::string>& Arc::URL::HTTPOptions ( ) const`

Returns HTTP options if any.

**6.279.3.19** `bool Arc::URL::IsSecureProtocol ( ) const`

Indicates whether the protocol is secure or not.

**6.279.3.20** `const std::list<std::string>& Arc::URL::LDAPAttributes ( ) const`

Returns the LDAP attributes if any.

**6.279.3.21** `const std::string& Arc::URL::LDAPFilter ( ) const`

Returns the LDAP filter.

**6.279.3.22** `Scope Arc::URL::LDAPScope ( ) const`

Returns the LDAP scope.

**6.279.3.23** `const std::list<URLLocation>& Arc::URL::Locations ( ) const`

Returns the locations if any.

**6.279.3.24** `const std::string& Arc::URL::MetaDataOption ( const std::string & option, const std::string & undefined = "" ) const`

Returns the value of a metadata option.

**Parameters**

*option* The option whose value is returned.

*undefined* This value is returned if the metadata option is not defined.

**6.279.3.25** `const std::map<std::string, std::string>& Arc::URL::MetaDataOptions ( ) const`

Returns metadata options if any.

**6.279.3.26** `Arc::URL::operator bool ( ) const`

Check if instance holds valid **URL** (p. 323)

**6.279.3.27** `bool Arc::URL::operator< ( const URL & url ) const`

Compares one **URL** (p. 323) to another

**6.279.3.28** `bool Arc::URL::operator==( const URL & url ) const`

Is one **URL** (p. 323) equal to another?

**6.279.3.29** `const std::string& Arc::URL::Option ( const std::string & option, const std::string & undefined = "" ) const`

Returns the value of a **URL** (p. 323) option.

#### Parameters

*option* The option whose value is returned.

*undefined* This value is returned if the **URL** (p. 323) option is not defined.

**6.279.3.30** `const std::map<std::string, std::string>& Arc::URL::Options ( ) const`

Returns **URL** (p. 323) options if any.

**6.279.3.31** `static std::string Arc::URL::OptionString ( const std::map< std::string, std::string > & options, char separator ) [static]`

Returns a string representation of the options given in the options map

**6.279.3.32** `std::map<std::string, std::string> Arc::URL::ParseOptions ( const std::string & optstring, char separator )`

Parse a string of options separated by separator into an attribute->value map

**6.279.3.33** `const std::string& Arc::URL::Passwd ( ) const`

Returns the password of the **URL** (p. 323).

**6.279.3.34** `const std::string& Arc::URL::Path ( ) const`

Returns the path of the **URL** (p. 323).

**6.279.3.35** `static std::string Arc::URL::Path2BaseDN ( const std::string & ) [static, protected]`

a private method that converts an ldap path to a basedn.

**6.279.3.36** `virtual std::string Arc::URL::plainstr ( ) const [virtual]`

Returns a string representation of the **URL** (p. 323) without any options

**6.279.3.37** `int Arc::URL::Port ( ) const`

Returns the port of the **URL** (p. 323).

**6.279.3.38** `const std::string& Arc::URL::Protocol ( ) const`

Returns the protocol of the **URL** (p. 323).

**6.279.3.39** `virtual std::string Arc::URL::str ( ) const [virtual]`

Returns a string representation of the **URL** (p. 323) including meta-options.

Reimplemented in **Arc::URLLocation** (p. 332).

**6.279.3.40** `const std::string& Arc::URL::Username ( ) const`

Returns the username of the **URL** (p. 323).

## 6.279.4 Friends And Related Function Documentation

**6.279.4.1** `std::ostream& operator<< ( std::ostream & out, const URL & u ) [friend]`

Overloaded operator << to print a **URL** (p. 323).

## 6.279.5 Field Documentation

**6.279.5.1** `std::map<std::string, std::string> Arc::URL::commonlocoptions [protected]`

common location options for index server URLs.

**6.279.5.2** `std::string Arc::URL::host [protected]`

hostname of the url.

**6.279.5.3** `std::map<std::string, std::string> Arc::URL::httpoptions [protected]`

HTTP options of the url.

**6.279.5.4 bool Arc::URL::ip6addr [protected]**

if host is IPv6 numerical address notation.

**6.279.5.5 std::list<std::string> Arc::URL::ldapattributes [protected]**

LDAP attributes of the url.

**6.279.5.6 std::string Arc::URL::ldapfilter [protected]**

LDAP filter of the url.

**6.279.5.7 Scope Arc::URL::ldapscope [protected]**

LDAP scope of the url.

**6.279.5.8 std::list<URLLocation> Arc::URL::locations [protected]**

locations for index server URLs.

**6.279.5.9 std::map<std::string, std::string> Arc::URL::metadataoptions [protected]**

Meta data options

**6.279.5.10 std::string Arc::URL::passwd [protected]**

password of the url.

**6.279.5.11 std::string Arc::URL::path [protected]**

the url path.

**6.279.5.12 int Arc::URL::port [protected]**

portnumber of the url.

**6.279.5.13 std::string Arc::URL::protocol [protected]**

the url protocol.

**6.279.5.14 std::map<std::string, std::string> Arc::URL::urloptions [protected]**

options of the url.

**6.279.5.15** `std::string Arc::URL::username` [protected]

username of the url.

**6.279.5.16** `bool Arc::URL::valid` [protected]

flag to describe validity of **URL** (p. 323)

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

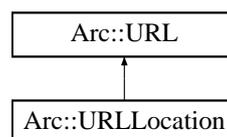
- **URL.h**

**6.280 Arc::URLLocation Class Reference**

Class to hold a resolved **URL** (p. 323) location.

```
#include <URL.h>
```

Inheritance diagram for Arc::URLLocation:

**Public Member Functions**

- **URLLocation** (const std::string &url)
- **URLLocation** (const std::string &url, const std::string &name)
- **URLLocation** (const **URL** &url)
- **URLLocation** (const **URL** &url, const std::string &name)
- **URLLocation** (const std::map< std::string, std::string > &options, const std::string &name)
- virtual ~**URLLocation** ()
- const std::string & **Name** () const
- virtual std::string **str** () const
- virtual std::string **fullstr** () const

**Protected Attributes**

- std::string **name**

**6.280.1 Detailed Description**

Class to hold a resolved **URL** (p. 323) location. It is specific to file indexing service registrations.

## 6.280.2 Constructor & Destructor Documentation

### 6.280.2.1 `Arc::URLLocation::URLLocation ( const std::string & url )`

Creates a `URLLocation` (p. 331) from a string representaion.

### 6.280.2.2 `Arc::URLLocation::URLLocation ( const std::string & url, const std::string & name )`

Creates a `URLLocation` (p. 331) from a string representaion and a name.

### 6.280.2.3 `Arc::URLLocation::URLLocation ( const URL & url )`

Creates a `URLLocation` (p. 331) from a `URL` (p. 323).

### 6.280.2.4 `Arc::URLLocation::URLLocation ( const URL & url, const std::string & name )`

Creates a `URLLocation` (p. 331) from a `URL` (p. 323) and a name.

### 6.280.2.5 `Arc::URLLocation::URLLocation ( const std::map< std::string, std::string > & options, const std::string & name )`

Creates a `URLLocation` (p. 331) from options and a name.

### 6.280.2.6 `virtual Arc::URLLocation::~~URLLocation ( ) [virtual]`

`URLLocation` (p. 331) destructor.

## 6.280.3 Member Function Documentation

### 6.280.3.1 `virtual std::string Arc::URLLocation::fullstr ( ) const [virtual]`

Returns a string representation including options and locations

Reimplemented from `Arc::URL` (p. 326).

### 6.280.3.2 `const std::string& Arc::URLLocation::Name ( ) const`

Returns the `URLLocation` (p. 331) name.

### 6.280.3.3 `virtual std::string Arc::URLLocation::str ( ) const [virtual]`

Returns a string representation of the `URLLocation` (p. 331).

Reimplemented from `Arc::URL` (p. 329).

## 6.280.4 Field Documentation

### 6.280.4.1 std::string Arc::URLLocation::name [protected]

the `URLLocation` (p. 331) name as registered in the indexing service.

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

- `URL.h`

## 6.281 Arc::URLMap Class Reference

### Data Structures

- class `map_entry`

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

- `URLMap.h`

## 6.282 Arc::User Class Reference

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

- `User.h`

## 6.283 Arc::UserConfig Class Reference

User configuration class

```
#include <UserConfig.h>
```

### Public Member Functions

- `UserConfig` (`initializeCredentialsType` initializeCredentials=`initializeCredentialsType`())
- `UserConfig` (const std::string &conffile, `initializeCredentialsType` initializeCredentials=`initializeCredentialsType`(), bool loadSysConfig=true)
- `UserConfig` (const std::string &conffile, const std::string &jfile, `initializeCredentialsType` initializeCredentials=`initializeCredentialsType`(), bool loadSysConfig=true)
- `UserConfig` (const long int &ptraddr)
- void `InitializeCredentials` ()
- bool `CredentialsFound` () const
- bool `LoadConfigurationFile` (const std::string &conffile, bool ignoreJobListFile=true)
- bool `SaveToFile` (const std::string &filename) const
- void `ApplyToConfig` (`BaseConfig` &ccfg) const
- `operator bool` () const
- bool `operator!` () const
- bool `JobListFile` (const std::string &path)

- const std::string & **JobListFile** () const
- bool **AddServices** (const std::list< std::string > &services, ServiceType st)
- bool **AddServices** (const std::list< std::string > &selected, const std::list< std::string > &rejected, ServiceType st)
- const URLListMap & **GetSelectedServices** (ServiceType st) const
- const URLListMap & **GetRejectedServices** (ServiceType st) const
- void **ClearSelectedServices** ()
- void **ClearSelectedServices** (ServiceType st)
- void **ClearRejectedServices** ()
- void **ClearRejectedServices** (ServiceType st)
- bool **Timeout** (int newTimeout)
- int **Timeout** () const
- bool **Verbosity** (const std::string &newVerbosity)
- const std::string & **Verbosity** () const
- bool **Broker** (const std::string &name)
- bool **Broker** (const std::string &name, const std::string &argument)
- const std::pair< std::string, std::string > & **Broker** () const
- bool **Bartender** (const std::vector< URL > &urls)
- void **AddBartender** (const URL &url)
- const std::vector< URL > & **Bartender** () const
- bool **VOMSServerPath** (const std::string &path)
- const std::string & **VOMSServerPath** () const
- bool **UserName** (const std::string &name)
- const std::string & **UserName** () const
- bool **Password** (const std::string &newPassword)
- const std::string & **Password** () const
- bool **ProxyPath** (const std::string &newProxyPath)
- const std::string & **ProxyPath** () const
- bool **CertificatePath** (const std::string &newCertificatePath)
- const std::string & **CertificatePath** () const
- bool **KeyPath** (const std::string &newKeyPath)
- const std::string & **KeyPath** () const
- bool **KeyPassword** (const std::string &newKeyPassword)
- const std::string & **KeyPassword** () const
- bool **KeySize** (int newKeySize)
- int **KeySize** () const
- bool **CACertificatePath** (const std::string &newCACertificatePath)
- const std::string & **CACertificatePath** () const
- bool **CACertificatesDirectory** (const std::string &newCACertificatesDirectory)
- const std::string & **CACertificatesDirectory** () const
- bool **CertificateLifeTime** (const Period &newCertificateLifeTime)
- const Period & **CertificateLifeTime** () const
- bool **SLCS** (const URL &newSLCS)
- const URL & **SLCS** () const
- bool **StoreDirectory** (const std::string &newStoreDirectory)
- const std::string & **StoreDirectory** () const
- bool **IdPName** (const std::string &name)
- const std::string & **IdPName** () const
- bool **OverlayFile** (const std::string &path)
- const std::string & **OverlayFile** () const
- bool **UtilsDirPath** (const std::string &dir)
- const std::string & **UtilsDirPath** () const

## Static Public Attributes

- static const std::string **ARCUSERDIRECTORY**
- static const std::string **SYSCONFIG**
- static const std::string **SYSCONFIGARCLOC**
- static const std::string **DEFAULTCONFIG**
- static const std::string **EXAMPLECONFIG**
- static const int **DEFAULT\_TIMEOUT** = 20
- static const std::string **DEFAULT\_BROKER**

### 6.283.1 Detailed Description

User configuration class This class provides a container for a selection of various attributes/parameters which can be configured to needs of the user, and can be read by implementing instances or programs. The class can be used in two ways. One can create a object from a configuration file, or simply set the desired attributes by using the setter method, associated with every setable attribute. The list of attributes which can be configured in this class are:

- certificatepath / **CertificatePath(const std::string&)** (p. 343)
- keypath / **KeyPath(const std::string&)** (p. 349)
- proxypath / **ProxyPath(const std::string&)** (p. 354)
- cacertificatesdirectory / **CACertificatesDirectory(const std::string&)** (p. 342)
- cacertificatepath / **CACertificatePath(const std::string&)** (p. 341)
- timeout / **Timeout(int)** (p. 356)
- joblist / **JobListFile(const std::string&)** (p. 348)
- defaultservices / **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 338)
- rejectservices / **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 338)
- verbosity / **Verbosity(const std::string&)** (p. 358)
- brokername / **Broker(const std::string&)** (p. 340) or **Broker(const std::string&, const std::string&)** (p. 341)
- brokerarguments / **Broker(const std::string&)** (p. 340) or **Broker(const std::string&, const std::string&)** (p. 341)
- bartender / **Bartender(const std::list<URL>&)**
- vomsserverpath / **VOMSServerPath(const std::string&)** (p. 358)
- username / **UserName(const std::string&)** (p. 357)
- password / **Password(const std::string&)** (p. 353)
- keypassword / **KeyPassword(const std::string&)** (p. 349)
- keysize / **KeySize(int)** (p. 351)
- certificatelifetime / **CertificateLifeTime(const Period&)** (p. 343)

- `slcs` / `SLCS(const URL&)` (p. 355)
- `storedirectory` / `StoreDirectory(const std::string&)` (p. 355)
- `idpname` / `IdPName(const std::string&)` (p. 346)

where the first term is the name of the attribute used in the configuration file, and the second term is the associated setter method (for more information about a given attribute see the description of the setter method).

The configuration file should have a INI-style format and the `IniConfig` (p. 178) class will thus be used to parse the file. The above mentioned attributes should be placed in the common section. Another section is also valid in the configuration file, which is the alias section. Here it is possible to define aliases representing one or multiple services. These aliases can be used in the `AddServices(const std::list<std::string>&, ServiceType)` (p. 338) and `AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)` (p. 338) methods.

The `UserConfig` (p. 333) class also provides a method `InitializeCredentials()` (p. 347) for locating user credentials by searching in different standard locations. The `CredentialsFound()` (p. 345) method can be used to test if locating the credentials succeeded.

## 6.283.2 Constructor & Destructor Documentation

### 6.283.2.1 `Arc::UserConfig::UserConfig ( initializeCredentialsType initializeCredentials = initializeCredentialsType () )`

Create a `UserConfig` (p. 333) object.

The `UserConfig` (p. 333) object created by this constructor initializes only default values, and if specified by the `initializeCredentials` boolean credentials will be tried initialized using the `InitializeCredentials()` (p. 347) method. The object is only non-valid if initialization of credentials fails which can be checked with the `operator bool()` (p. 352) method.

#### Parameters

`initializeCredentials` is a optional boolean indicating if the `InitializeCredentials()` (p. 347) method should be invoked, the default is `true`.

#### See also

`InitializeCredentials()` (p. 347)  
`operator bool()` (p. 352)

### 6.283.2.2 `Arc::UserConfig::UserConfig ( const std::string & configfile, initializeCredentialsType initializeCredentials = initializeCredentialsType (), bool loadSysConfig = true )`

Create a `UserConfig` (p. 333) object.

The `UserConfig` (p. 333) object created by this constructor will, if specified by the `loadSysConfig` boolean, first try to load the system configuration file by invoking the `LoadConfigurationFile()` (p. 351) method, and if this fails a WARNING is reported. Then the configuration file passed will be tried loaded using the before mentioned method, and if this fails an ERROR is reported, and the created object will be non-valid. Note that if the passed file path is empty the example configuration will be tried copied to the default configuration file path specified by `DEFAULTCONFIG`. If the example file cannot be copied one or more WARNING messages will be reported and no configuration will be loaded. If loading the configurations file

succeeded and if *initializeCredentials* is `true` then credentials will be initialized using the **InitializeCredentials()** (p. 347) method, and if no valid credentials are found the created object will be non-valid.

#### Parameters

*confFile* is the path to a INI-configuration file.

*initializeCredentials* is a boolean indicating if credentials should be initialized, the default is `true`.

*loadSysConfig* is a boolean indicating if the system configuration file should be loaded aswell, the default is `true`.

#### See also

**LoadConfigurationFile(const std::string&, bool)** (p. 351)

**InitializeCredentials()** (p. 347)

**operator bool()** (p. 352)

**SYSCONFIG** (p. 360)

**EXAMPLECONFIG** (p. 360)

#### 6.283.2.3 Arc::UserConfig::UserConfig ( const std::string & *confFile*, const std::string & *jfile*, initializeCredentialsType *initializeCredentials* = initializeCredentialsType (), bool *loadSysConfig* = `true` )

Create a **UserConfig** (p. 333) object.

The **UserConfig** (p. 333) object created by this constructor does only differ from the **UserConfig(const std::string&, bool, bool)** constructor in that it is possible to pass the path of the job list file directly to this constructor. If the job list file *joblistfile* is empty, the behaviour of this constructor is exactly the same as the before mentioned, otherwise the job list file will be initialized by invoking the setter method **JobListFile(const std::string&)** (p. 348). If it fails the created object will be non-valid, otherwise the specified configuration file *confFile* will be loaded with the *ignoreJobListFile* argument set to `true`.

#### Parameters

*confFile* is the path to a INI-configuration file

*jfile* is the path to a (non-)existing job list file.

*initializeCredentials* is a boolean indicating if credentials should be initialized, the default is `true`.

*loadSysConfig* is a boolean indicating if the system configuration file should be loaded aswell, the default is `true`.

#### See also

**JobListFile(const std::string&)** (p. 348)

**LoadConfigurationFile(const std::string&, bool)** (p. 351)

**InitializeCredentials()** (p. 347)

**operator bool()** (p. 352)

#### 6.283.2.4 Arc::UserConfig::UserConfig ( const long int & *ptraddr* )

Language binding constructor.

The passed long int should be a pointer address to a **UserConfig** (p. 333) object, and this address is then casted into this **UserConfig** (p. 333) object.

#### Parameters

*ptraddr* is an memory address to a **UserConfig** (p. 333) object.

### 6.283.3 Member Function Documentation

#### 6.283.3.1 void Arc::UserConfig::AddBartender ( const URL & *url* ) [inline]

Set bartenders, used to contact Chelonia.

Takes as input a Bartender **URL** (p. 323) and adds this to the list of bartenders.

##### Parameters

*url* is a **URL** (p. 323) to be added to the list of bartenders.

##### See also

Bartender(const std::list<URL>&)

Bartender() const (p. 340)

#### 6.283.3.2 bool Arc::UserConfig::AddServices ( const std::list< std::string > & *selected*, const std::list< std::string > & *rejected*, ServiceType *st* )

Add selected and rejected services.

The only difference in behaviour of this method compared to the **AddServices(const std::list<std::string>&, ServiceType)** (p. 338) method is the input parameters and the format these parameters should follow. Instead of having an optional '-' in front of the string selected and rejected services should be specified in the two different arguments.

Two attributes are indirectly associated with this setter method 'defaultservices' and 'rejectservices'. The values specified with the 'defaultservices' attribute will be added to the list of selected services, and likewise with the 'rejectservices' attribute.

##### Parameters

*selected* is a list of services which will be added to the selected services of this object.

*rejected* is a list of services which will be added to the rejected services of this object.

*st* specifies the ServiceType of the services to add.

##### Returns

This method return `false` in case an alias cannot be resolved. In any other case `true` is returned.

##### See also

**AddServices(const std::list<std::string>&, ServiceType)** (p. 338)

**GetSelectedServices()** (p. 346)

**GetRejectedServices()** (p. 346)

**ClearSelectedServices()** (p. 345)

**ClearRejectedServices()** (p. 344)

**LoadConfigurationFile()** (p. 351)

#### 6.283.3.3 bool Arc::UserConfig::AddServices ( const std::list< std::string > & *services*, ServiceType *st* )

Add selected and rejected services.

This method adds selected services and adds services to reject from the specified list *services*, which contains string objects. The syntax of a single element in the list must be expressed in the following two formats:

$$[-] < flavour > : < service\_url > | [-] < alias >$$

where the optional '-' indicate that the service should be added to the private list of services to reject. In the first format the <flavour> part indicates the type of ACC plugin to use when contacting the service, which is specified by the **URL** (p. 323) <service\_url>, and in the second format the <alias> part specifies a alias defined in a parsed configuration file, note that the alias must not contain any of the characters '.', ' ', '\t'. If a alias cannot be resolved an **ERROR** will be reported to the logger and the method will return false. If a element in the list *services* cannot be parsed an **ERROR** will be reported, and the element is skipped.

Two attributes are indirectly associated with this setter method 'defaultservices' and 'rejectservices'. The values specified with the 'defaultservices' attribute will be added to the list of selected services, and likewise with the 'rejectservices' attribute.

### Parameters

*services* is a list of services to either select or reject.

*st* indicates the type of the specified services.

### Returns

This method returns `false` in case an alias cannot be resolved. In any other case `true` is returned.

### See also

`AddServices(const std::string&, const std::string&, ServiceType)`

`GetSelectedServices()` (p. 346)

`GetRejectedServices()` (p. 346)

`ClearSelectedServices()` (p. 345)

`ClearRejectedServices()` (p. 344)

`LoadConfigurationFile()` (p. 351)

#### 6.283.3.4 void Arc::UserConfig::ApplyToConfig ( BaseConfig & ccfg ) const

Apply credentials to **BaseConfig** (p. 60).

This methods sets the **BaseConfig** (p. 60) credentials to the credentials contained in this object. It also passes user defined configuration overlay if any.

### See also

`InitializeCredentials()` (p. 347)

`CredentialsFound()` (p. 345)

**BaseConfig** (p. 60)

### Parameters

*ccfg* a **BaseConfig** (p. 60) object which will configured with the credentials of this object.

**6.283.3.5** `bool Arc::UserConfig::Bartender ( const std::vector< URL > & urls ) [inline]`

Set bartenders, used to contact Chelonia.

Takes as input a vector of Bartender URLs.

The attribute associated with this setter method is 'bartender'.

**Parameters**

*urls* is a list of **URL** (p. 323) object to be set as bartenders.

**Returns**

This method always returns `true`.

**See also**

**AddBartender(const URL&)** (p. 338)

**Bartender() const** (p. 340)

**6.283.3.6** `const std::vector<URL>& Arc::UserConfig::Bartender ( ) const [inline]`

Get bartenders.

Returns a list of Bartender URLs

**Returns**

The list of bartender **URL** (p. 323) objects is returned.

**See also**

**Bartender(const std::list<URL>&)**

**AddBartender(const URL&)** (p. 338)

**6.283.3.7** `bool Arc::UserConfig::Broker ( const std::string & name )`

Set broker to use in target matching.

The string passed to this method should be in the format:

$$\langle name \rangle [:\langle argument \rangle]$$

where the  $\langle name \rangle$  is the name of the broker and cannot contain any ':', and the optional  $\langle argument \rangle$  should contain arguments which should be passed to the broker.

Two attributes are associated with this setter method 'brokername' and 'brokerarguments'.

**Parameters**

*name* the broker name and argument specified in the format given above.

**Returns**

This method allways returns `true`.

**See also**

**Broker** (p. 62)  
**Broker(const std::string&, const std::string&)** (p. 341)  
**Broker() const** (p. 341)  
**DEFAULT\_BROKER** (p. 359)

**6.283.3.8 bool Arc::UserConfig::Broker ( const std::string & *name*, const std::string & *argument* ) [inline]**

Set broker to use in target matching.

As opposed to the **Broker(const std::string&)** (p. 340) method this method sets broker name and arguments directly from the passed two arguments.

Two attributes are associated with this setter method 'brokername' and 'brokerarguments'.

**Parameters**

*name* is the name of the broker.  
*argument* is the arguments of the broker.

**Returns**

This method always returns `true`.

**See also**

**Broker** (p. 62)  
**Broker(const std::string&)** (p. 340)  
**Broker() const** (p. 341)  
**DEFAULT\_BROKER** (p. 359)

**6.283.3.9 const std::pair<std::string, std::string>& Arc::UserConfig::Broker ( ) const [inline]**

Get the broker and corresponding arguments.

The returned pair contains the broker name as the first component and the argument as the second.

**See also**

**Broker(const std::string&)** (p. 340)  
**Broker(const std::string&, const std::string&)** (p. 341)  
**DEFAULT\_BROKER** (p. 359)

**6.283.3.10 bool Arc::UserConfig::CACertificatePath ( const std::string & *newCACertificatePath* ) [inline]**

Set CA-certificate path.

The path to the file containing CA-certificate will be set when calling this method. This configuration parameter is deprecated - use `CACertificatesDirectory` instead. Only `arcscls` uses it.

The attribute associated with this setter method is 'cacertificatepath'.

**Parameters**

*newCACertificatePath* is the path to the CA-certificate.

**Returns**

This method always returns `true`.

**See also**

**CACertificatePath() const** (p. 342)

**6.283.3.11 const std::string& Arc::UserConfig::CACertificatePath ( ) const [inline]**

Get path to CA-certificate.

Retrieve the path to the file containing CA-certificate. This configuration parameter is deprecated.

**Returns**

The path to the CA-certificate is returned.

**See also**

**CACertificatePath(const std::string&)** (p. 341)

**6.283.3.12 bool Arc::UserConfig::CACertificatesDirectory ( const std::string & newCACertificatesDirectory ) [inline]**

Set path to CA-certificate directory.

The path to the directory containing CA-certificates will be set when calling this method. Note that the **InitializeCredentials()** (p. 347) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'cacertificatesdirectory'.

**Parameters**

*newCACertificatesDirectory* is the path to the CA-certificate directory.

**Returns**

This method always returns `true`.

**See also**

**InitializeCredentials()** (p. 347)

**CredentialsFound() const** (p. 345)

**CACertificatesDirectory() const** (p. 342)

**6.283.3.13 const std::string& Arc::UserConfig::CACertificatesDirectory ( ) const [inline]**

Get path to CA-certificate directory.

Retrieve the path to the CA-certificate directory.

**Returns**

The path to the CA-certificate directory is returned.

**See also**

**InitializeCredentials()** (p. 347)  
**CredentialsFound() const** (p. 345)  
**CACertificatesDirectory(const std::string&)** (p. 342)

**6.283.3.14 bool Arc::UserConfig::CertificateLifeTime ( const Period & newCertificateLifeTime )  
[inline]**

Set certificate life time.

Sets lifetime of user certificate which will be obtained from Short Lived Credentials **Service** (p. 285).

The attribute associated with this setter method is 'certificatelifetime'.

**Parameters**

*newCertificateLifeTime* is the life time of a certificate, as a **Period** (p. 243) object.

**Returns**

This method always returns `true`.

**See also**

**CertificateLifeTime() const** (p. 343)

**6.283.3.15 const Period& Arc::UserConfig::CertificateLifeTime ( ) const [inline]**

Get certificate life time.

Gets lifetime of user certificate which will be obtained from Short Lived Credentials **Service** (p. 285).

**Returns**

The certificate life time is returned as a **Period** (p. 243) object.

**See also**

**CertificateLifeTime(const Period&)** (p. 343)

**6.283.3.16 bool Arc::UserConfig::CertificatePath ( const std::string & newCertificatePath )  
[inline]**

Set path to certificate.

The path to user certificate will be set by this method. The path to the corresponding key can be set with the **KeyPath(const std::string&)** (p. 349) method. Note that the **InitializeCredentials()** (p. 347) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'certificatepath'.

### Parameters

*newCertificatePath* is the path to the new certificate.

### Returns

This method always returns `true`.

### See also

**InitializeCredentials()** (p. 347)  
**CredentialsFound() const** (p. 345)  
**CertificatePath() const** (p. 344)  
**KeyPath(const std::string&)** (p. 349)

#### 6.283.3.17 `const std::string& Arc::UserConfig::CertificatePath ( ) const [inline]`

Get path to certificate.

The path to the cerficate is returned when invoking this method.

### Returns

The certificate path is returned.

### See also

**InitializeCredentials()** (p. 347)  
**CredentialsFound() const** (p. 345)  
**CertificatePath(const std::string&)** (p. 343)  
**KeyPath() const** (p. 350)

#### 6.283.3.18 `void Arc::UserConfig::ClearRejectedServices ( ServiceType st )`

Clear rejected services with specified `ServiceType`.

Calling this method will cause the internally stored rejected services with the `ServiceType st` to be cleared.

### See also

**ClearRejectedServices()** (p. 344)  
**ClearSelectedServices(ServiceType)** (p. 345)  
**AddServices(const std::list<std::string>&, ServiceType)** (p. 338)  
**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 338)  
**GetRejectedServices()** (p. 346)

#### 6.283.3.19 `void Arc::UserConfig::ClearRejectedServices ( )`

Clear selected services.

Calling this method will cause the internally stored rejected services to be cleared.

**See also**

**ClearRejectedServices(ServiceType)** (p. 344)  
**ClearSelectedServices()** (p. 345)  
**AddServices(const std::list<std::string>&, ServiceType)** (p. 338)  
**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 338)  
**GetRejectedServices()** (p. 346)

**6.283.3.20 void Arc::UserConfig::ClearSelectedServices ( )**

Clear selected services.

Calling this method will cause the internally stored selected services to be cleared.

**See also**

**ClearSelectedServices(ServiceType)** (p. 345)  
**ClearRejectedServices()** (p. 344)  
**AddServices(const std::list<std::string>&, ServiceType)** (p. 338)  
**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 338)  
**GetSelectedServices()** (p. 346)

**6.283.3.21 void Arc::UserConfig::ClearSelectedServices ( ServiceType st )**

Clear selected services with specified ServiceType.

Calling this method will cause the internally stored selected services with the ServiceType *st* to be cleared.

**See also**

**ClearSelectedServices()** (p. 345)  
**ClearRejectedServices(ServiceType)** (p. 344)  
**AddServices(const std::list<std::string>&, ServiceType)** (p. 338)  
**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 338)  
**GetSelectedServices()** (p. 346)

**6.283.3.22 bool Arc::UserConfig::CredentialsFound ( ) const [inline]**

Validate credential location.

Valid credentials consists of a combination of a path to existing CA-certificate directory and either a path to existing proxy or a path to existing user key/certificate pair. If valid credentials are found this method returns `true`, otherwise `false` is returned.

**Returns**

`true` if valid credentials are found, otherwise `false`.

**See also**

**InitializeCredentials()** (p. 347)

**6.283.3.23** `const URLListMap& Arc::UserConfig::GetRejectedServices ( ServiceType st ) const`

Get rejected services.

Get the rejected services with the ServiceType specified by *st*.

**Parameters**

*st* specifies which ServiceType should be returned by the method.

**Returns**

The rejected services is returned.

**See also**

`AddServices(const std::list<std::string>&, ServiceType)` (p. 338)

`AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)` (p. 338)

`GetSelectedServices(ServiceType)`

`ClearRejectedServices()` (p. 344)

**6.283.3.24** `const URLListMap& Arc::UserConfig::GetSelectedServices ( ServiceType st ) const`

Get selected services.

Get the selected services with the ServiceType specified by *st*.

**Parameters**

*st* specifies which ServiceType should be returned by the method.

**Returns**

The selected services is returned.

**See also**

`AddServices(const std::list<std::string>&, ServiceType)` (p. 338)

`AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)` (p. 338)

`GetRejectedServices(ServiceType) const` (p. 346)

`ClearSelectedServices()` (p. 345)

**6.283.3.25** `bool Arc::UserConfig::IdPName ( const std::string & name ) [inline]`

Set IdP name.

Sets Identity Provider name (Shibboleth) to which user belongs. It is used for contacting Short Lived Certificate **Service** (p. 285).

The attribute associated with this setter method is 'idpname'.

**Parameters**

*name* is the new IdP name.

**Returns**

This method always returns `true`.

See also

#### 6.283.3.26 `const std::string& Arc::UserConfig::IdPName ( ) const [inline]`

Get IdP name.

Gets Identity Provider name (Shibboleth) to which user belongs.

#### Returns

The IdP name

See also

`IdPName(const std::string&)` (p. 346)

#### 6.283.3.27 `void Arc::UserConfig::InitializeCredentials ( )`

Initialize user credentials.

The location of the user credentials will be tried located when calling this method and stored internally when found. The method searches in different locations. First the user proxy or the user key/certificate pair is tried located in the following order:

- Proxy path specified by the environment variable `X509_USER_PROXY`
- Key/certificate path specified by the environment `X509_USER_KEY` and `X509_USER_CERT`
- Proxy path specified in either configuration file passed to the constructor or explicitly set using the setter method `ProxyPath(const std::string&)` (p. 354)
- Key/certificate path specified in either configuration file passed to the constructor or explicitly set using the setter methods `KeyPath(const std::string&)` (p. 349) and `CertificatePath(const std::string&)` (p. 343)
- `ProxyPath` with file name `x509up_u` concatenated with the user ID located in the OS temporary directory.

If the proxy or key/certificate pair have been explicitly specified only the specified path(s) will be tried, and if not found a `ERROR` is reported. If the proxy or key/certificate have not been specified and it is not located in the temporary directory a `WARNING` will be reported and the host key/certificate pair is tried and then the Globus key/certificate pair and a `ERROR` will be reported if not found in any of these locations.

Together with the proxy and key/certificate pair, the path to the directory containing CA certificates is also tried located when invoking this method. The directory will be tried located in the following order:

- Path specified by the `X509_CERT_DIR` environment variable.
- Path explicitly specified either in a parsed configuration file using the `cacertificateirectory` or by using the setter method `CACertificatesDirectory()` (p. 342).
- Path created by concatenating the output of `User::Home()` with `'globus'` and `'certificates'` separated by the directory delimiter.

- Path created by concatenating the output of `Glib::get_home_dir()` with `'globus'` and `'certificates'` separated by the directory delimiter.
- Path created by concatenating the output of `ArcLocation::Get()` (p. 51), with `'etc'` and `'certificates'` separated by the directory delimiter.
- Path created by concatenating the output of `ArcLocation::Get()` (p. 51), with `'etc'`, `'grid-security'` and `'certificates'` separated by the directory delimiter.
- Path created by concatenating the output of `ArcLocation::Get()` (p. 51), with `'share'` and `'certificates'` separated by the directory delimiter.
- Path created by concatenating `'etc'`, `'grid-security'` and `'certificates'` separated by the directory delimiter.

If the CA certificate directory have explicitly been specified and the directory does not exist a `ERROR` is reported. If none of the directories above does not exist a `ERROR` is reported.

#### See also

`CredentialsFound()` (p. 345)  
`ProxyPath(const std::string&)` (p. 354)  
`KeyPath(const std::string&)` (p. 349)  
`CertificatePath(const std::string&)` (p. 343)  
`CACertificatesDirectory(const std::string&)` (p. 342)

### 6.283.3.28 `bool Arc::UserConfig::JobListFile ( const std::string & path )`

Set path to job list file.

The method takes a path to a file which will be used as the job list file for storing and reading job information. If the specified path *path* does not exist a empty job list file will be tried created. If creating the job list file in any way fails *false* will be returned and a `ERROR` message will be reported. Otherwise *true* is returned. If the directory containing the file does not exist, it will be tried created. The method will also return *false* if the file is not a regular file.

The attribute associated with this setter method is `'joblist'`.

#### Parameters

*path* the path to the job list file.

#### Returns

If the job list file is a regular file or if it can be created *true* is returned, otherwise *false* is returned.

#### See also

`JobListFile() const` (p. 348)

### 6.283.3.29 `const std::string& Arc::UserConfig::JobListFile ( ) const [inline]`

Get a reference to the path of the job list file.

The job list file is used to store and fetch information about submitted computing jobs to computing services. This method will return the path to the specified job list file.

**Returns**

The path to the job list file is returned.

**See also**

**JobListFile(const std::string&)** (p. 348)

**6.283.3.30 bool Arc::UserConfig::KeyPassword ( const std::string & newKeyPassword ) [inline]**

Set password for generated key.

Set password to be used to encode private key of credentials obtained from Short Lived Credentials **Service** (p. 285).

The attribute associated with this setter method is 'keypassword'.

**Parameters**

*newKeyPassword* is the new password to the key.

**Returns**

This method always returns `true`.

**See also**

**KeyPassword() const** (p. 349)

**KeyPath(const std::string&)** (p. 349)

**KeySize(int)** (p. 351)

**6.283.3.31 const std::string& Arc::UserConfig::KeyPassword ( ) const [inline]**

Get password for generated key.

Get password to be used to encode private key of credentials obtained from Short Lived Credentials **Service** (p. 285).

**Returns**

The key password is returned.

**See also**

**KeyPassword(const std::string&)** (p. 349)

**KeyPath() const** (p. 350)

**KeySize() const** (p. 350)

**6.283.3.32 bool Arc::UserConfig::KeyPath ( const std::string & newKeyPath ) [inline]**

Set path to key.

The path to user key will be set by this method. The path to the corresponding certificate can be set with the **CertificatePath(const std::string&)** (p. 343) method. Note that the **InitializeCredentials()** (p. 347) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'keypath'.

**Parameters**

*newKeyPath* is the path to the new key.

**Returns**

This method always returns `true`.

**See also**

**InitializeCredentials()** (p. 347)  
**CredentialsFound() const** (p. 345)  
**KeyPath() const** (p. 350)  
**CertificatePath(const std::string&)** (p. 343)  
**KeyPassword(const std::string&)** (p. 349)  
**KeySize(int)** (p. 351)

**6.283.3.33 const std::string& Arc::UserConfig::KeyPath ( ) const [inline]**

Get path to key.

The path to the key is returned when invoking this method.

**Returns**

The path to the user key is returned.

**See also**

**InitializeCredentials()** (p. 347)  
**CredentialsFound() const** (p. 345)  
**KeyPath(const std::string&)** (p. 349)  
**CertificatePath() const** (p. 344)  
**KeyPassword() const** (p. 349)  
**KeySize() const** (p. 350)

**6.283.3.34 int Arc::UserConfig::KeySize ( ) const [inline]**

Get key size.

Get size/strengt of private key of credentials obtained from Short Lived Credentials **Service** (p. 285).

**Returns**

The key size, as an integer, is returned.

**See also**

**KeySize(int)** (p. 351)  
**KeyPath() const** (p. 350)  
**KeyPassword() const** (p. 349)

**6.283.3.35 bool Arc::UserConfig::KeySize ( int *newKeySize* ) [inline]**

Set key size.

Set size/strenght of private key of credentials obtained from Short Lived Credentials **Service** (p. 285).

The attribute associated with this setter method is 'keysize'.

**Parameters**

*newKeySize* is the size, an an integer, of the key.

**Returns**

This method always returns `true`.

**See also**

**KeySize() const** (p. 350)

**KeyPath(const std::string&)** (p. 349)

**KeyPassword(const std::string&)** (p. 349)

**6.283.3.36 bool Arc::UserConfig::LoadConfigurationFile ( const std::string & *confFile*, bool *ignoreJobListFile* = `true` )**

Load specified configuration file.

The configuration file passed is parsed by this method by using the **IniConfig** (p. 178) class. If the parsing is unsuccessful a WARNING is reported.

The format of the configuration file should follow that of INI, and every attribute present in the file is only allowed once, if otherwise a WARNING will be reported. The file can contain at most two sections, one named common and the other name alias. If other sections exist a WARNING will be reported. Only the following attributes is allowed in the common section of the configuration `file`:

- `certificatepath` (**CertificatePath(const std::string&)** (p. 343))
- `keypath` (**KeyPath(const std::string&)** (p. 349))
- `proxypath` (**ProxyPath(const std::string&)** (p. 354))
- `cacertificatesdirectory` (**CACertificatesDirectory(const std::string&)** (p. 342))
- `cacertificatepath` (**CACertificatePath(const std::string&)** (p. 341))
- `timeout` (**Timeout(int)** (p. 356))
- `joblist` (**JobListFile(const std::string&)** (p. 348))
- `defaultservices` (**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 338))
- `rejectservices` (**AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p. 338))
- `verbosity` (**Verbosity(const std::string&)** (p. 358))
- `brokername` (**Broker(const std::string&)** (p. 340) or **Broker(const std::string&, const std::string&)** (p. 341))

- brokerarguments (**Broker(const std::string&)** (p.340) or **Broker(const std::string&, const std::string&)** (p.341))
- bartender (**Bartender(const std::list<URL>&)**)
- vomsserverpath (**VOMSServerPath(const std::string&)** (p.358))
- username (**UserName(const std::string&)** (p.357))
- password (**Password(const std::string&)** (p.353))
- keypassword (**KeyPassword(const std::string&)** (p.349))
- keysize (**KeySize(int)** (p.351))
- certificatelifetime (**CertificateLifeTime(const Period&)** (p.343))
- slcs (**SLCS(const URL&)** (p.355))
- storedirectory (**StoreDirectory(const std::string&)** (p.355))
- idpname (**IdPName(const std::string&)** (p.346))

where the method in parentheses is the associated setter method. If other attributes exist in the common section a WARNING will be reported for each of these attributes. In the alias section aliases can be defined, and should represent a selection of services. The alias can then be referred to by input to the **AddServices(const std::list<std::string>&, ServiceType)** (p.338) and **AddServices(const std::list<std::string>&, const std::list<std::string>&, ServiceType)** (p.338) methods. An alias can not contain any of the characters '.', ':', ',' or '\t' and should be defined as follows:

```
< alias_name >=< service_type >:< flavour >:< service_url > | < alias_ref > [...]
```

where <alias\_name> is the name of the defined alias, <service\_type> is the service type in lower case, <flavour> is the type of middleware plugin to use, <service\_url> is the **URL** (p.323) which should be used to contact the service and <alias\_ref> is another defined alias. The parsed aliases will be stored internally and resolved when needed. If a alias already exist, and another alias with the same name is parsed then this other alias will overwrite the existing alias.

### Parameters

*conffile* is the path to the configuration file.

*ignoreJobListFile* is a optional boolean which indicates whether the joblistfile attribute in the configuration file should be ignored. Default is to ignore it (`true`).

### Returns

If loading the configuration file succeeds `true` is returned, otherwise `false` is returned.

### See also

**SaveToFile()** (p.355)

### 6.283.3.37 Arc::UserConfig::operator bool ( void ) const [inline]

Check for validity.

The validity of an object created from this class can be checked using this casting operator. An object is valid if the constructor did not encounter any errors.

### See also

**operator!()** (p.353)

**6.283.3.38** `bool Arc::UserConfig::operator! ( void ) const [inline]`

Check for non-validity.

See `operator bool()` (p. 352) for a description.

See also

`operator bool()` (p. 352)

**6.283.3.39** `const std::string& Arc::UserConfig::OverlayFile ( ) const [inline]`

Get path to configuration overlay file.

**Returns**

The overlay file path

See also

`OverlayFile(const std::string&)` (p. 353)

**6.283.3.40** `bool Arc::UserConfig::OverlayFile ( const std::string & path ) [inline]`

Set path to configuration overlay file.

Content of specified file is a backdoor to configuration XML generated from information stored in this class. The content of file is passed to **BaseConfig** (p. 60) class in `ApplyToConfig(BaseConfig&)` then merged with internal configuration XML representation. This feature is meant for quick prototyping/testing/tuning of functionality without rewriting code. It is meant for developers and most users won't need it.

The attribute associated with this setter method is 'overlayfile'.

**Parameters**

*path* is the new overlay file path.

**Returns**

This method always returns `true`.

See also

**6.283.3.41** `bool Arc::UserConfig::Password ( const std::string & newPassword ) [inline]`

Set password.

Set password which is used for requesting credentials from Short Lived Credentials **Service** (p. 285).

The attribute associated with this setter method is 'password'.

**Parameters**

*newPassword* is the new password to set.

**Returns**

This method always returns true.

**See also**

**Password() const** (p. 354)

**6.283.3.42 const std::string& Arc::UserConfig::Password ( ) const [inline]**

Get password.

Get password which is used for requesting credentials from Short Lived Credentials **Service** (p. 285).

**Returns**

The password is returned.

**See also**

**Password(const std::string&)** (p. 353)

**6.283.3.43 bool Arc::UserConfig::ProxyPath ( const std::string & newProxyPath ) [inline]**

Set path to user proxy.

This method will set the path of the user proxy. Note that the **InitializeCredentials()** (p. 347) method will also try to set this path, by searching in different locations.

The attribute associated with this setter method is 'proxypath'

**Parameters**

*newProxyPath* is the path to a user proxy.

**Returns**

This method always returns `true`.

**See also**

**InitializeCredentials()** (p. 347)

**CredentialsFound()** (p. 345)

**ProxyPath() const** (p. 354)

**6.283.3.44 const std::string& Arc::UserConfig::ProxyPath ( ) const [inline]**

Get path to user proxy.

Retrieve path to user proxy.

**Returns**

Returns the path to the user proxy.

**See also**

**ProxyPath(const std::string&)** (p. 354)

**6.283.3.45** `bool Arc::UserConfig::SaveToFile ( const std::string & filename ) const`

Save to INI file.

This method will save the object data as a INI file. The saved file can be loaded with the LoadConfigurationFile method.

**Parameters**

*filename* the name of the file which the data will be saved to.

**Returns**

`false` if unable to get handle on file, otherwise `true` is returned.

**See also**

`LoadConfigurationFile()` (p. 351)

**6.283.3.46** `bool Arc::UserConfig::SLCS ( const URL & newSLCS ) [inline]`

Set the `URL` (p. 323) to the Short Lived Certificate `Service` (p. 285) (SLCS).

The attribute associated with this setter method is 'slcs'.

**Parameters**

*newSLCS* is the `URL` (p. 323) to the SLCS

**Returns**

This method always returns `true`.

**See also**

`SLCS() const` (p. 355)

**6.283.3.47** `const URL& Arc::UserConfig::SLCS ( ) const [inline]`

Get the `URL` (p. 323) to the Short Lived Certificate `Service` (p. 285) (SLCS).

**Returns**

The SLCS is returned.

**See also**

`SLCS(const URL&)` (p. 355)

**6.283.3.48** `bool Arc::UserConfig::StoreDirectory ( const std::string & newStoreDirectory ) [inline]`

Set store directory.

Sets directory which will be used to store credentials obtained from Short Lived `Credential` (p. 90) Service.

The attribute associated with this setter method is 'storedirectory'.

**Parameters**

*newStoreDirectory* is the path to the store directory.

**Returns**

This method always returns `true`.

**See also****6.283.3.49** `const std::string& Arc::UserConfig::StoreDirectory ( ) const [inline]`

Get store directory.

Sets directory which is used to store credentials obtained from Short Lived **Credential** (p. 90) Service.

**Returns**

The path to the store directory is returned.

**See also**

**StoreDirectory(const std::string&)** (p. 355)

**6.283.3.50** `int Arc::UserConfig::Timeout ( ) const [inline]`

Get timeout.

Returns the timeout in seconds.

**Returns**

timeout in seconds.

**See also**

**Timeout(int)** (p. 356)

**DEFAULT\_TIMEOUT** (p. 360)

**6.283.3.51** `bool Arc::UserConfig::Timeout ( int newTimeout )`

Set timeout.

When communicating with a service the timeout specifies how long, in seconds, the communicating instance should wait for a response. If the response have not been recieved before this period in time, the connection is typically dropped, and an error will be reported.

This method will set the timeout to the specified integer. If the passed integer is less than or equal to 0 then `false` is returned and the timeout will not be set, otherwise `true` is returned and the timeout will be set to the new value.

The attribute associated with this setter method is 'timeout'.

**Parameters**

*newTimeout* the new timeout value in seconds.

**Returns**

false in case *newTimeout* <= 0, otherwise true.

**See also**

**Timeout() const** (p. 356)

**DEFAULT\_TIMEOUT** (p. 360)

**6.283.3.52 const std::string& Arc::UserConfig::UserName ( ) const [inline]**

Get user-name.

Get username which is used for requesting credentials from Short Lived Credentials **Service** (p. 285).

**Returns**

The username is returned.

**See also**

**UserName(const std::string&)** (p. 357)

**6.283.3.53 bool Arc::UserConfig::UserName ( const std::string & name ) [inline]**

Set user-name for SLCS.

Set username which is used for requesting credentials from Short Lived Credentials **Service** (p. 285).

The attribute associated with this setter method is 'username'.

**Parameters**

*name* is the name of the user.

**Returns**

This method always return true.

**See also**

**UserName() const** (p. 357)

**6.283.3.54 const std::string& Arc::UserConfig::UtilsDirPath ( ) const [inline]**

Get path to directory storing utility files for DataPoints.

**Returns**

The utils dir path

**See also**

**UtilsDirPath(const std::string&)** (p. 358)

**6.283.3.55** `bool Arc::UserConfig::UtilsDirPath ( const std::string & dir )`

Set path to directory storing utility files for DataPoints.

Some DataPoints can store information on remote services in local files. This method sets the path to the directory containing these files. For example arc\* tools set it to ARCUSERDIRECTORY and A-REX sets it to the control directory. The directory is created if it does not exist.

**Parameters**

*path* is the new utils dir path.

**Returns**

This method always returns `true`.

**6.283.3.56** `const std::string& Arc::UserConfig::Verbosity ( ) const [inline]`

Get the user selected level of verbosity.

The string representation of the verbosity level specified by the user is returned when calling this method. If the user have not specified the verbosity level the empty string will be referenced.

**Returns**

the verbosity level, or empty if it has not been set.

**See also**

`Verbosity(const std::string&)` (p. 358)

**6.283.3.57** `bool Arc::UserConfig::Verbosity ( const std::string & newVerbosity )`

Set verbosity.

The verbosity will be set when invoking this method. If the string passed cannot be parsed into a corresponding `LogLevel`, using the function a `WARNING` is reported and `false` is returned, otherwise `true` is returned.

The attribute associated with this setter method is 'verbosity'.

**Returns**

`true` in case the verbosity could be set to a allowed `LogLevel`, otherwise `false`.

**See also**

`Verbosity() const` (p. 358)

**6.283.3.58** `bool Arc::UserConfig::VOMSServerPath ( const std::string & path ) [inline]`

Set path to file containing VOMS configuration.

Set path to file which contains list of VOMS services and associated configuration parameters needed to contact those services. It is used by arcproxy.

The attribute associated with this setter method is 'vomsserverpath'.

**Parameters**

*path* the path to VOMS configuration file

**Returns**

This method always return true.

**See also**

**VOMSServerPath() const** (p. 359)

**6.283.3.59 const std::string& Arc::UserConfig::VOMSServerPath ( ) const [inline]**

Get path to file containing VOMS configuration.

Get path to file which contains list of VOMS services and associated configuration parameters.

**Returns**

The path to VOMS configuration file is returned.

**See also**

**VOMSServerPath(const std::string&)** (p. 358)

**6.283.4 Field Documentation****6.283.4.1 const std::string Arc::UserConfig::ARCUSERDIRECTORY [static]**

Path to ARC user home directory.

The *ARCUSERDIRECTORY* variable is the path to the ARC home directory of the current user. This path is created using the `User::Home()` method.

**See also**

`User::Home()`

**6.283.4.2 const std::string Arc::UserConfig::DEFAULT\_BROKER [static]**

Default broker.

The *DEFAULT\_BROKER* specifies the name of the broker which should be used in case no broker is explicitly chosen.

**See also**

**Broker** (p. 62)

**Broker(const std::string&)** (p. 340)

**Broker(const std::string&, const std::string&)** (p. 341)

**Broker() const** (p. 341)

#### 6.283.4.3 `const int Arc::UserConfig::DEFAULT_TIMEOUT = 20` `[static]`

Default timeout in seconds.

The `DEFAULT_TIMEOUT` specifies interval which will be used in case no timeout interval have been explicitly specified. For a description about timeout see `Timeout(int)` (p. 356).

See also

`Timeout(int)` (p. 356)

`Timeout() const` (p. 356)

#### 6.283.4.4 `const std::string Arc::UserConfig::DEFAULTCONFIG` `[static]`

Path to default configuration file.

The `DEFAULTCONFIG` variable is the path to the default configuration file used in case no configuration file have been specified. The path is created from the `ARCUSERDIRECTORY` object.

#### 6.283.4.5 `const std::string Arc::UserConfig::EXAMPLECONFIG` `[static]`

Path to example configuration.

The `EXAMPLECONFIG` variable is the path to the example configuration file.

#### 6.283.4.6 `const std::string Arc::UserConfig::SYSCONFIG` `[static]`

Path to system configuration.

The `SYSCONFIG` variable is the path to the system configuration file. This variable is only equal to `SYSCONFIGARCLOC` if ARC is installed in the root (highly unlikely).

#### 6.283.4.7 `const std::string Arc::UserConfig::SYSCONFIGARCLOC` `[static]`

Path to system configuration at ARC location.

The `SYSCONFIGARCLOC` variable is the path to the system configuration file which reside at the ARC installation location.

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

- `UserConfig.h`

## 6.284 `Arc::UsernameToken` Class Reference

Interface for manipulation of WS-Security according to Username Token `Profile` (p. 258).

```
#include <UsernameToken.h>
```

### Public Types

- enum `PasswordType`

## Public Member Functions

- **UsernameToken** (SOAPEnvelope &soap)
- **UsernameToken** (SOAPEnvelope &soap, const std::string &username, const std::string &password, const std::string &uid, **PasswordType** pwdtype)
- **UsernameToken** (SOAPEnvelope &soap, const std::string &username, const std::string &id, bool mac, int iteration)
- **operator bool** (void)
- std::string **Username** (void)
- bool **Authenticate** (const std::string &password, std::string &derived\_key)
- bool **Authenticate** (std::istream &password, std::string &derived\_key)

### 6.284.1 Detailed Description

Interface for manipulation of WS-Security according to Username Token **Profile** (p. 258).

### 6.284.2 Member Enumeration Documentation

#### 6.284.2.1 enum Arc::UsernameToken::PasswordType

SOAP header element

### 6.284.3 Constructor & Destructor Documentation

#### 6.284.3.1 Arc::UsernameToken::UsernameToken ( SOAPEnvelope & soap )

Link to existing SOAP header and parse Username Token information. Username Token related information is extracted from SOAP header and stored in class variables.

#### 6.284.3.2 Arc::UsernameToken::UsernameToken ( SOAPEnvelope & soap, const std::string & username, const std::string & password, const std::string & uid, PasswordType pwdtype )

Add Username Token information into the SOAP header. Generated token contains elements Username and Password and is meant to be used for authentication.

#### Parameters

*soap* the SOAP message

*username* <wsse:Username>...</wsse:Username> - if empty it is entered interactively from stdin

*password* <wsse:Password Type="...">...</wsse:Password> - if empty it is entered interactively from stdin

*uid* <wsse:UsernameToken (p. 360) wsu:ID="...">

*pwdtype* <wsse:Password Type="...">...</wsse:Password>

### 6.284.3.3 `Arc::UsernameToken::UsernameToken` ( `SOAPEnvelope & soap`, `const std::string & username`, `const std::string & id`, `bool mac`, `int iteration` )

Add Username Token information into the SOAP header. Generated token contains elements Username and Salt and is meant to be used for deriving Key Derivation.

#### Parameters

*soap* the SOAP message

*username* `<wsse:Username>...</wsse:Username>`

*mac* if derived key is meant to be used for **Message** (p. 210) Authentication Code

*iteration* `<wsse11:Iteration>...</wsse11:Iteration>`

## 6.284.4 Member Function Documentation

### 6.284.4.1 `bool Arc::UsernameToken::Authenticate` ( `const std::string & password`, `std::string & derived_key` )

Checks parsed/generated token against specified password. If token is meant to be used for deriving a key then key is returned in `derived_key`. In that case authentication is performed outside of **UsernameToken** (p. 360) class using obtained `derived_key`.

### 6.284.4.2 `bool Arc::UsernameToken::Authenticate` ( `std::istream & password`, `std::string & derived_key` )

Checks parsed token against password stored in specified stream. If token is meant to be used for deriving a key then key is returned in `derived_key`

### 6.284.4.3 `Arc::UsernameToken::operator bool` ( `void` )

Returns true of constructor succeeded

### 6.284.4.4 `std::string Arc::UsernameToken::Username` ( `void` )

Returns username associated with this instance

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

- UsernameToken.h

## 6.285 `Arc::UserSwitch` Class Reference

```
#include <User.h>
```

### 6.285.1 Detailed Description

If this class is created user identity is switched to provided uid and gid. Due to internal lock there will be only one valid instance of this class. Any attempt to create another instance will block till first one is

destroyed. If uid and gid are set to 0 then user identity is not switched. But lock is applied anyway. The lock has dual purpose. First and most important is to protect communication with underlying operating system which may depend on user identity. For that it is advisable for code which talks to operating system to acquire valid instance of this class. Care must be taken for not to hold that instance too long cause that may block other code in multithreaded environment. Other purpose of this lock is to provide workaround for glibc bug in `__nptl_setxid`. That bug causes lockup of `seteuid()` function if racing with fork. To avoid this problem the lock mentioned above is used by **Run** (p. 270) class while spawning new process.

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

- User.h

## 6.286 Arc::VOMSTrustList Class Reference

```
#include <VOMSUtil.h>
```

### Public Member Functions

- **VOMSTrustList** (const std::vector< std::string > &encoded\_list)
- **VOMSTrustList** (const std::vector< VOMSTrustChain > &chains, const std::vector< VOMSTrustRegex > &regexs)
- VOMSTrustChain & **AddChain** (const VOMSTrustChain &chain)
- VOMSTrustChain & **AddChain** (void)
- **RegularExpression** & **AddRegex** (const VOMSTrustRegex &reg)

### 6.286.1 Detailed Description

Stores definitions for making decision if VOMS server is trusted

### 6.286.2 Constructor & Destructor Documentation

#### 6.286.2.1 Arc::VOMSTrustList::VOMSTrustList ( const std::vector< std::string > & encoded\_list )

Creates chain lists and regexps from plain list. List is made of chunks delimited by elements containing pattern "NEXT CHAIN". Each chunk with more than one element is converted into one instance of VOMSTrustChain. Chunks with single element are converted to VOMSTrustChain if element does not have special symbols. Otherwise it is treated as regular expression. Those symbols are '^','\$' and '\*'. Trusted chains can be configured in two ways: one way is: <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=host/arthur.hep.lu.se</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>----NEXT CHAIN--- </tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/CN=CERN Trusted Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> the other way is: <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=host/arthur.hep.lu.se</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDNChain> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch</tls:VOMSCertTrustDN> <tls:VOMSCertTrustDN>/DC=ch/DC=cern/CN=CERN Trusted Certification

Authority</tls:VOMSCertTrustDN> </tls:VOMSCertTrustDNChain> each chunk is supposed to contain a suit of DN of trusted certificate chain, in which the first DN is the DN of the certificate (cert0) which is used to sign the Attribute Certificate (AC), the second DN is the DN of the issuer certificate(cert1) which is used to sign cert0. So if there are one or more intermediate issuers, then there should be 3 or more than 3 DNs in this chunk (considering cert0 and the root certificate, plus the intermediate certificate) .

### 6.286.2.2 Arc::VOMSTrustList::VOMSTrustList ( const std::vector< VOMSTrustChain > & chains, const std::vector< VOMSTrustRegex > & regexs )

Creates chain lists and regexps from those specified in arguments. See **AddChain()** (p.364) and **AddRegex()** (p.364) for more information.

## 6.286.3 Member Function Documentation

### 6.286.3.1 VOMSTrustChain& Arc::VOMSTrustList::AddChain ( const VOMSTrustChain & chain )

Adds chain of trusted DNs to list. During verification each signature of AC is checked against all stored chains. DNs of chain of certificate used for signing AC are compared against DNs stored in these chains one by one. If needed DN of issuer of last certificate is checked too. Comparison succeeds if DNs in at least one stored chain are same as those in certificate chain. Comparison stops when all DNs in stored chain are compared. If there are more DNs in stored chain than in certificate chain then comparison fails. Empty stored list matches any certificate chain. Taking into account that certificate chains are verified down to trusted CA anyway, having more than one DN in stored chain seems to be useless. But such feature may be found useful by some very strict sysadmins. ??? IMO,DN list here is not only for authentication, it is also kind of ACL, which means the AC consumer only trusts those DNs which issues AC.

### 6.286.3.2 VOMSTrustChain& Arc::VOMSTrustList::AddChain ( void )

Adds empty chain of trusted DNs to list.

### 6.286.3.3 RegularExpression& Arc::VOMSTrustList::AddRegex ( const VOMSTrustRegex & reg )

Adds regular expression to list. During verification each signature of AC is checked against all stored regular expressions. DN of signing certificate must match at least one of stored regular expressions.

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

- VOMSUtil.h

## 6.287 Arc::WSAEndpointReference Class Reference

Interface for manipulation of WS-Addressing Endpoint Reference.

```
#include <WSA.h>
```

### Public Member Functions

- **WSAEndpointReference** (XMLNode epr)

- **WSAEndpointReference** (const **WSAEndpointReference** &wsa)
- **WSAEndpointReference** (const std::string &address)
- **WSAEndpointReference** (void)
- **~WSAEndpointReference** (void)
- std::string **Address** (void) const
- void **Address** (const std::string &uri)
- **WSAEndpointReference** & **operator=** (const std::string &address)
- **XMLNode ReferenceParameters** (void)
- **XMLNode MetaData** (void)
- **operator XMLNode** (void)

### 6.287.1 Detailed Description

Interface for manipulation of WS-Addressing Endpoint Reference. It works on Endpoint Reference stored in XML tree. No information is stored in this object except reference to corresponding XML subtree.

### 6.287.2 Constructor & Destructor Documentation

#### 6.287.2.1 Arc::WSAEndpointReference::WSAEndpointReference ( XMLNode *epr* )

Link to top level EPR XML node Linking to existing EPR in XML tree

#### 6.287.2.2 Arc::WSAEndpointReference::WSAEndpointReference ( const WSAEndpointReference & *wsa* )

Copy constructor

#### 6.287.2.3 Arc::WSAEndpointReference::WSAEndpointReference ( const std::string & *address* )

Creating independent EPR - not implemented

#### 6.287.2.4 Arc::WSAEndpointReference::WSAEndpointReference ( void )

Dummy constructor - creates invalid instance

#### 6.287.2.5 Arc::WSAEndpointReference::~~WSAEndpointReference ( void )

Destructor. All empty elements of EPR XML are destroyed here too

### 6.287.3 Member Function Documentation

#### 6.287.3.1 std::string Arc::WSAEndpointReference::Address ( void ) const

Returns Address (URL (p. 323)) encoded in EPR

**6.287.3.2 void Arc::WSAEndpointReference::Address ( const std::string & uri )**

Assigns new Address value. If EPR had no Address element it is created.

**6.287.3.3 XMLNode Arc::WSAEndpointReference::MetaData ( void )**

Access to MetaData element of EPR. Obtained XML element should be manipulated directly in application-dependent way. If EPR had no MetaData element it is created.

**6.287.3.4 Arc::WSAEndpointReference::operator XMLNode ( void )**

Returns reference to EPR top XML node

**6.287.3.5 WSAEndpointReference& Arc::WSAEndpointReference::operator= ( const std::string & address )**

Same as Address(uri)

**6.287.3.6 XMLNode Arc::WSAEndpointReference::ReferenceParameters ( void )**

Access to ReferenceParameters element of EPR. Obtained XML element should be manipulated directly in application-dependent way. If EPR had no ReferenceParameters element it is created.

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

- WSA.h

**6.288 Arc::WSAHeader Class Reference**

Interface for manipulation WS-Addressing information in SOAP header.

```
#include <WSA.h>
```

**Public Member Functions**

- **WSAHeader** (SOAPEnvelope &soap)
- **WSAHeader** (const std::string &action)
- std::string **To** (void) const
- void **To** (const std::string &uri)
- **WSAEndpointReference From** (void)
- **WSAEndpointReference ReplyTo** (void)
- **WSAEndpointReference FaultTo** (void)
- std::string **Action** (void) const
- void **Action** (const std::string &uri)
- std::string **MessageID** (void) const
- void **MessageID** (const std::string &uri)
- std::string **RelatesTo** (void) const
- void **RelatesTo** (const std::string &uri)

- `std::string RelationshipType` (void) const
- `void RelationshipType` (const `std::string &uri`)
- `XMLNode ReferenceParameter` (int `n`)
- `XMLNode ReferenceParameter` (const `std::string &name`)
- `XMLNode NewReferenceParameter` (const `std::string &name`)
- `operator XMLNode` (void)

### Static Public Member Functions

- static `bool Check` (`SOAPEnvelope &soap`)

### Protected Attributes

- `bool header_allocated_`

## 6.288.1 Detailed Description

Interface for manipulation WS-Addressing information in SOAP header. It works on Endpoint Reference stored in XML tree. No information is stored in this object except reference to corresponding XML subtree.

## 6.288.2 Constructor & Destructor Documentation

### 6.288.2.1 `Arc::WSAHeader::WSAHeader ( SOAPEnvelope & soap )`

Linking to a header of existing SOAP message

### 6.288.2.2 `Arc::WSAHeader::WSAHeader ( const std::string & action )`

Creating independent SOAP header - not implemented

## 6.288.3 Member Function Documentation

### 6.288.3.1 `std::string Arc::WSAHeader::Action ( void ) const`

Returns content of Action element of SOAP Header.

### 6.288.3.2 `void Arc::WSAHeader::Action ( const std::string & uri )`

Set content of Action element of SOAP Header. If such element does not exist it's created.

### 6.288.3.3 `static bool Arc::WSAHeader::Check ( SOAPEnvelope & soap ) [static]`

Tells if specified SOAP message has WSA header

**6.288.3.4 WSAEndpointReference Arc::WSAHeader::FaultTo ( void )**

Returns FaultTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulated.

**6.288.3.5 WSAEndpointReference Arc::WSAHeader::From ( void )**

Returns From element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulated.

**6.288.3.6 std::string Arc::WSAHeader::MessageID ( void ) const**

Returns content of MessageID element of SOAP Header.

**6.288.3.7 void Arc::WSAHeader::MessageID ( const std::string & uri )**

Set content of MessageID element of SOAP Header. If such element does not exist it's created.

**6.288.3.8 XMLNode Arc::WSAHeader::NewReferenceParameter ( const std::string & name )**

Creates new ReferenceParameter element with specified name. Returns reference to created element.

**6.288.3.9 Arc::WSAHeader::operator XMLNode ( void )**

Returns reference to SOAP Header - not implemented

**6.288.3.10 XMLNode Arc::WSAHeader::ReferenceParameter ( const std::string & name )**

Returns first ReferenceParameter element with specified name

**6.288.3.11 XMLNode Arc::WSAHeader::ReferenceParameter ( int n )**

Return n-th ReferenceParameter element

**6.288.3.12 void Arc::WSAHeader::RelatesTo ( const std::string & uri )**

Set content of RelatesTo element of SOAP Header. If such element does not exist it's created.

**6.288.3.13 std::string Arc::WSAHeader::RelatesTo ( void ) const**

Returns content of RelatesTo element of SOAP Header.

**6.288.3.14 void Arc::WSAHeader::RelationshipType ( const std::string & uri )**

Set content of RelationshipType element of SOAP Header. If such element does not exist it's created.

**6.288.3.15** `std::string Arc::WSAHeader::RelationshipType ( void ) const`

Returns content of RelationshipType element of SOAP Header.

**6.288.3.16** `WSAEndpointReference Arc::WSAHeader::ReplyTo ( void )`

Returns ReplyTo element of SOAP Header. If such element does not exist it's created. Obtained element may be manipulated.

**6.288.3.17** `std::string Arc::WSAHeader::To ( void ) const`

Returns content of To element of SOAP Header.

**6.288.3.18** `void Arc::WSAHeader::To ( const std::string & uri )`

Set content of To element of SOAP Header. If such element does not exist it's created.

**6.288.4** **Field Documentation****6.288.4.1** `bool Arc::WSAHeader::header_allocated_ [protected]`

SOAP header element

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

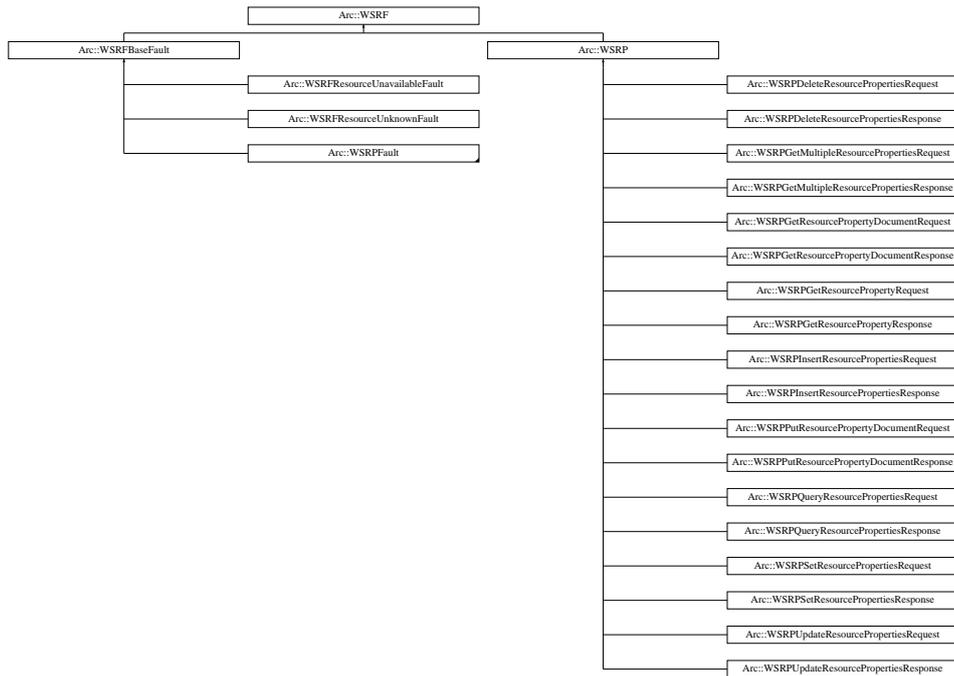
- WSA.h

**6.289** **Arc::WSRF Class Reference**

Base class for every WSRF (p. 369) message.

```
#include <WSRF.h>
```

Inheritance diagram for Arc::WSRF:



## Public Member Functions

- **WSRF** (SOAPEnvelope &soap, const std::string &action="")
- **WSRF** (bool fault=false, const std::string &action="")
- virtual SOAPEnvelope & **SOAP** (void)
- virtual **operator bool** (void)

## Protected Member Functions

- void **set\_namespaces** (void)

## Protected Attributes

- bool **allocated\_**
- bool **valid\_**

### 6.289.1 Detailed Description

Base class for every **WSRF** (p. 369) message. This class is not intended to be used directly. Use it like reference while passing through unknown **WSRF** (p. 369) message or use classes derived from it.

### 6.289.2 Constructor & Destructor Documentation

#### 6.289.2.1 Arc::WSRF::WSRF ( SOAPEnvelope & soap, const std::string & action = "" )

Constructor - creates object out of supplied SOAP tree.



## Public Member Functions

- **WSRFBaseFault** (SOAPEnvelope &soap)
- **WSRFBaseFault** (const std::string &type)

## Protected Member Functions

- void **set\_namespaces** (void)

### 6.290.1 Detailed Description

Base class for **WSRF** (p. 369) fault messages. Use classes inherited from it for specific faults.

### 6.290.2 Constructor & Destructor Documentation

#### 6.290.2.1 Arc::WSRFBaseFault::WSRFBaseFault ( SOAPEnvelope & soap )

Constructor - creates object out of supplied SOAP tree.

#### 6.290.2.2 Arc::WSRFBaseFault::WSRFBaseFault ( const std::string & type )

Constructor - creates new **WSRF** (p. 369) fault

### 6.290.3 Member Function Documentation

#### 6.290.3.1 void Arc::WSRFBaseFault::set\_namespaces ( void ) [protected]

set WS-ResourceProperties namespaces and default prefixes in SOAP message

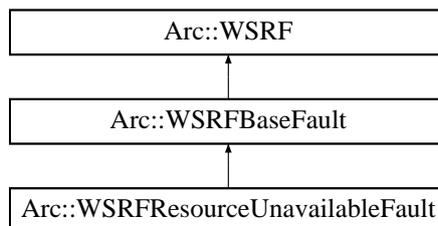
Reimplemented from **Arc::WSRF** (p. 371).

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

- WSRFBaseFault.h

## 6.291 Arc::WSRFResourceUnavailableFault Class Reference

Inheritance diagram for Arc::WSRFResourceUnavailableFault:

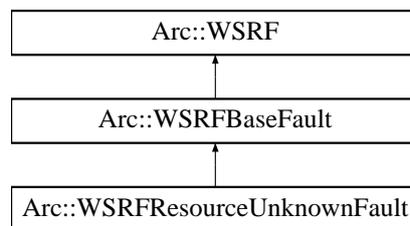


The documentation for this class was generated from the following file:

- WSRFBaseFault.h

## 6.292 Arc::WSRFResourceUnknownFault Class Reference

Inheritance diagram for Arc::WSRFResourceUnknownFault:



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

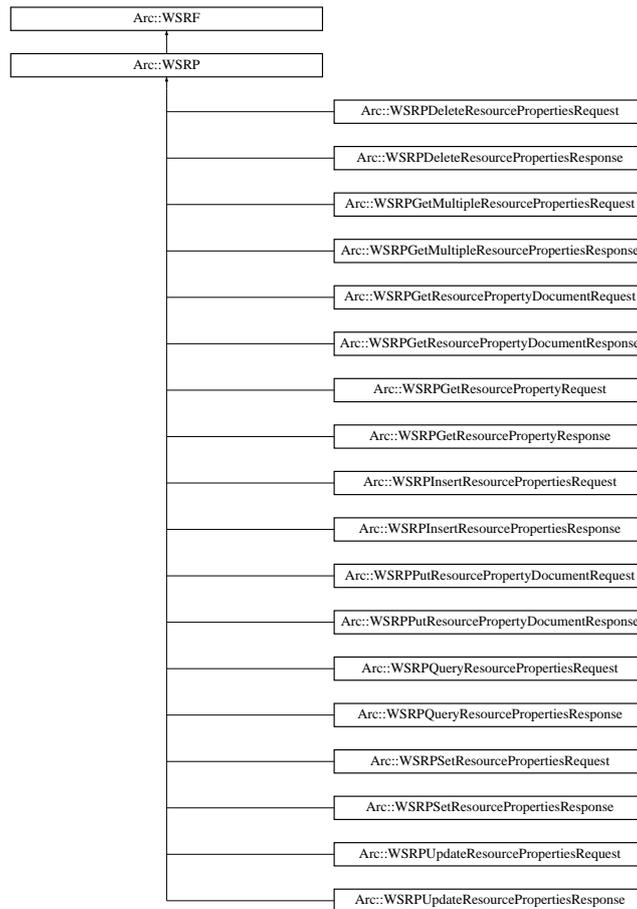
- WSRFBaseFault.h

## 6.293 Arc::WSRP Class Reference

Base class for WS-ResourceProperties structures.

```
#include <WSResourceProperties.h>
```

Inheritance diagram for Arc::WSRP:



## Public Member Functions

- **WSRP** (bool fault=false, const std::string &action="")
- **WSRP** (SOAPEnvelope &soap, const std::string &action="")

## Protected Member Functions

- void **set\_namespaces** (void)

### 6.293.1 Detailed Description

Base class for WS-ResourceProperties structures. Inheriting classes implement specific WS-ResourceProperties messages and their properties/elements. Refer to WS-ResourceProperties specifications for things specific to every message.

### 6.293.2 Constructor & Destructor Documentation

#### 6.293.2.1 Arc::WSRP::WSRP ( bool *fault* = *false*, const std::string & *action* = "" )

Constructor - prepares object for creation of new **WSRP** (p. 373) request/response/fault

### 6.293.2.2 Arc::WSRP::WSRP ( SOAPEnvelope & soap, const std::string & action = "" )

Constructor - creates object out of supplied SOAP tree. It does not check if 'soap' represents valid WS-ResourceProperties structure. Actual check for validity of structure has to be done by derived class.

## 6.293.3 Member Function Documentation

### 6.293.3.1 void Arc::WSRP::set\_namespaces ( void ) [protected]

set WS-ResourceProperties namespaces and default prefixes in SOAP message

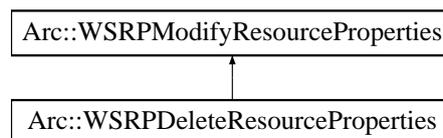
Reimplemented from Arc::WSRF (p. 371).

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

- WSResourceProperties.h

## 6.294 Arc::WSRPDeleteResourceProperties Class Reference

Inheritance diagram for Arc::WSRPDeleteResourceProperties:

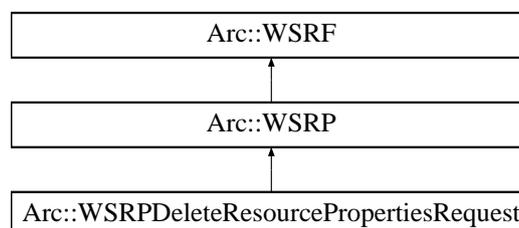


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.295 Arc::WSRPDeleteResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPDeleteResourcePropertiesRequest:

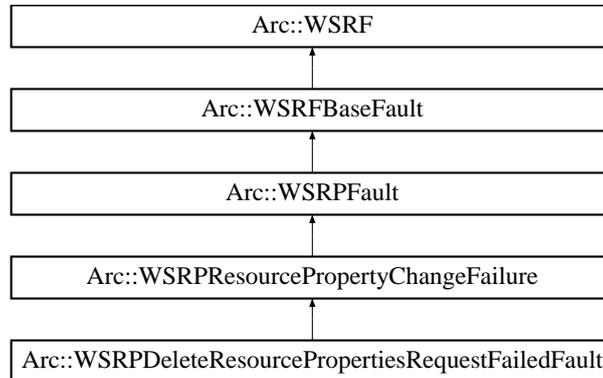


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.296 Arc::WSRPDeleteResourcePropertiesRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPDeleteResourcePropertiesRequestFailedFault:

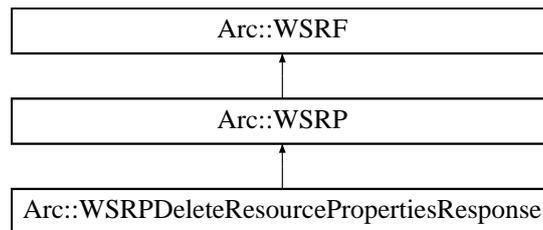


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.297 Arc::WSRPDeleteResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPDeleteResourcePropertiesResponse:



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

- WSResourceProperties.h

## 6.298 Arc::WSRPFault Class Reference

Base class for WS-ResourceProperties faults.

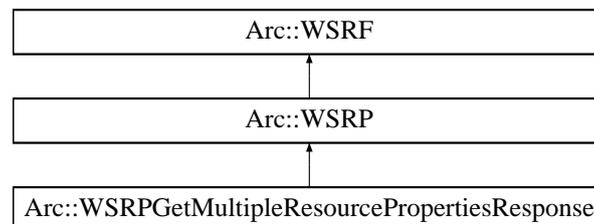
```
#include <WSResourceProperties.h>
```

Inheritance diagram for Arc::WSRPFault:



### 6.300 Arc::WSRPGetMultipleResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPGetMultipleResourcePropertiesResponse:

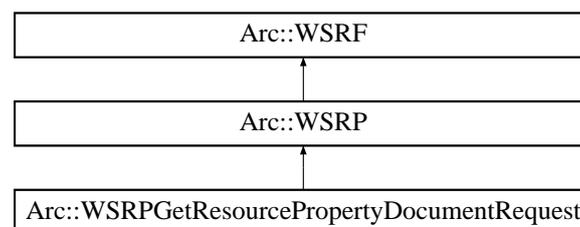


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.301 Arc::WSRPGetResourcePropertyDocumentRequest Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyDocumentRequest:

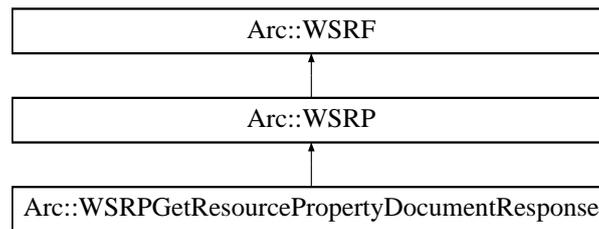


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.302 Arc::WSRPGetResourcePropertyDocumentResponse Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyDocumentResponse:

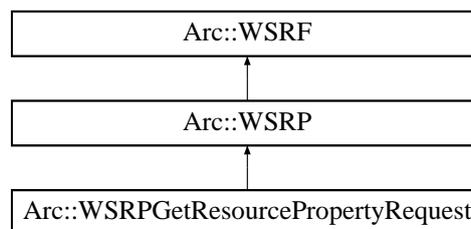


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.303 Arc::WSRPGetResourcePropertyRequest Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyRequest:

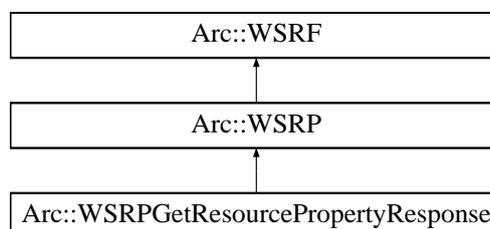


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.304 Arc::WSRPGetResourcePropertyResponse Class Reference

Inheritance diagram for Arc::WSRPGetResourcePropertyResponse:

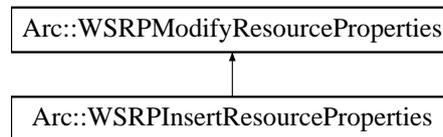


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.305 Arc::WSRPInsertResourceProperties Class Reference

Inheritance diagram for Arc::WSRPInsertResourceProperties:

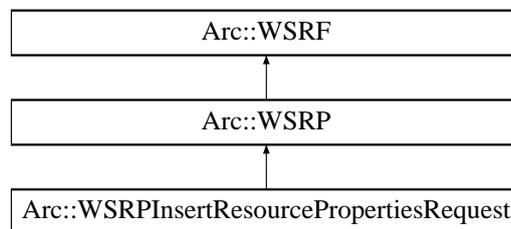


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.306 Arc::WSRPInsertResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPInsertResourcePropertiesRequest:

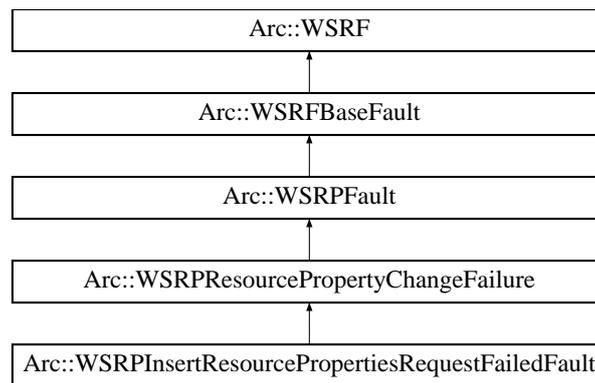


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.307 Arc::WSRPInsertResourcePropertiesRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPInsertResourcePropertiesRequestFailedFault:

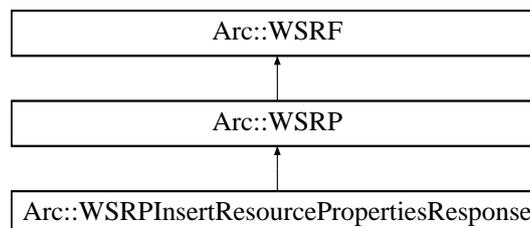


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.308 Arc::WSRPInsertResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPInsertResourcePropertiesResponse:

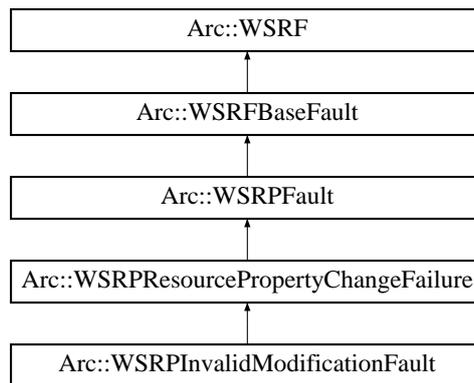


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.309 Arc::WSRPInvalidModificationFault Class Reference

Inheritance diagram for Arc::WSRPInvalidModificationFault:

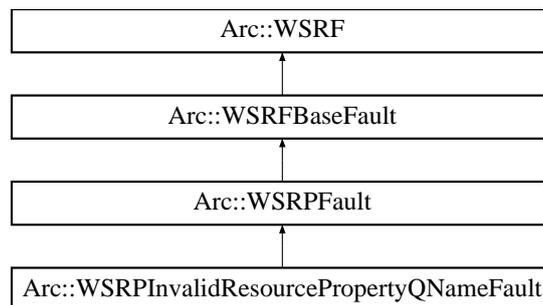


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.310 Arc::WSRPInvalidResourcePropertyQNameFault Class Reference

Inheritance diagram for Arc::WSRPInvalidResourcePropertyQNameFault:

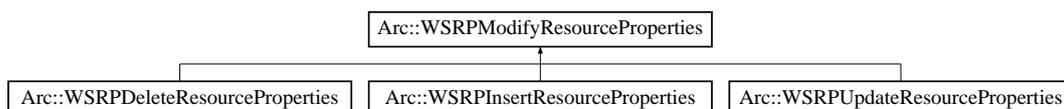


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.311 Arc::WSRPModifyResourceProperties Class Reference

Inheritance diagram for Arc::WSRPModifyResourceProperties:

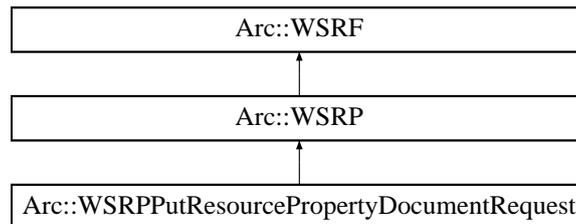


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.312 Arc::WSRPPutResourcePropertyDocumentRequest Class Reference

Inheritance diagram for Arc::WSRPPutResourcePropertyDocumentRequest:

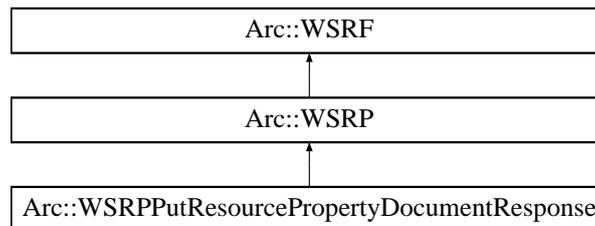


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.313 Arc::WSRPPutResourcePropertyDocumentResponse Class Reference

Inheritance diagram for Arc::WSRPPutResourcePropertyDocumentResponse:

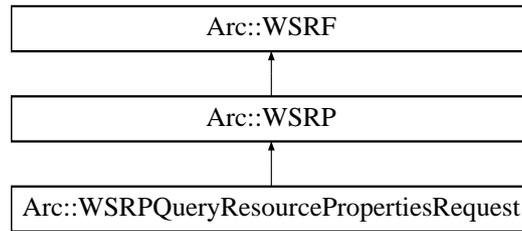


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.314 Arc::WSRPQueryResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPQueryResourcePropertiesRequest:

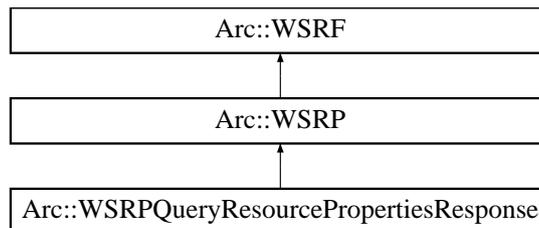


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.315 Arc::WSRPQueryResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPQueryResourcePropertiesResponse:



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

- WSResourceProperties.h

### 6.316 Arc::WSRPResourcePropertyChangeFailure Class Reference

```
#include <WSResourceProperties.h>
```

Inheritance diagram for Arc::WSRPResourcePropertyChangeFailure:



#### Public Member Functions

- **WSRPResourcePropertyChangeFailure** (SOAPEnvelope &soap)
- **WSRPResourcePropertyChangeFailure** (const std::string &type)

### 6.316.1 Detailed Description

Base class for WS-ResourceProperties faults which contain ResourcePropertyChangeFailure

### 6.316.2 Constructor & Destructor Documentation

#### 6.316.2.1 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure ( SOAPEnvelope & soap ) [inline]

Constructor - creates object out of supplied SOAP tree.

#### 6.316.2.2 Arc::WSRPResourcePropertyChangeFailure::WSRPResourcePropertyChangeFailure ( const std::string & type ) [inline]

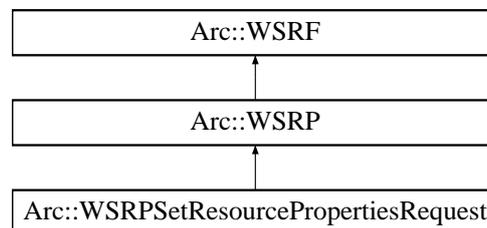
Constructor - creates new **WSRP** (p. 373) fault

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

- WSResourceProperties.h

## 6.317 Arc::WSRPSetResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPSetResourcePropertiesRequest:

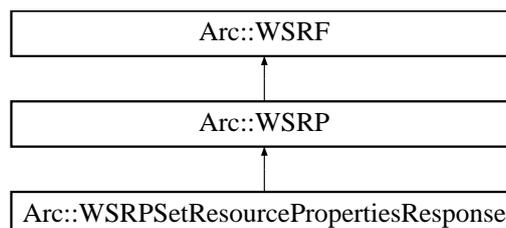


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.318 Arc::WSRPSetResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPSetResourcePropertiesResponse:

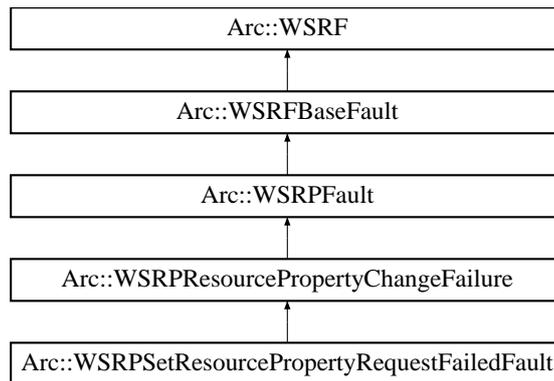


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.319 Arc::WSRPSetResourcePropertyRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPSetResourcePropertyRequestFailedFault:

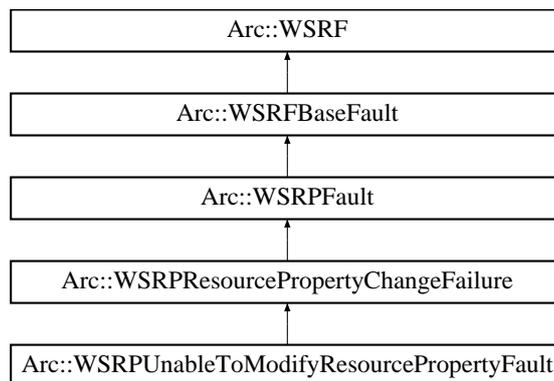


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.320 Arc::WSRPUnableToModifyResourcePropertyFault Class Reference

Inheritance diagram for Arc::WSRPUnableToModifyResourcePropertyFault:

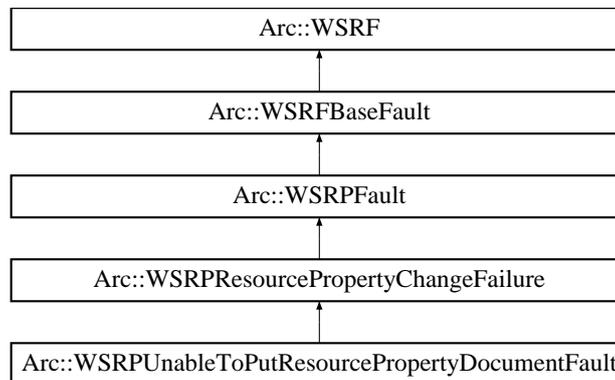


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.321 Arc::WSRPUnableToPutResourcePropertyDocumentFault Class Reference

Inheritance diagram for Arc::WSRPUnableToPutResourcePropertyDocumentFault:

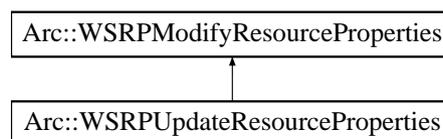


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.322 Arc::WSRPUUpdateResourceProperties Class Reference

Inheritance diagram for Arc::WSRPUUpdateResourceProperties:

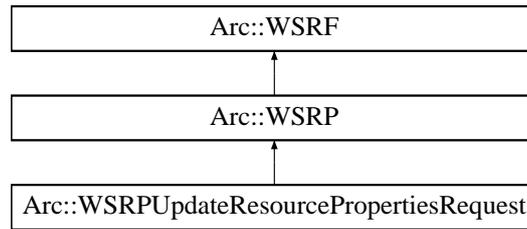


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.323 Arc::WSRPUUpdateResourcePropertiesRequest Class Reference

Inheritance diagram for Arc::WSRPUUpdateResourcePropertiesRequest:

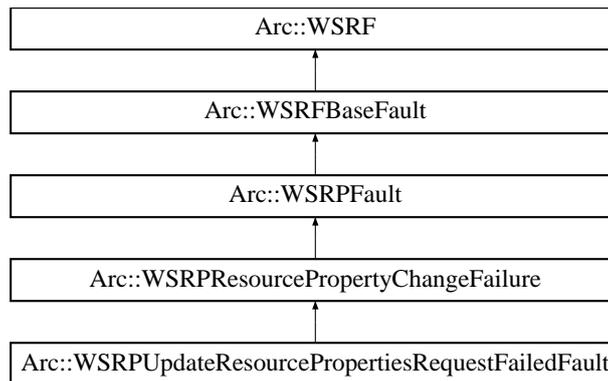


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.324 Arc::WSRPUUpdateResourcePropertiesRequestFailedFault Class Reference

Inheritance diagram for Arc::WSRPUUpdateResourcePropertiesRequestFailedFault:

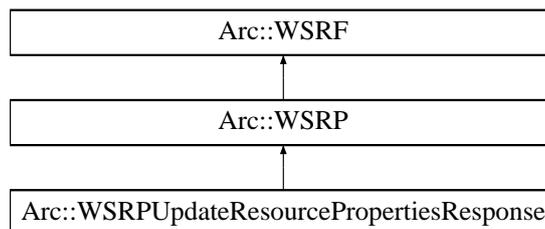


The documentation for this class was generated from the following file:

- WSResourceProperties.h

### 6.325 Arc::WSRPUUpdateResourcePropertiesResponse Class Reference

Inheritance diagram for Arc::WSRPUUpdateResourcePropertiesResponse:

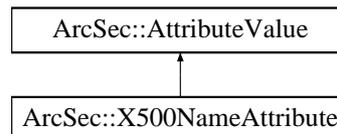


The documentation for this class was generated from the following file:

- WSResourceProperties.h

## 6.326 ArcSec::X500NameAttribute Class Reference

Inheritance diagram for ArcSec::X500NameAttribute:



### Public Member Functions

- virtual bool **equal** (AttributeValue \*other, bool check\_id=true)
- virtual std::string **encode** ()
- virtual std::string **getType** ()
- virtual std::string **getId** ()

### 6.326.1 Member Function Documentation

#### 6.326.1.1 virtual std::string ArcSec::X500NameAttribute::encode ( ) [inline, virtual]

encode the value in a string format

Implements ArcSec::AttributeValue (p. 57).

#### 6.326.1.2 virtual bool ArcSec::X500NameAttribute::equal ( AttributeValue \* value, bool check\_id = true ) [virtual]

Evaluate whether "this" equal to the parameter value

Implements ArcSec::AttributeValue (p. 58).

#### 6.326.1.3 virtual std::string ArcSec::X500NameAttribute::getId ( ) [inline, virtual]

Get the AttributeId of the <Attribute>

Implements ArcSec::AttributeValue (p. 58).

#### 6.326.1.4 virtual std::string ArcSec::X500NameAttribute::getType ( ) [inline, virtual]

Get the DataType of the <Attribute>

Implements ArcSec::AttributeValue (p. 58).

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

- X500NameAttribute.h

## 6.327 Arc::X509Token Class Reference

Class for manipulating X.509 Token **Profile** (p. 258).

```
#include <X509Token.h>
```

### Public Types

- enum **X509TokenType**

### Public Member Functions

- **X509Token** (SOAPEnvelope &soap, const std::string &keyfile="")
- **X509Token** (SOAPEnvelope &soap, const std::string &certfile, const std::string &keyfile, **X509TokenType** token\_type=Signature)
- ~**X509Token** (void)
- **operator bool** (void)
- bool **Authenticate** (const std::string &cafile, const std::string &capath)
- bool **Authenticate** (void)

### 6.327.1 Detailed Description

Class for manipulating X.509 Token **Profile** (p. 258). This class is for generating/consuming X.509 Token profile. Currently it is used by x509token handler (src/hed/pdc/x509tokensh/) It is not necessary to directly called this class. If we need to use X.509 Token functionality, we only need to configure the x509token handler into service and client.

### 6.327.2 Member Enumeration Documentation

#### 6.327.2.1 enum Arc::X509Token::X509TokenType

X509TokeType is for distinguishing two types of operation. It is used as the parameter of constructor.

### 6.327.3 Constructor & Destructor Documentation

#### 6.327.3.1 Arc::X509Token::X509Token ( SOAPEnvelope & soap, const std::string & keyfile = "" )

Constructor.Parse X509 Token information from SOAP header. X509 Token related information is extracted from SOAP header and stored in class variables. And then it the **X509Token** (p. 390) object will be used for authentication if the tokentype is Signature; otherwise if the tokentype is Encryption, the encrypted soap body will be decrypted and replaced by decrypted message. keyfile is only needed when the **X509Token** (p. 390) is encryption token

#### 6.327.3.2 Arc::X509Token::X509Token ( SOAPEnvelope & soap, const std::string & certfile, const std::string & keyfile, X509TokenType token\_type = Signature )

Constructor. Add X509 Token information into the SOAP header. Generated token contains elements X509 token and signature, and is meant to be used for authentication on the consuming side.

**Parameters**

- soap* The SOAP message to which the X509 Token will be inserted
- certfile* The certificate file which will be used to encrypt the SOAP body (if parameter tokentype is Encryption), or be used as <wsse:BinarySecurityToken/> (if parameter tokentype is Signature).
- keyfile* The key file which will be used to create signature. Not needed when create encryption.
- tokentype* Token type: Signature or Encryption.

**6.327.3.3 Arc::X509Token::~~X509Token ( void )**

Deconstructor. Nothing to be done except finalizing the xmlsec library.

**6.327.4 Member Function Documentation****6.327.4.1 bool Arc::X509Token::Authenticate ( const std::string & *cafile*, const std::string & *capath* )**

Check signature by using the certificate information in **X509Token** (p. 390) which is parsed by the constructor, and the trusted certificates specified as one of the two parameters. Not only the signature (in the **X509Token** (p. 390)) itself is checked, but also the certificate which is supposed to check the signature needs to be trusted (which means the certificate is issued by the ca certificate from CA file or CA directory). At least one the the two parameters should be set.

**Parameters**

- cafile* The CA file
- capath* The CA directory

**Returns**

true if authentication passes; otherwise false

**6.327.4.2 bool Arc::X509Token::Authenticate ( void )**

Check signature by using the cert information in soap message. Only the signature itself is checked, and it is not guranteed that the certificate which is supposed to check the signature is trusted.

**6.327.4.3 Arc::X509Token::operator bool ( void )**

Returns true of constructor succeeded

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

- X509Token.h

**6.328 Arc::XmlContainer Class Reference**

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

- XmlContainer.h

## 6.329 Arc::XmlDatabase Class Reference

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

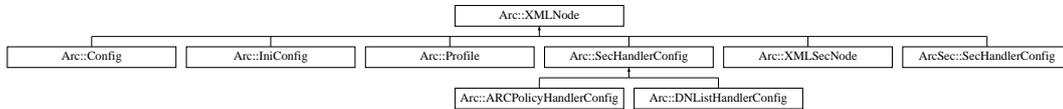
- XmlDatabase.h

## 6.330 Arc::XMLNode Class Reference

Wrapper for LibXML library Tree interface.

```
#include <XMLNode.h>
```

Inheritance diagram for Arc::XMLNode:



### Public Member Functions

- **XMLNode** (void)
- **XMLNode** (const **XMLNode** &node)
- **XMLNode** (const std::string &xml)
- **XMLNode** (const char \*xml, int len=-1)
- **XMLNode** (long ptr\_addr)
- **XMLNode** (const NS &ns, const char \*name)
- **~XMLNode** (void)
- void **New** (**XMLNode** &node) const
- void **Exchange** (**XMLNode** &node)
- void **Move** (**XMLNode** &node)
- void **Swap** (**XMLNode** &node)
- **operator bool** (void) const
- bool **operator!** (void) const
- bool **operator==** (const **XMLNode** &node)
- bool **operator!=** (const **XMLNode** &node)
- bool **Same** (const **XMLNode** &node)
- bool **operator==** (bool val)
- bool **operator!=** (bool val)
- bool **operator==** (const std::string &str)
- bool **operator!=** (const std::string &str)
- bool **operator==** (const char \*str)
- bool **operator!=** (const char \*str)
- **XMLNode Child** (int n=0)
- **XMLNode operator[]** (const char \*name) const
- **XMLNode operator[]** (const std::string &name) const
- **XMLNode operator[]** (int n) const
- void **operator++** (void)
- void **operator--** (void)

- int **Size** (void) const
- **XMLNode Get** (const std::string &name) const
- std::string **Name** (void) const
- std::string **Prefix** (void) const
- std::string **FullName** (void) const
- std::string **Namespace** (void) const
- void **Name** (const char \*name)
- void **Name** (const std::string &name)
- void **GetXML** (std::string &out\_xml\_str, bool user\_friendly=false) const
- void **GetXML** (std::string &out\_xml\_str, const std::string &encoding, bool user\_friendly=false) const
- void **GetDoc** (std::string &out\_xml\_str, bool user\_friendly=false) const
- **operator std::string** (void) const
- **XMLNode & operator=** (const char \*content)
- **XMLNode & operator=** (const std::string &content)
- void **Set** (const std::string &content)
- **XMLNode & operator=** (const **XMLNode** &node)
- **XMLNode Attribute** (int n=0)
- **XMLNode Attribute** (const char \*name)
- **XMLNode Attribute** (const std::string &name)
- **XMLNode NewAttribute** (const char \*name)
- **XMLNode NewAttribute** (const std::string &name)
- int **AttributesSize** (void) const
- void **Namespaces** (const **NS** &namespaces, bool keep=false, int recursion=-1)
- **NS Namespaces** (void)
- std::string **NamespacePrefix** (const char \*urn)
- **XMLNode NewChild** (const char \*name, int n=-1, bool global\_order=false)
- **XMLNode NewChild** (const std::string &name, int n=-1, bool global\_order=false)
- **XMLNode NewChild** (const char \*name, const **NS** &namespaces, int n=-1, bool global\_order=false)
- **XMLNode NewChild** (const std::string &name, const **NS** &namespaces, int n=-1, bool global\_order=false)
- **XMLNode NewChild** (const **XMLNode** &node, int n=-1, bool global\_order=false)
- void **Replace** (const **XMLNode** &node)
- void **Destroy** (void)
- **XMLNodeList Path** (const std::string &path)
- **XMLNodeList XPathLookup** (const std::string &xpathExpr, const **NS** &nsList)
- **XMLNode GetRoot** (void)
- **XMLNode Parent** (void)
- bool **SaveToFile** (const std::string &file\_name) const
- bool **SaveToStream** (std::ostream &out) const
- bool **ReadFromFile** (const std::string &file\_name)
- bool **ReadFromStream** (std::istream &in)
- bool **Validate** (const std::string &schema\_file, std::string &err\_msg)

## Protected Member Functions

- **XMLNode** (xmlNodePtr node)

## Protected Attributes

- bool `is_owner_`
- bool `is_temporary_`

## Friends

- bool `MatchXMLName` (const `XMLNode` &node1, const `XMLNode` &node2)
- bool `MatchXMLName` (const `XMLNode` &node, const char \*name)
- bool `MatchXMLName` (const `XMLNode` &node, const std::string &name)
- bool `MatchXMLNamespace` (const `XMLNode` &node1, const `XMLNode` &node2)
- bool `MatchXMLNamespace` (const `XMLNode` &node, const char \*uri)
- bool `MatchXMLNamespace` (const `XMLNode` &node, const std::string &uri)

### 6.330.1 Detailed Description

Wrapper for LibXML library Tree interface. This class wraps XML Node, Document and Property/Attribute structures. Each instance serves as pointer to actual LibXML element and provides convenient (for chosen purpose) methods for manipulating it. This class has no special ties to LibXML library and may be easily rewritten for any XML parser which provides interface similar to LibXML Tree. It implements only small subset of XML capabilities, which is probably enough for performing most of useful actions. This class also filters out (usually) useless textual nodes which are often used to make XML documents human-readable.

### 6.330.2 Constructor & Destructor Documentation

#### 6.330.2.1 `Arc::XMLNode::XMLNode ( xmlNodePtr node ) [inline, protected]`

Private constructor for inherited classes Creates instance and links to existing LibXML structure. Acquired structure is not owned by class instance. If there is need to completely pass control of LibXML document to then instance's `is_owner_` variable has to be set to true.

#### 6.330.2.2 `Arc::XMLNode::XMLNode ( void ) [inline]`

Constructor of invalid node Created instance does not point to XML element. All methods are still allowed for such instance but produce no results.

#### 6.330.2.3 `Arc::XMLNode::XMLNode ( const XMLNode & node ) [inline]`

Copies existing instance. Underlying XML element is NOT copied. Ownership is NOT inherited. Strictly speaking it should be no const here - but that conflicts with C++.

#### 6.330.2.4 `Arc::XMLNode::XMLNode ( const std::string & xml )`

Creates XML document structure from textual representation of XML document. Created structure is pointed and owned by constructed instance

**6.330.2.5 Arc::XMLNode::XMLNode ( const char \* *xml*, int *len* = -1 )**

Same as previous

**6.330.2.6 Arc::XMLNode::XMLNode ( long *ptr\_addr* )**

Copy constructor. Used by language bindings

**6.330.2.7 Arc::XMLNode::XMLNode ( const NS & *ns*, const char \* *name* )**

Creates empty XML document structure with specified namespaces. Created XML contains only root element named 'name'. Created structure is pointed and owned by constructed instance

**6.330.2.8 Arc::XMLNode::~~XMLNode ( void )**

Destructor Also destroys underlying XML document if owned by this instance

**6.330.3 Member Function Documentation****6.330.3.1 XMLNode Arc::XMLNode::Attribute ( int *n* = 0 )**

Returns list of all attributes of node.

Returns **XMLNode** (p. 392) instance representing n-th attribute of node.

Referenced by Attribute().

**6.330.3.2 XMLNode Arc::XMLNode::Attribute ( const char \* *name* )**

Returns **XMLNode** (p. 392) instance representing first attribute of node with specified by name

**6.330.3.3 XMLNode Arc::XMLNode::Attribute ( const std::string & *name* ) [inline]**

Returns **XMLNode** (p. 392) instance representing first attribute of node with specified by name

References Attribute().

**6.330.3.4 int Arc::XMLNode::AttributesSize ( void ) const**

Returns number of attributes of node

**6.330.3.5 XMLNode Arc::XMLNode::Child ( int *n* = 0 )**

Returns **XMLNode** (p. 392) instance representing n-th child of XML element. If such does not exist invalid **XMLNode** (p. 392) instance is returned

**6.330.3.6 void Arc::XMLNode::Destroy ( void )**

Destroys underlying XML element. XML element is unlinked from XML tree and destroyed. After this operation `XMLNode` (p. 392) instance becomes invalid

**6.330.3.7 void Arc::XMLNode::Exchange ( XMLNode & node )**

Exchanges XML (sub)trees. Following combinations are possible If either this or node are referring owned XML tree (top level node) then references are simply exchanged. This operation is fast. If both this and node are referring to XML (sub)tree of different documents then (sub)trees are exchanged between documents. If both this and node are referring to XML (sub)tree of same document then (sub)trees are moved inside document. The main reason for this method is to provide effective way to insert one XML document inside another. One should take into account that if any of exchanged nodes is top level it must be also owner of document. Otherwise method will fail. If both nodes are top level owners and/or invalid nodes then this method is identical to `Swap()` (p. 402).

**6.330.3.8 std::string Arc::XMLNode::FullName ( void ) const [inline]**

Returns prefix:name of XML node

References `Name()`, and `Prefix()`.

**6.330.3.9 XMLNode Arc::XMLNode::Get ( const std::string & name ) const [inline]**

Same as `operator[]`

References `operator[]()`.

**6.330.3.10 void Arc::XMLNode::GetDoc ( std::string & out\_xml\_str, bool user\_friendly = false ) const**

Fills argument with whole XML document textual representation

**6.330.3.11 XMLNode Arc::XMLNode::GetRoot ( void )**

Get the root node from any child node of the tree

**6.330.3.12 void Arc::XMLNode::GetXML ( std::string & out\_xml\_str, bool user\_friendly = false ) const**

Fills argument with this instance XML subtree textual representation

**6.330.3.13 void Arc::XMLNode::GetXML ( std::string & out\_xml\_str, const std::string & encoding, bool user\_friendly = false ) const**

Fills argument with this instance XML subtree textual representation if the XML subtree is corresponding to the encoding format specified in the argument, e.g. utf-8

**6.330.3.14 void Arc::XMLNode::Move ( XMLNode & node )**

Moves content of this XML (sub)tree to node This operation is similar to New except that XML (sub)tree to referred by this is destroyed. This method is more effective than combination of New() (p. 397) and Destroy() (p. 396) because internally it is optimized not to copy data if not needed. The main purpose of this is to effectively extract part of XML document.

**6.330.3.15 std::string Arc::XMLNode::Name ( void ) const**

Returns name of XML node

Referenced by FullName(), and Name().

**6.330.3.16 void Arc::XMLNode::Name ( const std::string & name ) [inline]**

Assigns new name to XML node

References Name().

**6.330.3.17 void Arc::XMLNode::Name ( const char \* name )**

Assigns new name to XML node

**6.330.3.18 std::string Arc::XMLNode::Namespace ( void ) const**

Returns namespace URI of XML node

**6.330.3.19 std::string Arc::XMLNode::NamespacePrefix ( const char \* urn )**

Returns prefix of specified namespace. Empty string if no such namespace.

**6.330.3.20 NS Arc::XMLNode::Namespaces ( void )**

Returns namespaces known at this node

**6.330.3.21 void Arc::XMLNode::Namespaces ( const NS & namespaces, bool keep = false, int recursion = -1 )**

Assigns namespaces of XML document at point specified by this instance. If namespace already exists it gets new prefix. New namespaces are added. It is useful to apply this method to XML being processed in order to refer to it's elements by known prefix. If keep is set to false existing namespace definition residing at this instance and below are removed (default behavior). If recursion is set to positive number then depth of prefix replacement is limited by this number (0 limits it to this node only). For unlimited recursion use -1. If recursion is limited then value of keep is ignored and existing namespaces are always kept.

**6.330.3.22 void Arc::XMLNode::New ( XMLNode & node ) const**

Creates a copy of XML (sub)tree. If object does not represent whole document - top level document is created. 'new\_node' becomes a pointer owning new XML document.

**6.330.3.23 XMLNode Arc::XMLNode::NewAttribute ( const char \* name )**

Creates new attribute with specified name.

Referenced by NewAttribute().

**6.330.3.24 XMLNode Arc::XMLNode::NewAttribute ( const std::string & name ) [inline]**

Creates new attribute with specified name.

References NewAttribute().

**6.330.3.25 XMLNode Arc::XMLNode::NewChild ( const char \* name, int n = -1, bool global\_order = false )**

Creates new child XML element at specified position with specified name. Default is to put it at end of list. If global order is true position applies to whole set of children, otherwise only to children of same name. Returns created node.

Referenced by NewChild().

**6.330.3.26 XMLNode Arc::XMLNode::NewChild ( const std::string & name, int n = -1, bool global\_order = false ) [inline]**

Same as NewChild(const char\*,int,bool) (p. 398)

References NewChild().

**6.330.3.27 XMLNode Arc::XMLNode::NewChild ( const char \* name, const NS & namespaces, int n = -1, bool global\_order = false )**

Creates new child XML element at specified position with specified name and namespaces. For more information look at NewChild(const char\*,int,bool) (p. 398)

**6.330.3.28 XMLNode Arc::XMLNode::NewChild ( const std::string & name, const NS & namespaces, int n = -1, bool global\_order = false ) [inline]**

Same as NewChild(const char\*,const NS&,int,bool) (p. 398)

References NewChild().

**6.330.3.29 XMLNode Arc::XMLNode::NewChild ( const XMLNode & node, int n = -1, bool global\_order = false )**

Link a copy of supplied XML node as child. Returns instance referring to new child. XML element is a copy of supplied one but not owned by returned instance

**6.330.3.30 Arc::XMLNode::operator bool ( void ) const [inline]**

Returns true if instance points to XML element - valid instance

References is\_temporary\_.

**6.330.3.31** `Arc::XMLNode::operator std::string ( void ) const`

Returns textual content of node excluding content of children nodes

**6.330.3.32** `bool Arc::XMLNode::operator! ( void ) const [inline]`

Returns true if instance does not point to XML element - invalid instance

References `is_temporary_`.

**6.330.3.33** `bool Arc::XMLNode::operator!= ( const XMLNode & node ) [inline]`

Returns false if 'node' represents same XML element

**6.330.3.34** `bool Arc::XMLNode::operator!= ( bool val ) [inline]`

This operator is needed to avoid ambiguity

**6.330.3.35** `bool Arc::XMLNode::operator!= ( const std::string & str ) [inline]`

This operator is needed to avoid ambiguity

**6.330.3.36** `bool Arc::XMLNode::operator!= ( const char * str ) [inline]`

This operator is needed to avoid ambiguity

**6.330.3.37** `void Arc::XMLNode::operator++ ( void )`

Convenience operator to switch to next element of same name. If there is no such node this object becomes invalid.

**6.330.3.38** `void Arc::XMLNode::operator-- ( void )`

Convenience operator to switch to previous element of same name. If there is no such node this object becomes invalid.

**6.330.3.39** `XMLNode& Arc::XMLNode::operator= ( const char * content )`

Sets textual content of node. All existing children nodes are discarded.

Referenced by `operator=()`, and `Set()`.

**6.330.3.40** `XMLNode& Arc::XMLNode::operator= ( const XMLNode & node )`

Make instance refer to another XML node. Ownership is not inherited. Due to nature of `XMLNode` (p. 392) there should be no `const` here, but that does not fit into C++.

**6.330.3.41 XMLNode& Arc::XMLNode::operator=( const std::string & content ) [inline]**

Sets textual content of node. All existing children nodes are discarded.

References operator=().

**6.330.3.42 bool Arc::XMLNode::operator==( bool val ) [inline]**

This operator is needed to avoid ambiguity

**6.330.3.43 bool Arc::XMLNode::operator==( const XMLNode & node ) [inline]**

Returns true if 'node' represents same XML element

Referenced by Same().

**6.330.3.44 bool Arc::XMLNode::operator==( const char \* str ) [inline]**

This operator is needed to avoid ambiguity

**6.330.3.45 bool Arc::XMLNode::operator==( const std::string & str ) [inline]**

This operator is needed to avoid ambiguity

**6.330.3.46 XMLNode Arc::XMLNode::operator[]( const char \* name ) const**

Returns **XMLNode** (p. 392) instance representing first child element with specified name. Name may be "namespace\_prefix:name" or simply "name". In last case namespace is ignored. If such node does not exist invalid **XMLNode** (p. 392) instance is returned. This method should not be marked const because obtaining unrestricted **XMLNode** (p. 392) of child element allows modification of underlying XML tree. But in order to keep const in other places non-const-handling is passed to programmer. Otherwise C++ compiler goes nuts.

Referenced by Get(), and operator[ ]().

**6.330.3.47 XMLNode Arc::XMLNode::operator[]( const std::string & name ) const [inline]**

Similar to previous method

References operator[ ]().

**6.330.3.48 XMLNode Arc::XMLNode::operator[]( int n ) const**

Returns **XMLNode** (p. 392) instance representing n-th node in sequence of siblings of same name. It's main purpose is to be used to retrieve element in array of children of same name like node["name"][5]. This method should not be marked const because obtaining unrestricted **XMLNode** (p. 392) of child element allows modification of underlying XML tree. But in order to keep const in other places non-const-handling is passed to programmer. Otherwise C++ compiler goes nuts.

**6.330.3.49 XMLNode Arc::XMLNode::Parent ( void )**

Get the parent node from any child node of the tree

**6.330.3.50 XMLNodeList Arc::XMLNode::Path ( const std::string & path )**

Collects nodes corresponding to specified path. This is a convenience function to cover common use of XPath but without performance hit. Path is made of node\_name[/node\_name[...]] and is relative to current node. node\_names are treated in same way as in operator[]. Returns all nodes which are represented by path.

**6.330.3.51 std::string Arc::XMLNode::Prefix ( void ) const**

Returns namespace prefix of XML node

Referenced by FullName().

**6.330.3.52 bool Arc::XMLNode::ReadFromFile ( const std::string & file\_name )**

Read XML document from file and associate it with this node

**6.330.3.53 bool Arc::XMLNode::ReadFromStream ( std::istream & in )**

Read XML document from stream and associate it with this node

**6.330.3.54 void Arc::XMLNode::Replace ( const XMLNode & node )**

Makes a copy of supplied XML node and makes this instance refer to it

**6.330.3.55 bool Arc::XMLNode::Same ( const XMLNode & node ) [inline]**

Returns true if 'node' represents same XML element - for bindings

References operator==().

**6.330.3.56 bool Arc::XMLNode::SaveToFile ( const std::string & file\_name ) const**

Save string representation of node to file

**6.330.3.57 bool Arc::XMLNode::SaveToStream ( std::ostream & out ) const**

Save string representation of node to stream

**6.330.3.58 void Arc::XMLNode::Set ( const std::string & content ) [inline]**

Same as operator=. Used for bindings.

References operator=().

**6.330.3.59 int Arc::XMLNode::Size ( void ) const**

Returns number of children nodes

**6.330.3.60 void Arc::XMLNode::Swap ( XMLNode & node )**

Swaps XML (sub)trees to this this and node refer. For XML subtrees this method is not anyhow different then using combinaion **XMLNode** (p. 392) tmp=\*this; \*this=node; node=tmp; But in case of either this or node owning XML document ownership is swapped too. And this is a main purpose of **Swap()** (p. 402) method.

**6.330.3.61 bool Arc::XMLNode::Validate ( const std::string & schema\_file, std::string & err\_msg )**

Remove all eye-candy information leaving only informational parts \* void Purify(void); XML schema validation against the schema file defined as argument

**6.330.3.62 XMLNodeList Arc::XMLNode::XPathLookup ( const std::string & xpathExpr, const NS & nsList )**

Uses xPath to look up the whole xml structure, Returns a list of **XMLNode** (p. 392) points. The xpathExpr should be like "//xx:child1/" which indicates the namespace and node that you would like to find; The nsList is the namespace the result should belong to (e.g. xx="uri:test"). **Query** (p. 259) is run on whole XML document but only the elements belonging to this XML subtree are returned.

**6.330.4 Friends And Related Function Documentation****6.330.4.1 bool MatchXMLName ( const XMLNode & node1, const XMLNode & node2 ) [friend]**

Returns true if underlying XML elements have same names

**6.330.4.2 bool MatchXMLName ( const XMLNode & node, const std::string & name ) [friend]**

Returns true if 'name' matches name of 'node'. If name contains prefix it's checked too

**6.330.4.3 bool MatchXMLName ( const XMLNode & node, const char \* name ) [friend]**

Returns true if 'name' matches name of 'node'. If name contains prefix it's checked too

**6.330.4.4 bool MatchXMLNamespace ( const XMLNode & node1, const XMLNode & node2 ) [friend]**

Returns true if underlying XML elements belong to same namespaces

#### 6.330.4.5 bool MatchXMLNamespace ( const XMLNode & *node*, const std::string & *uri* ) [friend]

Returns true if 'namespace' matches 'node's namespace.

#### 6.330.4.6 bool MatchXMLNamespace ( const XMLNode & *node*, const char \* *uri* ) [friend]

Returns true if 'namespace' matches 'node's namespace.

### 6.330.5 Field Documentation

#### 6.330.5.1 bool Arc::XMLNode::is\_owner\_ [protected]

If true node is owned by this instance - hence released in destructor. Normally that may be true only for top level node of XML document.

#### 6.330.5.2 bool Arc::XMLNode::is\_temporary\_ [protected]

This variable is for future

Referenced by operator bool(), and operator!().

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

- XMLNode.h

## 6.331 Arc::XMLNodeContainer Class Reference

```
#include <XMLNode.h>
```

### Public Member Functions

- **XMLNodeContainer** (void)
- **XMLNodeContainer** (const XMLNodeContainer &)
- **XMLNodeContainer & operator=** (const XMLNodeContainer &)
- void **Add** (const XMLNode &)
- void **Add** (const std::list< XMLNode > &)
- void **AddNew** (const XMLNode &)
- void **AddNew** (const std::list< XMLNode > &)
- int **Size** (void) const
- XMLNode **operator[]** (int)
- std::list< XMLNode > **Nodes** (void)

### 6.331.1 Detailed Description

Container for multiple XMLNode (p. 392) elements

## 6.331.2 Constructor & Destructor Documentation

### 6.331.2.1 `Arc::XMLNodeContainer::XMLNodeContainer ( void )`

Default constructor

### 6.331.2.2 `Arc::XMLNodeContainer::XMLNodeContainer ( const XMLNodeContainer & )`

Copy constructor. Add nodes from argument. Nodes owning XML document are copied using `AddNew()` (p. 404). Not owning nodes are linked using `Add()` (p. 404) method.

## 6.331.3 Member Function Documentation

### 6.331.3.1 `void Arc::XMLNodeContainer::Add ( const XMLNode & )`

Link XML subtree referred by node to container. XML tree must be available as long as this object is used.

### 6.331.3.2 `void Arc::XMLNodeContainer::Add ( const std::list< XMLNode > & )`

Link multiple XML subtrees to container.

### 6.331.3.3 `void Arc::XMLNodeContainer::AddNew ( const XMLNode & )`

Copy XML subtree referenced by node to container. After this operation container refers to independent XML document. This document is deleted when container is destroyed.

### 6.331.3.4 `void Arc::XMLNodeContainer::AddNew ( const std::list< XMLNode > & )`

Copy multiple XML subtrees to container.

### 6.331.3.5 `std::list<XMLNode> Arc::XMLNodeContainer::Nodes ( void )`

Returns all stored nodes.

### 6.331.3.6 `XMLNodeContainer& Arc::XMLNodeContainer::operator= ( const XMLNodeContainer & )`

Same as copy constructor with current nodes being deleted first.

### 6.331.3.7 `XMLNode Arc::XMLNodeContainer::operator[] ( int )`

Returns n-th node in a store.

### 6.331.3.8 `int Arc::XMLNodeContainer::Size ( void ) const`

Return number of referred/stored nodes.

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

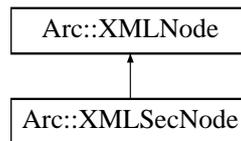
- XMLNode.h

## 6.332 Arc::XMLSecNode Class Reference

Extends **XMLNode** (p. 392) class to support XML security operation.

```
#include <XMLSecNode.h>
```

Inheritance diagram for Arc::XMLSecNode:



### Public Member Functions

- **XMLSecNode** (**XMLNode** &node)
- void **AddSignatureTemplate** (const std::string &id\_name, const SignatureMethod sign\_method, const std::string &incl\_namespaces="")
- bool **SignNode** (const std::string &privkey\_file, const std::string &cert\_file)
- bool **VerifyNode** (const std::string &id\_name, const std::string &ca\_file, const std::string &ca\_path, bool verify\_trusted=true)
- bool **EncryptNode** (const std::string &cert\_file, const SymEncryptionType encript\_type)
- bool **DecryptNode** (const std::string &privkey\_file, **XMLNode** &decrypted\_node)

#### 6.332.1 Detailed Description

Extends **XMLNode** (p. 392) class to support XML security operation. All **XMLNode** (p. 392) methods are exposed by inheriting from **XMLNode** (p. 392). **XMLSecNode** (p. 405) itself does not own node, instead it uses the node from the base class **XMLNode** (p. 392).

#### 6.332.2 Constructor & Destructor Documentation

##### 6.332.2.1 Arc::XMLSecNode::XMLSecNode ( XMLNode & node )

Create a object based on an **XMLNode** (p. 392) instance.

#### 6.332.3 Member Function Documentation

##### 6.332.3.1 void Arc::XMLSecNode::AddSignatureTemplate ( const std::string & id\_name, const SignatureMethod sign\_method, const std::string & incl\_namespaces = " " )

Add the signature template for later signing.

#### Parameters

*id\_name* The identifier name under this node which will be used for the <Signature> to refer to.

*sign\_method* The sign method for signing. Two options now, RSA\_SHA1, DSA\_SHA1

### 6.332.3.2 `bool Arc::XMLSecNode::DecryptNode ( const std::string & privkey_file, XMLNode & decrypted_node )`

Decrypt the `<xenc:EncryptedData/>` under this node, the decrypted node will be output in the second argument of `DecryptNode` method. And the `<xenc:EncryptedData/>` under this node will be removed after decryption.

#### Parameters

*privkey\_file* The private key file, which is used for decrypting

*decrypted\_node* Output the decrypted node

### 6.332.3.3 `bool Arc::XMLSecNode::EncryptNode ( const std::string & cert_file, const SymEncryptionType encrypt_type )`

Encrypt this node, after encryption, this node will be replaced by the encrypted node

#### Parameters

*cert\_file* The certificate file, the public key parsed from this certificate is used to encrypted the symmetric key, and then the symmetric key is used to encrypted the node

*encrypt\_type* The encryption type when encrypting the node, four option in `SymEncryptionType`

*verify\_trusted* Verify trusted certificates or not. If set to false, then only the signature will be checked (by using the public key from `KeyInfo`).

### 6.332.3.4 `bool Arc::XMLSecNode::SignNode ( const std::string & privkey_file, const std::string & cert_file )`

Sign this node (identified by `id_name`).

#### Parameters

*privkey\_file* The private key file. The private key is used for signing

*cert\_file* The certificate file. The certificate is used as the `<KeyInfo>` part of the `<Signature>`; `<KeyInfo>` will be used for the other end to verify this `<Signature>`

*incl\_namespaces* `InclusiveNamespaces` for Transform in Signature

### 6.332.3.5 `bool Arc::XMLSecNode::VerifyNode ( const std::string & id_name, const std::string & ca_file, const std::string & ca_path, bool verify_trusted = true )`

Verify the signature under this node

#### Parameters

*id\_name* The id of this node, which is used for identifying the node

*ca\_file* The CA file which used as trusted certificate when verify the certificate in the `<KeyInfo>` part of `<Signature>`

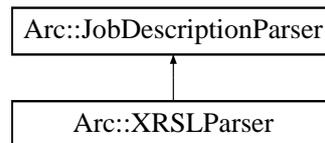
*ca\_path* The CA directory; either *ca\_file* or *ca\_path* should be set.

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

- XMLSecNode.h

### 6.333 Arc::XRSLParser Class Reference

Inheritance diagram for Arc::XRSLParser:



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

- XRSLParser.h



# Chapter 7

## File Documentation

### 7.1 URL.h File Reference

Class to hold general URL's.

```
#include <iostream>
#include <list>
#include <map>
#include <string>
```

#### Data Structures

- class **Arc::URL**
- class **Arc::URLLocation**  
*Class to hold a resolved **URL** (p. 323) location.*
- class **Arc::PathIterator**  
*Class to iterate through elements of path.*

#### Namespaces

- namespace **Arc**

#### Defines

- `#define RC_DEFAULT_PORT 389`

#### Functions

- `std::list< URL > Arc::ReadURLList (const URL &urllist)`

### 7.1.1 Detailed Description

Class to hold general URL's. The URL is split into protocol, hostname, port and path. This class tries to follow RFC 3986 for splitting URLs at least for protocol + host part. It also accepts local file paths which are converted to `file:path`. Usual system dependant file paths are supported. Relative paths are converted to absolute ones by prepending them with current working directory path. File path can't start from # symbol (why?). If string representation of URL starts from '@' then it is treated as path to file containing list of URLs. Simple URL is parsed in following way: `[protocol:][//[username:passwd@][host][:port]][;urloptions[;...]][/path[?httpoption[&...]][;metadataoption[...]]]` The 'protocol' and 'host' parts are treated as case-insensitive and to avoid confusion are converted to lowercase in constructor. Note that 'path' is always converted to absolute path in constructor. Meaning of 'absolute' may depend upon URL type. For generic URL and local POSIX file paths that means path starts from / like `/path/to/file` For Windows paths absolute path may look like `C:` It is important to note that path still can be empty. For referencing local file using absolute path on POSIX filesystem one may use either `file:///path/to/file` or `file:/path/to/file` Relative path will look like `file:to/file` For local Windows files possible URLs are `file:C:file:to` URLs representing LDAP resources have different structure of options following 'path' part `ldap://host[:port][;urloptions[;...]][/path[?attributes[?scope[?filter]]]]` For LDAP URLs paths are converted from `/key1=value1/.../keyN=valueN` notation to `keyN=valueN,...,key1=value1` and hence path does not contain leading /. If LDAP URL initially had path in second notation leading / is treated as separator only and is stripped. URLs of indexing services optionally may have locations specified before 'host' part `protocol://[location[;location[;...]]@[host][:port]...` The structure of 'location' element is protocol specific.

### 7.1.2 Define Documentation

#### 7.1.2.1 #define RC\_DEFAULT\_PORT 389

Default ports for different protocols

# Index

- ~BrokerLoader
    - Arc::BrokerLoader, 64
  - ~Counter
    - Arc::Counter, 83
  - ~Database
    - Arc::Database, 99
  - ~IntraProcessCounter
    - Arc::IntraProcessCounter, 180
  - ~JobControllerLoader
    - Arc::JobControllerLoader, 187
  - ~Loader
    - Arc::Loader, 191
  - ~Logger
    - Arc::Logger, 196
  - ~MCCLoader
    - Arc::MCCLoader, 209
  - ~Message
    - Arc::Message, 211
  - ~PayloadRaw
    - Arc::PayloadRaw, 230
  - ~PayloadStream
    - Arc::PayloadStream, 236
  - ~Plexer
    - Arc::Plexer, 248
  - ~Query
    - Arc::Query, 260
  - ~Run
    - Arc::Run, 271
  - ~SAMLToken
    - Arc::SAMLToken, 279
  - ~SOAPMessage
    - Arc::SOAPMessage, 289
  - ~SubmitterLoader
    - Arc::SubmitterLoader, 309
  - ~TargetRetrieverLoader
    - Arc::TargetRetrieverLoader, 315
  - ~URL
    - Arc::URL, 325
  - ~URLLocation
    - Arc::URLLocation, 332
  - ~WSAEndpointReference
    - Arc::WSAEndpointReference, 365
  - ~X509Token
    - Arc::X509Token, 391
  - ~XMLNode
    - Arc::XMLNode, 395
- Abandon
    - Arc::Run, 272
  - ACCESS\_LATENCY\_LARGE
    - Arc::DataPoint, 111
  - ACCESS\_LATENCY\_SMALL
    - Arc::DataPoint, 111
  - ACCESS\_LATENCY\_ZERO
    - Arc::DataPoint, 111
  - Acquire
    - Arc::DelegationConsumer, 137
    - Arc::InformationContainer, 175
  - acquireDelegation
    - Arc::ClientX509Delegation, 74
  - Action
    - Arc::WSAHeader, 367
  - Add
    - Arc::MessageContext, 217
    - Arc::XMLNodeContainer, 404
  - add
    - Arc::DataBuffer, 102
    - Arc::MessageAttributes, 213
    - Arc::SoftwareRequirement, 299, 300
  - AddBartender
    - Arc::UserConfig, 338
  - AddCADir
    - Arc::BaseConfig, 60
  - AddCAFile
    - Arc::BaseConfig, 60
  - AddCertExtObj
    - Arc::Credential, 92
  - AddCertificate
    - Arc::BaseConfig, 60
  - AddChain
    - Arc::VOMSTrustList, 364
  - AddChecksumObject
    - Arc::DataPoint, 112
    - Arc::DataPointDirect, 119
    - Arc::DataPointIndex, 124
  - addDestination
    - Arc::Logger, 196
  - AddDN
    - Arc::FileCache, 161
  - AddExtension

- Arc::Credential, 92
- AddIndexServer
  - Arc::TargetGenerator, 311
- AddJob
  - Arc::TargetGenerator, 311
- AddLDAPAttribute
  - Arc::URL, 325
- AddLocation
  - Arc::DataPoint, 112
  - Arc::DataPointDirect, 119
  - Arc::DataPointIndex, 124
- AddMetaDataOption
  - Arc::URL, 325
- AddNew
  - Arc::XMLNodeContainer, 404
- AddOption
  - Arc::URL, 325
- AddOverlay
  - Arc::BaseConfig, 61
- AddPluginsPath
  - Arc::BaseConfig, 61
- addPolicy
  - ArcSec::Evaluator, 150
  - ArcSec::Policy, 255
- AddPrivateKey
  - Arc::BaseConfig, 61
- AddProxy
  - Arc::BaseConfig, 61
- AddRegex
  - Arc::VOMSTrustList, 364
- addRegistrar
  - Arc::InfoRegisterContainer, 172
- addRequestItem
  - ArcSec::Request, 264
- Address
  - Arc::WSAEndpointReference, 365
- AddSecHandler
  - Arc::ClientSOAP, 72
  - Arc::MCC, 203
  - Arc::Service, 286
- AddService
  - Arc::TargetGenerator, 312
- addService
  - Arc::InfoRegisterContainer, 172
  - Arc::InfoRegistrar, 174
- AddServices
  - Arc::UserConfig, 338
- AddSignatureTemplate
  - Arc::XMLSecNode, 405
- AddTarget
  - Arc::TargetGenerator, 312
- addVOMSAC
  - Arc, 37
- AfterFork
  - Arc::Run, 272
- allocated
  - Arc::PayloadRawBuf, 232
- allocated\_
  - Arc::WSRF, 371
- ApplicationEnvironments
  - Arc::ExecutionTarget, 156
- ApplyToConfig
  - Arc::UserConfig, 339
- approveCSR
  - Arc::OAuthConsumer, 226
  - Arc::SAML2SSOHTTPClient, 276
- Arc, 23
  - addVOMSAC, 37
  - AttrConstIter, 36
  - AttrIter, 36
  - AttrMap, 36
  - BUSY\_ERROR, 37
  - ContentFromPayload, 37
  - CreateThreadFunction, 37
  - createVOMSAC, 37
  - CredentialLogger, 43
  - FileOpen, 38
  - final\_xmlsec, 38
  - GENERIC\_ERROR, 36
  - get\_cert\_str, 38
  - get\_key\_from\_certfile, 38
  - get\_key\_from\_certstr, 38
  - get\_key\_from\_keyfile, 38
  - get\_key\_from\_keystr, 38
  - get\_node, 38
  - get\_plugin\_instance, 36
  - get\_property, 38
  - GUID, 39
  - init\_xmlsec, 39
  - istring\_to\_level, 39
  - load\_key\_from\_certfile, 39
  - load\_key\_from\_certstr, 39
  - load\_key\_from\_keyfile, 39
  - load\_trusted\_cert\_file, 39
  - load\_trusted\_cert\_str, 40
  - load\_trusted\_certs, 40
  - LogLevel, 36
  - MatchXMLName, 40
  - MatchXMLNamespace, 40
  - OpenSSLInit, 40
  - operator<<, 40, 41
  - parseVOMSAC, 41, 42
  - PARSING\_ERROR, 37
  - passphrase\_callback, 42
  - plugins\_table\_name, 43
  - PROTOCOL\_RECOGNIZED\_ERROR, 37
  - SESSION\_CLOSE, 37
  - STATUS\_OK, 36

- StatusKind, 36
- string, 42
- thread\_stacksize, 43
- TimeStamp, 42
- UNKNOWN\_SERVICE\_ERROR, 37
- VOMSDecode, 42
- WSAFault, 37
- WSAFaultAssign, 42
- WSAFaultExtract, 42
- WSAFaultInvalidAddressingHeader, 37
- WSAFaultUnknown, 37
- Arc::Adler32Sum, 47
- Arc::ApplicationEnvironment, 50
- Arc::ApplicationType, 50
- Arc::ARCJSDDLParser, 50
- Arc::ArcLocation, 51
  - GetPlugins, 51
  - Init, 51
- Arc::ARCPolicyHandlerConfig, 51
- Arc::AttributeIterator, 53
  - AttributeIterator, 53
  - current\_, 55
  - end\_, 55
  - hasMore, 54
  - key, 54
  - MessageAttributes, 55
  - operator\*, 54
  - operator++, 54
  - operator->, 55
- Arc::AutoPointer, 59
- Arc::Base64, 60
- Arc::BaseConfig, 60
  - AddCADir, 60
  - AddCAFile, 60
  - AddCertificate, 60
  - AddOverlay, 61
  - AddPluginsPath, 61
  - AddPrivateKey, 61
  - AddProxy, 61
  - GetOverlay, 61
  - MakeConfig, 61
- Arc::Broker, 62
  - GetBestTarget, 63
  - PossibleTargets, 64
  - PreFilterTargets, 63
  - SortTargets, 63
- Arc::BrokerLoader, 64
  - ~BrokerLoader, 64
  - BrokerLoader, 64
  - GetBrokers, 65
  - load, 65
- Arc::BrokerPluginArgument, 65
- Arc::ByteArray, 65
- Arc::CacheParameters, 66
- Arc::ChainContext, 66
  - operator PluginsFactory \*, 66
- Arc::Checksum, 67
- Arc::ChecksumAny, 67
- Arc::CIStringValue, 67
  - CIStringValue, 68
  - equal, 68
  - operator bool, 68
- Arc::ClassLoader, 69
- Arc::ClassLoaderPluginArgument, 69
- Arc::ClientHTTP, 69
- Arc::ClientHTTPwithSAML2SSO, 70
  - ClientHTTPwithSAML2SSO, 70
  - process, 70
- Arc::ClientInterface, 71
- Arc::ClientSOAP, 71
  - AddSecHandler, 72
  - ClientSOAP, 72
  - GetEntry, 72
  - Load, 72
  - process, 72
- Arc::ClientSOAPwithSAML2SSO, 73
  - ClientSOAPwithSAML2SSO, 73
  - process, 73
- Arc::ClientTCP, 73
- Arc::ClientX509Delegation, 74
  - acquireDelegation, 74
  - ClientX509Delegation, 74
  - createDelegation, 75
- Arc::Config, 76
  - Config, 77
  - getFileName, 78
  - parse, 78
  - print, 78
  - save, 78
  - setFileName, 78
- Arc::ConfusaCertHandler, 78
  - ConfusaCertHandler, 79
  - createCertRequest, 79
  - getCertRequestB64, 79
- Arc::ConfusaParserUtils, 79
  - destroy\_doc, 79
  - evaluate\_path, 79
  - extract\_body\_information, 80
  - get\_doc, 80
  - handle\_redirect\_step, 80
  - urlencode, 80
  - urlencode\_params, 80
- Arc::CountedPointer, 80
- Arc::Counter, 81
  - ~Counter, 83
  - cancel, 84
  - changeExcess, 84
  - changeLimit, 84

- Counter, 83
- extend, 84
- getCounterTicket, 85
- getCurrentTime, 85
- getExcess, 85
- getExpirationReminder, 85
- getExpiryTime, 86
- getLimit, 86
- getValue, 86
- IDType, 83
- reserve, 87
- setExcess, 87
- setLimit, 87
- Arc::CounterTicket, 88
  - cancel, 89
  - CounterTicket, 89
  - extend, 89
  - isValid, 89
- Arc::CRC32Sum, 89
- Arc::Credential, 90
  - AddCertExtObj, 92
  - AddExtension, 92
  - Credential, 91, 92
  - GenerateEECRequest, 93
  - GenerateRequest, 93
  - GetCert, 93
  - GetCertNumofChain, 93
  - GetCertReq, 94
  - GetDN, 94
  - GetEndTime, 94
  - getFormat, 94
  - GetIdentityName, 94
  - GetLifeTime, 94
  - GetPrivKey, 94
  - GetProxyPolicy, 94
  - GetPubKey, 94
  - GetStartTime, 94
  - GetType, 94
  - GetVerification, 95
  - InitProxyCertInfo, 95
  - InquireRequest, 95
  - LogError, 95
  - OutputCertificate, 95
  - OutputCertificateChain, 95
  - OutputPrivatekey, 96
  - OutputPublickey, 96
  - SetLifeTime, 96
  - SetProxyPolicy, 96
  - SetStartTime, 96
  - SignEECRequest, 96
  - SignRequest, 96, 97
  - STACK\_OF, 97
- Arc::CredentialError, 97
  - CredentialError, 98
- Arc::CredentialStore, 98
- Arc::Database, 98
  - ~Database, 99
  - close, 99
  - connect, 99
  - Database, 99
  - enable\_ssl, 99
  - isconnected, 100
  - shutdown, 100
- Arc::DataBuffer, 100
  - add, 102
  - buffer\_size, 102
  - checksum\_object, 102
  - checksum\_valid, 102
  - DataBuffer, 101
  - eof\_read, 102
  - eof\_write, 103
  - error, 103
  - error\_read, 103
  - error\_write, 103
  - for\_read, 103
  - for\_write, 103, 104
  - is\_notwritten, 104
  - is\_read, 104
  - is\_written, 105
  - set, 105
  - wait\_any, 105
- Arc::DataCallback, 105
- Arc::DataHandle, 106
- Arc::DataMover, 106
  - checks, 107
  - force\_to\_meta, 107
  - secure, 107
  - set\_default\_max\_inactivity\_time, 107
  - set\_default\_min\_average\_speed, 107
  - set\_default\_min\_speed, 107
  - Transfer, 107, 108
  - verbose, 108
- Arc::DataPoint, 108
  - ACCESS\_LATENCY\_LARGE, 111
  - ACCESS\_LATENCY\_SMALL, 111
  - ACCESS\_LATENCY\_ZERO, 111
  - AddChecksumObject, 112
  - AddLocation, 112
  - Check, 112
  - CompareLocationMetadata, 112
  - CompareMeta, 112
  - CurrentLocationMetadata, 112
  - DataPoint, 111
  - DataPointAccessLatency, 111
  - DataPointInfoType, 111
  - GetFailureReason, 113
  - INFO\_TYPE\_ACCESS, 111
  - INFO\_TYPE\_ALL, 111

- INFO\_TYPE\_CONTENT, 111
- INFO\_TYPE\_NAME, 111
- INFO\_TYPE\_REST, 111
- INFO\_TYPE\_STRUCT, 111
- INFO\_TYPE\_TIMES, 111
- INFO\_TYPE\_TYPE, 111
- List, 113
- NextLocation, 113
- Passive, 113
- PostRegister, 113
- PreRegister, 114
- PreUnregister, 114
- ProvidesMeta, 114
- Range, 114
- ReadOutOfOrder, 114
- Registered, 115
- Resolve, 115
- SetAdditionalChecks, 115
- SetMeta, 115
- SetSecure, 116
- SortLocations, 116
- StartReading, 116
- StartWriting, 116
- Stat, 117
- StopReading, 117
- StopWriting, 117
- Unregister, 117
- valid\_url\_options, 118
- WriteOutOfOrder, 117
- Arc::DataPointDirect, 118
  - AddChecksumObject, 119
  - AddLocation, 119
  - CompareLocationMetadata, 119
  - CurrentLocationMetadata, 120
  - NextLocation, 120
  - Passive, 120
  - PostRegister, 120
  - PreRegister, 120
  - PreUnregister, 121
  - ProvidesMeta, 121
  - Range, 121
  - ReadOutOfOrder, 121
  - Registered, 121
  - Resolve, 122
  - SetAdditionalChecks, 122
  - SetSecure, 122
  - SortLocations, 122
  - Unregister, 122
  - WriteOutOfOrder, 123
- Arc::DataPointIndex, 123
  - AddChecksumObject, 124
  - AddLocation, 124
  - Check, 125
  - CompareLocationMetadata, 125
  - CurrentLocationMetadata, 125
  - NextLocation, 125
  - Passive, 125
  - ProvidesMeta, 125
  - Range, 126
  - ReadOutOfOrder, 126
  - Registered, 126
  - SetAdditionalChecks, 126
  - SetMeta, 126
  - SetSecure, 127
  - SortLocations, 127
  - StartReading, 127
  - StartWriting, 127
  - StopReading, 128
  - StopWriting, 128
  - WriteOutOfOrder, 128
- Arc::DataPointLoader, 128
- Arc::DataPointPluginArgument, 129
- Arc::DataSourceType, 129
- Arc::DataSpeed, 129
  - DataSpeed, 130
  - hold, 130
  - set\_base, 130
  - set\_max\_data, 130
  - set\_max\_inactivity\_time, 131
  - set\_min\_average\_speed, 131
  - set\_min\_speed, 131
  - set\_progress\_indicator, 131
  - transfer, 131
  - verbose, 131, 132
- Arc::DataStagingType, 132
- Arc::DataStatus, 132
  - CacheError, 133
  - CheckError, 133
  - CredentialsExpiredError, 133
  - DataStatusType, 133
  - DeleteError, 133
  - InconsistentMetadataError, 133
  - IsReadingError, 133
  - IsWritingError, 133
  - ListError, 133
  - LocationAlreadyExistsError, 133
  - NoLocationError, 133
  - NotInitializedError, 133
  - NotSupportedForDirectDataPointsError, 133
  - PostRegisterError, 133
  - PreRegisterError, 133
  - ReadAcquireError, 133
  - ReadError, 133
  - ReadResolveError, 133
  - ReadStartError, 133
  - ReadStopError, 133
  - StageError, 133
  - StatError, 133

- Success, 133
- SuccessCached, 133
- SystemError, 133
- TransferError, 133
- UnimplementedError, 133
- UnknownError, 133
- UnregisterError, 133
- WriteAcquireError, 133
- WriteError, 133
- WriteResolveError, 133
- WriteStartError, 133
- WriteStopError, 133
- Arc::DataTargetType, 134
- Arc::DataType, 134
- Arc::DBranch, 136
- Arc::DelegationConsumer, 136
  - Acquire, 137
  - Backup, 137
  - DelegationConsumer, 137
  - Generate, 137
  - ID, 138
  - LogError, 138
  - Request, 138
  - Restore, 138
- Arc::DelegationConsumerSOAP, 138
  - DelegateCredentialsInit, 139
  - DelegatedToken, 139
  - DelegationConsumerSOAP, 139
  - UpdateCredentials, 139
- Arc::DelegationContainerSOAP, 139
  - context\_lock\_, 140
  - DelegateCredentialsInit, 140
  - DelegatedToken, 140
  - max\_duration\_, 140
  - max\_size\_, 140
  - max\_usage\_, 141
  - restricted\_, 141
  - UpdateCredentials, 140
- Arc::DelegationProvider, 141
  - Delegate, 142
  - DelegationProvider, 141
- Arc::DelegationProviderSOAP, 142
  - DelegateCredentialsInit, 143
  - DelegatedToken, 143
  - DelegationProviderSOAP, 143
  - ID, 143
  - UpdateCredentials, 143, 144
- Arc::DirectoryType, 145
- Arc::DiskSpaceRequirementType, 145
- Arc::DItem, 146
- Arc::DItemString, 146
- Arc::DNListHandlerConfig, 146
- Arc::ExecutableType, 154
- Arc::ExecutionTarget, 154
  - ApplicationEnvironments, 156
  - ComputingShareName, 156
  - ExecutionTarget, 155
  - FreeSlotsWithDuration, 157
  - GetSubmitter, 156
  - MaxDiskSpace, 157
  - MaxMainMemory, 157
  - MaxVirtualMemory, 157
  - OperatingSystem, 157
  - operator=, 156
  - Print, 156
  - Update, 156
- Arc::ExpirationReminder, 158
  - getExpiryTime, 158
  - getReservationID, 158
  - operator<, 158
- Arc::FileCache, 159
  - AddDN, 161
  - CheckCreated, 161
  - CheckDN, 161
  - CheckValid, 161
  - Copy, 161
  - File, 162
  - FileCache, 160, 161
  - GetCreated, 162
  - GetValid, 162
  - Link, 162
  - operator bool, 162
  - operator==, 162
  - Release, 162
  - SetValid, 162
  - Start, 163
  - Stop, 163
  - StopAndDelete, 163
- Arc::FileInfo, 164
- Arc::FileLock, 164
- Arc::FileType, 165
- Arc::FinderLoader, 165
- Arc::GlobusResult, 168
- Arc::GSSCredential, 168
- Arc::HakaClient, 168
  - processConsent, 169
  - processIdP2Confusa, 169
  - processIdPLogin, 169
- Arc::HTTPClientInfo, 169
- Arc::InfoCache, 169
  - InfoCache, 170
- Arc::InfoCacheInterface, 170
  - Get, 170
- Arc::InfoFilter, 170
  - Filter, 171
  - InfoFilter, 171
- Arc::InfoRegister, 171
- Arc::InfoRegisterContainer, 172

- addRegistrar, 172
- addService, 172
- removeService, 172
- Arc::InfoRegisters, 173
  - InfoRegisters, 173
- Arc::InfoRegistrar, 173
  - addService, 174
  - registration, 174
- Arc::InformationContainer, 174
  - Acquire, 175
  - Assign, 175
  - doc\_, 175
  - Get, 175
  - InformationContainer, 175
- Arc::InformationInterface, 175
  - Get, 176
  - InformationInterface, 176
  - lock\_, 176
- Arc::InformationRequest, 176
  - InformationRequest, 177
  - SOAP, 177
- Arc::InformationResponse, 178
  - InformationResponse, 178
  - Result, 178
- Arc::IniConfig, 178
- Arc::initializeCredentialsType, 179
- Arc::IntraProcessCounter, 179
  - ~IntraProcessCounter, 180
  - cancel, 181
  - changeExcess, 181
  - changeLimit, 181
  - extend, 181
  - getExcess, 182
  - getLimit, 182
  - getValue, 182
  - IntraProcessCounter, 180
  - reserve, 183
  - setExcess, 183
  - setLimit, 183
- Arc::ISIS\_description, 184
- Arc::IString, 184
- Arc::JDLParser, 184
- Arc::Job, 184
  - Job, 185
  - Print, 185
- Arc::JobController, 185
  - FillJobStore, 186
  - Migrate, 186
  - PrintJobStatus, 186
- Arc::JobControllerLoader, 187
  - ~JobControllerLoader, 187
  - GetJobControllers, 188
  - JobControllerLoader, 187
  - load, 188
- Arc::JobControllerPluginArgument, 188
- Arc::JobDescription, 188
- Arc::JobDescriptionParser, 189
- Arc::JobIdentificationType, 189
- Arc::JobMetaType, 189
- Arc::JobState, 189
- Arc::JobSupervisor, 190
  - GetJobControllers, 190
  - JobSupervisor, 190
- Arc::LoadableModuleDescription, 190
- Arc::Loader, 191
  - ~Loader, 191
  - factory\_, 191
  - Loader, 191
- Arc::LogDestination, 192
  - LogDestination, 192
- Arc::LogFile, 192
  - log, 194
  - LogFile, 193
  - setBackups, 194
  - setMaxSize, 194
  - setReopen, 194
- Arc::Logger, 195
  - ~Logger, 196
  - addDestination, 196
  - getRootLogger, 196
  - getThreshold, 196
  - Logger, 195
  - msg, 196, 197
  - setThreshold, 197
- Arc::LoggerFormat, 197
- Arc::LogMessage, 197
  - getLevel, 198
  - Logger, 199
  - LogMessage, 198
  - operator<<, 199
  - setIdentifier, 199
- Arc::LogStream, 199
  - log, 200
  - LogStream, 200
- Arc::MCC, 202
  - AddSecHandler, 203
  - logger, 204
  - MCC, 203
  - Next, 203
  - next\_, 204
  - process, 203
  - ProcessSecHandlers, 203
  - sechandlers\_, 204
  - Unlink, 203
- Arc::MCC\_Status, 204
  - getExplanation, 205
  - getKind, 205
  - getOrigin, 205

- isOk, 205
- MCC\_Status, 205
- operator bool, 206
- operator std::string, 206
- Arc::MCCConfig, 206
  - MakeConfig, 207
- Arc::MCCInterface, 207
  - process, 207
- Arc::MCCLoader, 208
  - ~MCCLoader, 209
  - MCCLoader, 209
  - operator[], 209
- Arc::MCCPluginArgument, 209
- Arc::MD5Sum, 209
- Arc::MemoryAllocationException, 210
- Arc::Message, 210
  - ~Message, 211
  - Attributes, 211
  - Auth, 211
  - AuthContext, 211
  - Context, 212
  - Message, 211
  - operator=, 212
  - Payload, 212
- Arc::MessageAttributes, 212
  - add, 213
  - attributes\_, 215
  - count, 213
  - get, 214
  - getAll, 214
  - MessageAttributes, 213
  - remove, 214
  - removeAll, 214
  - set, 215
- Arc::MessageAuth, 215
  - Export, 216
  - Filter, 216
- Arc::MessageAuthContext, 216
- Arc::MessageContext, 217
  - Add, 217
- Arc::MessageContextElement, 217
- Arc::MessagePayload, 218
- Arc::ModuleDesc, 218
- Arc::ModuleManager, 218
  - find, 219
  - findLocation, 219
  - load, 219
  - makePersistent, 219, 220
  - ModuleManager, 219
  - reload, 220
  - setCfg, 220
  - unload, 220
- Arc::MultiSecAttr, 220
  - Export, 221
  - operator bool, 221
- Arc::MySQLDatabase, 221
  - close, 222
  - connect, 222
  - enable\_ssl, 222
  - isconnected, 222
  - shutdown, 222
- Arc::MySQLQuery, 223
  - execute, 223
  - get\_array, 223
  - get\_num\_columns, 223
  - get\_num\_rows, 224
  - get\_row, 224
  - get\_row\_field, 224
- Arc::NotificationType, 225
- Arc::NS, 225
- Arc::OAuthConsumer, 225
  - approveCSR, 226
  - OAuthConsumer, 226
  - parseDN, 226
  - processLogin, 226
  - pushCSR, 226
  - storeCert, 226
- Arc::OpenIdpClient, 226
  - processConsent, 227
  - processIdP2Confusa, 227
  - processIdPLogin, 227
- Arc::OptionParser, 227
- Arc::PathIterator, 228
  - operator bool, 229
  - operator\*, 229
  - operator++, 229
  - operator--, 229
  - PathIterator, 228
  - Rest, 229
- Arc::PayloadRaw, 229
  - ~PayloadRaw, 230
  - Buffer, 230
  - BufferPos, 230
  - BufferSize, 230
  - Content, 230
  - Insert, 231
  - operator[], 231
  - PayloadRaw, 230
  - Size, 231
  - Truncate, 231
- Arc::PayloadRawBuf, 231
  - allocated, 232
  - length, 232
  - size, 232
- Arc::PayloadRawInterface, 232
  - Buffer, 233
  - BufferPos, 233
  - BufferSize, 233

- Content, 233
- Insert, 233
- operator[], 233
- Size, 234
- Truncate, 234
- Arc::PayloadSOAP, 234
  - PayloadSOAP, 235
- Arc::PayloadStream, 235
  - ~PayloadStream, 236
  - Get, 236
  - handle\_, 238
  - Limit, 236
  - operator bool, 237
  - PayloadStream, 236
  - Pos, 237
  - Put, 237
  - seekable\_, 238
  - Size, 237
  - Timeout, 238
- Arc::PayloadStreamInterface, 238
  - Get, 239
  - Limit, 239
  - operator bool, 239
  - Pos, 240
  - Put, 240
  - Size, 240
  - Timeout, 240
- Arc::PayloadWSRF, 241
  - PayloadWSRF, 241
- Arc::Period, 243
  - GetPeriod, 244
  - istr, 244
  - operator std::string, 244
  - operator<, 244
  - operator<=, 244
  - operator>, 244
  - operator>=, 244
  - operator=, 244
  - operator==, 244
  - Period, 243
  - SetPeriod, 245
- Arc::Plexer, 247
  - ~Plexer, 248
  - logger, 248
  - Next, 248
  - Plexer, 248
  - process, 248
- Arc::PlexerEntry, 249
- Arc::Plugin, 249
- Arc::PluginArgument, 250
  - get\_factory, 251
  - get\_module, 251
- Arc::PluginDesc, 252
- Arc::PluginDescriptor, 252
  - Arc::PluginsFactory, 252
    - load, 253
    - PluginsFactory, 253
    - report, 253
    - scan, 253
    - TryLoad, 253
  - Arc::Printf, 258
  - Arc::PrintfBase, 258
  - Arc::Profile, 258
  - Arc::Query, 259
    - ~Query, 260
    - execute, 260
    - get\_array, 260
    - get\_num\_columns, 260
    - get\_num\_rows, 260
    - get\_row, 260, 261
    - get\_row\_field, 261
    - Query, 259
  - Arc::Range, 261
  - Arc::Register\_Info\_Type, 261
  - Arc::RegisteredService, 262
    - RegisteredService, 262
  - Arc::RegularExpression, 262
    - match, 263
  - Arc::ResourceSlotType, 266
  - Arc::ResourcesType, 266
  - Arc::ResourceTargetType, 267
  - Arc::RSL, 267
  - Arc::RSLBoolean, 268
  - Arc::RSLConcat, 268
  - Arc::RSLCondition, 268
  - Arc::RSLList, 269
  - Arc::RSLLiteral, 269
  - Arc::RSLParser, 269
  - Arc::RSLSequence, 270
  - Arc::RSLValue, 270
  - Arc::RSLVariable, 270
  - Arc::Run, 270
    - ~Run, 271
    - Abandon, 272
    - AfterFork, 272
    - AssignStderr, 272
    - AssignStdin, 272
    - AssignStdout, 272
    - AssignWorkingDirectory, 272
    - CloseStderr, 272
    - CloseStdin, 272
    - CloseStdout, 272
    - KeepStderr, 272
    - KeepStdin, 273
    - KeepStdout, 273
    - Kill, 273
    - operator bool, 273
    - ReadStderr, 273

- ReadStdout, 273
- Result, 273
- Run, 271
- Running, 273
- Start, 273
- Wait, 274
- WriteStdin, 274
- Arc::SAML2LoginClient, 274
  - findSimpleSAMLInstallation, 275
  - processLogin, 275
  - SAML2LoginClient, 274
- Arc::SAML2SSOHTTPClient, 275
  - approveCSR, 276
  - parseDN, 276
  - processConsent, 276
  - processIdP2Confusa, 276
  - processIdPLogin, 276
  - processLogin, 276
  - pushCSR, 276
  - storeCert, 276
- Arc::SAMLToken, 277
  - ~SAMLToken, 279
  - Authenticate, 279
  - operator bool, 279
  - SAMLToken, 278
  - SAMLVersion, 278
- Arc::ScalableTime, 279
- Arc::ScalableTime< int >, 280
- Arc::SecAttr, 280
  - Export, 281
  - Import, 281
  - operator bool, 281
  - operator==, 281
- Arc::SecAttrFormat, 281
- Arc::SecAttrValue, 282
  - operator bool, 282
  - operator==, 282
- Arc::SecHandlerConfig, 283
- Arc::Service, 285
  - AddSecHandler, 286
  - getID, 286
  - logger, 286
  - ProcessSecHandlers, 286
  - RegistrationCollector, 286
  - sechandlers\_, 286
  - Service, 286
- Arc::ServicePluginArgument, 287
- Arc::SimpleCondition, 287
  - broadcast, 288
  - lock, 288
  - reset, 288
  - signal, 288
  - signal\_nonblock, 288
  - unlock, 288
  - wait, 288
  - wait\_nonblock, 288
- Arc::SimpleCounter, 288
  - wait, 289
- Arc::SOAPMessage, 289
  - ~SOAPMessage, 289
  - Attributes, 290
  - Payload, 290
  - SOAPMessage, 289
- Arc::Software, 290
  - ComparisonOperator, 291
  - ComparisonOperatorEnum, 292
  - convert, 293
  - empty, 293
  - EQUAL, 292
  - getFamily, 293
  - getName, 293
  - getVersion, 294
  - GREATERTHAN, 292
  - GREATERTHANOREQUAL, 292
  - LESSTHAN, 292
  - LESSTHANOREQUAL, 292
  - NOTEQUAL, 292
  - operator std::string, 294
  - operator<, 295
  - operator<<, 297
  - operator<=, 295
  - operator>, 296
  - operator>=, 296
  - operator(), 294
  - operator==, 295
  - Software, 292
  - toString, 296
  - VERSIONTOKENS, 297
- Arc::SoftwareRequirement, 297
  - add, 299, 300
  - clear, 300
  - empty, 300
  - getComparisonOperatorList, 300
  - getSoftwareList, 300
  - isRequiringAll, 301
  - isResolved, 301
  - isSatisfied, 301, 302
  - operator=, 303
  - selectSoftware, 303, 304
  - setRequirement, 304
  - SoftwareRequirement, 298, 299
- Arc::Submitter, 308
  - Migrate, 308
  - Submit, 309
- Arc::SubmitterLoader, 309
  - ~SubmitterLoader, 309
  - GetSubmitters, 310
  - load, 310

- SubmitterLoader, 309
- Arc::SubmitterPluginArgument, 310
- Arc::TargetGenerator, 310
  - AddIndexServer, 311
  - AddJob, 311
  - AddService, 312
  - AddTarget, 312
  - FoundJobs, 312
  - FoundTargets, 312
  - GetTargets, 312
  - ModifyFoundTargets, 313
  - PrintTargetInfo, 313
  - ServiceCounter, 313
  - TargetGenerator, 311
- Arc::TargetRetriever, 313
  - GetTargets, 314
  - TargetRetriever, 314
- Arc::TargetRetrieverLoader, 315
  - ~TargetRetrieverLoader, 315
  - GetTargetRetrievers, 315
  - load, 315
  - TargetRetrieverLoader, 315
- Arc::TargetRetrieverPluginArgument, 316
- Arc::ThreadInitializer, 318
- Arc::ThreadRegistry, 318
  - WaitForExit, 318
  - WaitOrCancel, 318
- Arc::Time, 318
  - GetFormat, 320
  - GetTime, 320
  - operator std::string, 320
  - operator<, 320
  - operator<=, 320
  - operator>, 321
  - operator>=, 321
  - operator+, 320
  - operator-, 320
  - operator=, 320, 321
  - operator==, 321
  - SetFormat, 321
  - SetTime, 321
  - str, 321
  - Time, 319
- Arc::URL, 323
  - ~URL, 325
  - AddLDAPAttribute, 325
  - AddMetaDataOption, 325
  - AddOption, 325
  - BaseDN2Path, 325
  - ChangeHost, 325
  - ChangeLDAPFilter, 325
  - ChangeLDAPScope, 325
  - ChangePath, 325
  - ChangePort, 326
  - ChangeProtocol, 326
  - CommonLocOption, 326
  - CommonLocOptions, 326
  - commonlocoptions, 329
  - ConnectionURL, 326
  - FullPath, 326
  - fullstr, 326
  - Host, 326
  - host, 329
  - HTTPOption, 326
  - HTTPOptions, 327
  - httpoptions, 329
  - ip6addr, 329
  - IsSecureProtocol, 327
  - LDAPAttributes, 327
  - ldapattributes, 329
  - LDAPFilter, 327
  - ldapfilter, 330
  - LDAPScope, 327
  - ldapscope, 330
  - Locations, 327
  - locations, 330
  - MetaDataOption, 327
  - MetaDataOptions, 327
  - metadataoptions, 330
  - operator bool, 327
  - operator<, 328
  - operator<<, 329
  - operator==, 328
  - Option, 328
  - Options, 328
  - OptionString, 328
  - ParseOptions, 328
  - Passwd, 328
  - passwd, 330
  - Path, 328
  - path, 330
  - Path2BaseDN, 328
  - plainstr, 328
  - Port, 329
  - port, 330
  - Protocol, 329
  - protocol, 330
  - Scope, 324
  - str, 329
  - URL, 325
  - urloptions, 330
  - Username, 329
  - username, 330
  - valid, 330
- Arc::URLLocation, 331
  - ~URLLocation, 332
  - fullstr, 332
  - Name, 332

- name, 332
- str, 332
- URLLocation, 331, 332
- Arc::URLMap, 333
- Arc::User, 333
- Arc::UserConfig, 333
  - AddBartender, 338
  - AddServices, 338
  - ApplyToConfig, 339
  - ARCUSERDIRECTORY, 359
  - Bartender, 339, 340
  - Broker, 340, 341
  - CACertificatePath, 341, 342
  - CACertificatesDirectory, 342
  - CertificateLifeTime, 343
  - CertificatePath, 343, 344
  - ClearRejectedServices, 344
  - ClearSelectedServices, 345
  - CredentialsFound, 345
  - DEFAULT\_BROKER, 359
  - DEFAULT\_TIMEOUT, 359
  - DEFAULTCONFIG, 360
  - EXAMPLECONFIG, 360
  - GetRejectedServices, 345
  - GetSelectedServices, 346
  - IdPName, 346, 347
  - InitializeCredentials, 347
  - JobListFile, 348
  - KeyPassword, 349
  - KeyPath, 349, 350
  - KeySize, 350
  - LoadConfigurationFile, 351
  - operator bool, 352
  - OverlayFile, 353
  - Password, 353, 354
  - ProxyPath, 354
  - SaveToFile, 354
  - SLCS, 355
  - StoreDirectory, 355, 356
  - SYSCONFIG, 360
  - SYSCONFIGARCLOC, 360
  - Timeout, 356
  - UserConfig, 336, 337
  - UserName, 357
  - UtilsDirPath, 357
  - Verbosity, 358
  - VOMSServerPath, 358, 359
- Arc::UsernameToken, 360
  - Authenticate, 362
  - operator bool, 362
  - PasswordType, 361
  - Username, 362
  - UsernameToken, 361
- Arc::UserSwitch, 362
- Arc::VOMSTrustList, 363
  - AddChain, 364
  - AddRegex, 364
  - VOMSTrustList, 363, 364
- Arc::WSAEndpointReference, 364
  - ~WSAEndpointReference, 365
  - Address, 365
  - MetaData, 366
  - operator XMLNode, 366
  - operator=, 366
  - ReferenceParameters, 366
  - WSAEndpointReference, 365
- Arc::WSAHeader, 366
  - Action, 367
  - Check, 367
  - FaultTo, 367
  - From, 368
  - header\_allocated\_, 369
  - MessageID, 368
  - NewReferenceParameter, 368
  - operator XMLNode, 368
  - ReferenceParameter, 368
  - RelatesTo, 368
  - RelationshipType, 368
  - ReplyTo, 369
  - To, 369
  - WSAHeader, 367
- Arc::WSRF, 369
  - allocated\_, 371
  - operator bool, 371
  - set\_namespaces, 371
  - SOAP, 371
  - valid\_, 371
  - WSRF, 370
- Arc::WSRFBaseFault, 371
  - set\_namespaces, 372
  - WSRFBaseFault, 372
- Arc::WSRFResourceUnavailableFault, 372
- Arc::WSRFResourceUnknownFault, 373
- Arc::WSRP, 373
  - set\_namespaces, 375
  - WSRP, 374
- Arc::WSRPDeleteResourceProperties, 375
- Arc::WSRPDeleteResourcePropertiesRequest, 375
- Arc::WSRPDeleteResourcePropertiesRequestFailedFault, 376
- Arc::WSRPDeleteResourcePropertiesResponse, 376
- Arc::WSRPFault, 376
  - WSRPFault, 377
- Arc::WSRPGetMultipleResourcePropertiesRequest, 377
- Arc::WSRPGetMultipleResourcePropertiesResponse, 378

- Arc::WSRPGetResourcePropertyDocumentRequest, 378
- Arc::WSRPGetResourcePropertyDocumentResponse, 378
- Arc::WSRPGetResourcePropertyRequest, 379
- Arc::WSRPGetResourcePropertyResponse, 379
- Arc::WSRPInsertResourceProperties, 380
- Arc::WSRPInsertResourcePropertiesRequest, 380
- Arc::WSRPInsertResourcePropertiesRequestFailedFault, 380
- Arc::WSRPInsertResourcePropertiesResponse, 381
- Arc::WSRPInvalidModificationFault, 381
- Arc::WSRPInvalidResourcePropertyQNameFault, 382
- Arc::WSRPModifyResourceProperties, 382
- Arc::WSRPPutResourcePropertyDocumentRequest, 383
- Arc::WSRPPutResourcePropertyDocumentResponse, 383
- Arc::WSRPQueryResourcePropertiesRequest, 383
- Arc::WSRPQueryResourcePropertiesResponse, 384
- Arc::WSRPResourcePropertyChangeFailure, 384
  - WSRPResourcePropertyChangeFailure, 385
- Arc::WSRPSetResourcePropertiesRequest, 385
- Arc::WSRPSetResourcePropertiesResponse, 385
- Arc::WSRPSetResourcePropertyRequestFailedFault, 386
- Arc::WSRPUnableToModifyResourcePropertyFault, 386
- Arc::WSRPUnableToPutResourcePropertyDocumentFault, 387
- Arc::WSRPUpdateResourceProperties, 387
- Arc::WSRPUpdateResourcePropertiesRequest, 387
- Arc::WSRPUpdateResourcePropertiesRequestFailedFault, 388
- Arc::WSRPUpdateResourcePropertiesResponse, 388
- Arc::X509Token, 390
  - ~X509Token, 391
  - Authenticate, 391
  - operator bool, 391
  - X509Token, 390
  - X509TokenType, 390
- Arc::XmlContainer, 391
- Arc::XmlDatabase, 392
- Arc::XMLNode, 392
  - ~XMLNode, 395
  - Attribute, 395
  - AttributesSize, 395
  - Child, 395
  - Destroy, 395
  - Exchange, 396
  - FullName, 396
  - Get, 396
  - GetDoc, 396
  - GetRoot, 396
  - GetXML, 396
  - is\_owner\_, 403
  - is\_temporary\_, 403
  - MatchXMLName, 402
  - MatchXMLNamespace, 402, 403
  - Move, 396
  - Name, 397
  - Namespace, 397
  - NamespacePrefix, 397
  - Namespaces, 397
  - New, 397
  - NewAttribute, 397, 398
  - NewChild, 398
  - operator bool, 398
  - operator std::string, 398
  - operator++, 399
  - operator--, 399
  - operator=, 399
  - operator==, 400
  - operator[], 400
  - Parent, 400
  - Path, 401
  - Prefix, 401
  - ReadFromFile, 401
  - ReadFromStream, 401
  - Replace, 401
  - Same, 401
  - SaveToFile, 401
  - SaveToStream, 401
  - Set, 401
  - Size, 401
  - Swap, 402
  - Validate, 402
  - XMLNode, 394, 395
  - XPathLookup, 402
- Arc::XMLNodeContainer, 403
  - Add, 404
  - AddNew, 404
  - Nodes, 404
  - operator=, 404
  - operator[], 404
  - Size, 404
  - XMLNodeContainer, 404
- Arc::XMLSecNode, 405
  - AddSignatureTemplate, 405
  - DecryptNode, 406
  - EncryptNode, 406
  - SignNode, 406
  - VerifyNode, 406
  - XMLSecNode, 405
- Arc::XRSLParser, 407

- ArcCredential, 43
  - CERT\_TYPE\_CA, 44
  - CERT\_TYPE\_EEC, 44
  - CERT\_TYPE\_GSI\_2\_LIMITED\_PROXY, 44
  - CERT\_TYPE\_GSI\_2\_PROXY, 44
  - CERT\_TYPE\_GSI\_3\_IMPERSONATION\_PROXY, 44
  - CERT\_TYPE\_GSI\_3\_INDEPENDENT\_PROXY, 44
  - CERT\_TYPE\_GSI\_3\_LIMITED\_PROXY, 44
  - CERT\_TYPE\_GSI\_3\_RESTRICTED\_PROXY, 44
  - CERT\_TYPE\_RFC\_ANYLANGUAGE\_PROXY, 44
  - CERT\_TYPE\_RFC\_IMPERSONATION\_PROXY, 44
  - CERT\_TYPE\_RFC\_INDEPENDENT\_PROXY, 44
  - CERT\_TYPE\_RFC\_LIMITED\_PROXY, 44
  - CERT\_TYPE\_RFC\_RESTRICTED\_PROXY, 44
  - certType, 44
- ArcCredential::ACACI, 45
- ArcCredential::ACATTHOLDER, 45
- ArcCredential::ACATTR, 45
- ArcCredential::ACATTRIBUTE, 45
- ArcCredential::ACC, 45
- ArcCredential::ACCERTS, 46
- ArcCredential::ACDIGEST, 46
- ArcCredential::ACFORM, 46
- ArcCredential::ACFULLATTRIBUTES, 46
- ArcCredential::ACHOLDER, 46
- ArcCredential::ACIETFATTR, 46
- ArcCredential::ACINFO, 46
- ArcCredential::ACIS, 47
- ArcCredential::ACSEQ, 47
- ArcCredential::ACTARGET, 47
- ArcCredential::ACTARGETS, 47
- ArcCredential::ACVAL, 47
- ArcCredential::cert\_verify\_context, 66
- ArcCredential::PROXYCERTINFO\_st, 259
- ArcCredential::PROXYPOLICY\_st, 259
- ArcSec::AlgFactory, 48
  - createAlg, 48
- ArcSec::AnyURIAttribute, 49
  - encode, 49
  - equal, 49
  - getId, 49
  - getType, 49
- ArcSec::ArcPeriod, 51
- ArcSec::Attr, 52
- ArcSec::AttributeFactory, 52
- ArcSec::AttributeProxy, 55
  - getAttribute, 56
- ArcSec::AttributeValue, 56
  - encode, 57
  - equal, 58
  - getId, 58
  - getType, 58
- ArcSec::Attrs, 58
- ArcSec::AuthzRequest, 59
- ArcSec::AuthzRequestSection, 59
- ArcSec::BooleanAttribute, 61
  - encode, 62
  - equal, 62
  - getId, 62
  - getType, 62
- ArcSec::CombiningAlg, 75
  - combine, 76
  - getalgId, 76
- ArcSec::DateAttribute, 134
  - encode, 134
  - equal, 134
  - getId, 135
  - getType, 135
- ArcSec::DateTimeAttribute, 135
  - encode, 136
  - equal, 136
  - getId, 136
  - getType, 136
- ArcSec::DenyOverridesCombiningAlg, 144
  - combine, 145
  - getalgId, 145
- ArcSec::DurationAttribute, 147
  - encode, 147
  - equal, 147
  - getId, 147
  - getType, 147
- ArcSec::EqualFunction, 148
  - evaluate, 148
  - getFunctionName, 148
- ArcSec::EvalResult, 149
- ArcSec::EvaluationCtx, 149
  - EvaluationCtx, 149
- ArcSec::Evaluator, 150
  - addPolicy, 150
  - evaluate, 151
  - getAlgFactory, 151
  - getAttrFactory, 152
  - getFnFactory, 152
  - getName, 152
  - setCombiningAlg, 152
- ArcSec::EvaluatorContext, 152
  - operator AlgFactory \*, 153
  - operator AttributeFactory \*, 153
  - operator FnFactory \*, 153
- ArcSec::EvaluatorLoader, 153
  - getEvaluator, 153, 154

- getPolicy, 154
- getRequest, 154
- ArcSec::FnFactory, 165
  - createFn, 166
- ArcSec::Function, 166
  - evaluate, 167
- ArcSec::GenericAttribute, 167
  - encode, 167
  - equal, 167
  - getId, 168
  - getType, 168
- ArcSec::InRangeFunction, 179
  - evaluate, 179
- ArcSec::MatchFunction, 201
  - evaluate, 201
  - getFunctionName, 201
- ArcSec::OrderedCombiningAlg, 228
- ArcSec::PDP, 242
- ArcSec::PDPConfigContext, 242
- ArcSec::PDPPluginArgument, 243
- ArcSec::PeriodAttribute, 245
  - encode, 245
  - equal, 245
  - getId, 245
  - getType, 246
- ArcSec::PermitOverridesCombiningAlg, 246
  - combine, 246
  - getalgId, 247
- ArcSec::Policy, 254
  - addPolicy, 255
  - eval, 255
  - getEffect, 255
  - getEvalName, 255
  - getEvalResult, 255
  - getName, 255
  - make\_policy, 256
  - Policy, 255
  - setEvalResult, 256
  - setEvaluatorContext, 256
- ArcSec::PolicyParser, 256
  - parsePolicy, 257
- ArcSec::PolicyStore, 257
  - PolicyStore, 257
- ArcSec::PolicyStore::PolicyElement, 256
- ArcSec::Request, 263
  - addRequestItem, 264
  - getEvalName, 264
  - getName, 264
  - getRequestItems, 264
  - make\_request, 264
  - Request, 264
  - setAttributeFactory, 264
  - setRequestItems, 265
- ArcSec::RequestAttribute, 265
  - duplicate, 265
  - RequestAttribute, 265
- ArcSec::RequestItem, 266
  - RequestItem, 266
- ArcSec::RequestTuple, 266
- ArcSec::Response, 267
- ArcSec::ResponseItem, 267
- ArcSec::ResponseList, 267
- ArcSec::SecHandler, 283
- ArcSec::SecHandlerConfig, 284
- ArcSec::SecHandlerPluginArgument, 284
- ArcSec::Security, 284
- ArcSec::Source, 305
  - Source, 305
- ArcSec::SourceFile, 306
- ArcSec::SourceURL, 306
- ArcSec::StringAttribute, 307
  - encode, 307
  - equal, 307
  - getId, 307
  - getType, 308
- ArcSec::TimeAttribute, 321
  - encode, 322
  - equal, 322
  - getId, 322
  - getType, 322
- ArcSec::X500NameAttribute, 389
  - encode, 389
  - equal, 389
  - getId, 389
  - getType, 389
- ARCUSERDIRECTORY
  - Arc::UserConfig, 359
- Assign
  - Arc::InformationContainer, 175
- AssignStderr
  - Arc::Run, 272
- AssignStdin
  - Arc::Run, 272
- AssignStdout
  - Arc::Run, 272
- AssignWorkingDirectory
  - Arc::Run, 272
- AttrConstIter
  - Arc, 36
- Attribute
  - Arc::XMLNode, 395
- AttributeIterator
  - Arc::AttributeIterator, 53
- Attributes
  - Arc::Message, 211
  - Arc::SOAPMessage, 290
- attributes\_
  - Arc::MessageAttributes, 215

- AttributesSize
  - Arc::XMLNode, 395
- AttrIter
  - Arc, 36
- AttrMap
  - Arc, 36
- Auth
  - Arc::Message, 211
- AuthContext
  - Arc::Message, 211
- Authenticate
  - Arc::SAMLToken, 279
  - Arc::UsernameToken, 362
  - Arc::X509Token, 391
- Backup
  - Arc::DelegationConsumer, 137
- Bartender
  - Arc::UserConfig, 339, 340
- BaseDN2Path
  - Arc::URL, 325
- broadcast
  - Arc::SimpleCondition, 288
- Broker
  - Arc::UserConfig, 340, 341
- BrokerLoader
  - Arc::BrokerLoader, 64
- Buffer
  - Arc::PayloadRaw, 230
  - Arc::PayloadRawInterface, 233
- buffer\_size
  - Arc::DataBuffer, 102
- BufferPos
  - Arc::PayloadRaw, 230
  - Arc::PayloadRawInterface, 233
- BufferSize
  - Arc::PayloadRaw, 230
  - Arc::PayloadRawInterface, 233
- BUSY\_ERROR
  - Arc, 37
- CACertificatePath
  - Arc::UserConfig, 341, 342
- CACertificatesDirectory
  - Arc::UserConfig, 342
- CacheError
  - Arc::DataStatus, 133
- cancel
  - Arc::Counter, 84
  - Arc::CounterTicket, 89
  - Arc::IntraProcessCounter, 181
- CERT\_TYPE\_CA
  - ArcCredential, 44
- CERT\_TYPE\_EEC
  - ArcCredential, 44
- CERT\_TYPE\_GSI\_2\_LIMITED\_PROXY
  - ArcCredential, 44
- CERT\_TYPE\_GSI\_2\_PROXY
  - ArcCredential, 44
- CERT\_TYPE\_GSI\_3\_IMPERSONATION\_PROXY
  - ArcCredential, 44
- CERT\_TYPE\_GSI\_3\_INDEPENDENT\_PROXY
  - ArcCredential, 44
- CERT\_TYPE\_GSI\_3\_LIMITED\_PROXY
  - ArcCredential, 44
- CERT\_TYPE\_GSI\_3\_RESTRICTED\_PROXY
  - ArcCredential, 44
- CERT\_TYPE\_RFC\_ANYLANGUAGE\_PROXY
  - ArcCredential, 44
- CERT\_TYPE\_RFC\_IMPERSONATION\_PROXY
  - ArcCredential, 44
- CERT\_TYPE\_RFC\_INDEPENDENT\_PROXY
  - ArcCredential, 44
- CERT\_TYPE\_RFC\_LIMITED\_PROXY
  - ArcCredential, 44
- CERT\_TYPE\_RFC\_RESTRICTED\_PROXY
  - ArcCredential, 44
- CertificateLifeTime
  - Arc::UserConfig, 343
- CertificatePath
  - Arc::UserConfig, 343, 344
- certType
  - ArcCredential, 44
- changeExcess
  - Arc::Counter, 84
  - Arc::IntraProcessCounter, 181
- ChangeHost
  - Arc::URL, 325
- ChangeLDAPFilter
  - Arc::URL, 325
- ChangeLDAPScope
  - Arc::URL, 325
- changeLimit
  - Arc::Counter, 84
  - Arc::IntraProcessCounter, 181
- ChangePath
  - Arc::URL, 325
- ChangePort
  - Arc::URL, 326
- ChangeProtocol
  - Arc::URL, 326
- Check
  - Arc::DataPoint, 112
  - Arc::DataPointIndex, 125
  - Arc::WSAHeader, 367
- CheckCreated
  - Arc::FileCache, 161

- CheckDN
  - Arc::FileCache, 161
- CheckError
  - Arc::DataStatus, 133
- checks
  - Arc::DataMover, 107
- checksum\_object
  - Arc::DataBuffer, 102
- checksum\_valid
  - Arc::DataBuffer, 102
- CheckValid
  - Arc::FileCache, 161
- Child
  - Arc::XMLNode, 395
- CISStringValue
  - Arc::CISStringValue, 68
- clear
  - Arc::SoftwareRequirement, 300
- ClearRejectedServices
  - Arc::UserConfig, 344
- ClearSelectedServices
  - Arc::UserConfig, 345
- ClientHTTPwithSAML2SSO
  - Arc::ClientHTTPwithSAML2SSO, 70
- ClientSOAP
  - Arc::ClientSOAP, 72
- ClientSOAPwithSAML2SSO
  - Arc::ClientSOAPwithSAML2SSO, 73
- ClientX509Delegation
  - Arc::ClientX509Delegation, 74
- close
  - Arc::Database, 99
  - Arc::MySQLDatabase, 222
- CloseStderr
  - Arc::Run, 272
- CloseStdin
  - Arc::Run, 272
- CloseStdout
  - Arc::Run, 272
- combine
  - ArcSec::CombiningAlg, 76
  - ArcSec::DenyOverridesCombiningAlg, 145
  - ArcSec::PermitOverridesCombiningAlg, 246
- CommonLocOption
  - Arc::URL, 326
- CommonLocOptions
  - Arc::URL, 326
- commonlocoptions
  - Arc::URL, 329
- CompareLocationMetadata
  - Arc::DataPoint, 112
  - Arc::DataPointDirect, 119
  - Arc::DataPointIndex, 125
- CompareMeta
  - Arc::DataPoint, 112
- ComparisonOperator
  - Arc::Software, 291
- ComparisonOperatorEnum
  - Arc::Software, 292
- ComputingShareName
  - Arc::ExecutionTarget, 156
- Config
  - Arc::Config, 77
- ConfusaCertHandler
  - Arc::ConfusaCertHandler, 79
- connect
  - Arc::Database, 99
  - Arc::MySQLDatabase, 222
- ConnectionURL
  - Arc::URL, 326
- Content
  - Arc::PayloadRaw, 230
  - Arc::PayloadRawInterface, 233
- ContentFromPayload
  - Arc, 37
- Context
  - Arc::Message, 212
- context\_lock\_
  - Arc::DelegationContainerSOAP, 140
- convert
  - Arc::Software, 293
- Copy
  - Arc::FileCache, 161
- count
  - Arc::MessageAttributes, 213
- Counter
  - Arc::Counter, 83
- CounterTicket
  - Arc::CounterTicket, 89
- createAlg
  - ArcSec::AlgFactory, 48
- createCertRequest
  - Arc::ConfusaCertHandler, 79
- createDelegation
  - Arc::ClientX509Delegation, 75
- createFn
  - ArcSec::FnFactory, 166
- CreateThreadFunction
  - Arc, 37
- createVOMSAC
  - Arc, 37
- Credential
  - Arc::Credential, 91, 92
- CredentialError
  - Arc::CredentialError, 98
- CredentialLogger
  - Arc, 43
- CredentialsExpiredError

- Arc::DataStatus, 133
- CredentialsFound
  - Arc::UserConfig, 345
- current\_
  - Arc::AttributeIterator, 55
- CurrentLocationMetadata
  - Arc::DataPoint, 112
  - Arc::DataPointDirect, 120
  - Arc::DataPointIndex, 125
- Database
  - Arc::Database, 99
- DataBuffer
  - Arc::DataBuffer, 101
- DataPoint
  - Arc::DataPoint, 111
- DataPointAccessLatency
  - Arc::DataPoint, 111
- DataPointInfoType
  - Arc::DataPoint, 111
- DataSpeed
  - Arc::DataSpeed, 130
- DataStatusType
  - Arc::DataStatus, 133
- DecryptNode
  - Arc::XMLSecNode, 406
- DEFAULT\_BROKER
  - Arc::UserConfig, 359
- DEFAULT\_TIMEOUT
  - Arc::UserConfig, 359
- DEFAULTCONFIG
  - Arc::UserConfig, 360
- Delegate
  - Arc::DelegationProvider, 142
- DelegateCredentialsInit
  - Arc::DelegationConsumerSOAP, 139
  - Arc::DelegationContainerSOAP, 140
  - Arc::DelegationProviderSOAP, 143
- DelegatedToken
  - Arc::DelegationConsumerSOAP, 139
  - Arc::DelegationContainerSOAP, 140
  - Arc::DelegationProviderSOAP, 143
- DelegationConsumer
  - Arc::DelegationConsumer, 137
- DelegationConsumerSOAP
  - Arc::DelegationConsumerSOAP, 139
- DelegationProvider
  - Arc::DelegationProvider, 141
- DelegationProviderSOAP
  - Arc::DelegationProviderSOAP, 143
- DeleteError
  - Arc::DataStatus, 133
- Destroy
  - Arc::XMLNode, 395
- destroy\_doc
  - Arc::ConfusaParserUtils, 79
- doc\_
  - Arc::InformationContainer, 175
- duplicate
  - ArcSec::RequestAttribute, 265
- empty
  - Arc::Software, 293
  - Arc::SoftwareRequirement, 300
- enable\_ssl
  - Arc::Database, 99
  - Arc::MySQLDatabase, 222
- encode
  - ArcSec::AnyURIAttribute, 49
  - ArcSec::AttributeValue, 57
  - ArcSec::BooleanAttribute, 62
  - ArcSec::DateAttribute, 134
  - ArcSec::DateTimeAttribute, 136
  - ArcSec::DurationAttribute, 147
  - ArcSec::GenericAttribute, 167
  - ArcSec::PeriodAttribute, 245
  - ArcSec::StringAttribute, 307
  - ArcSec::TimeAttribute, 322
  - ArcSec::X500NameAttribute, 389
- EncryptNode
  - Arc::XMLSecNode, 406
- end\_
  - Arc::AttributeIterator, 55
- eof\_read
  - Arc::DataBuffer, 102
- eof\_write
  - Arc::DataBuffer, 103
- EQUAL
  - Arc::Software, 292
- equal
  - Arc::CIStringValue, 68
  - ArcSec::AnyURIAttribute, 49
  - ArcSec::AttributeValue, 58
  - ArcSec::BooleanAttribute, 62
  - ArcSec::DateAttribute, 134
  - ArcSec::DateTimeAttribute, 136
  - ArcSec::DurationAttribute, 147
  - ArcSec::GenericAttribute, 167
  - ArcSec::PeriodAttribute, 245
  - ArcSec::StringAttribute, 307
  - ArcSec::TimeAttribute, 322
  - ArcSec::X500NameAttribute, 389
- error
  - Arc::DataBuffer, 103
- error\_read
  - Arc::DataBuffer, 103
- error\_write
  - Arc::DataBuffer, 103

- eval
  - ArcSec::Policy, 255
- evaluate
  - ArcSec::EqualFunction, 148
  - ArcSec::Evaluator, 151
  - ArcSec::Function, 167
  - ArcSec::InRangeFunction, 179
  - ArcSec::MatchFunction, 201
- evaluate\_path
  - Arc::ConfusaParserUtils, 79
- EvaluationCtx
  - ArcSec::EvaluationCtx, 149
- EXAMPLECONFIG
  - Arc::UserConfig, 360
- Exchange
  - Arc::XMLNode, 396
- execute
  - Arc::MySQLQuery, 223
  - Arc::Query, 260
- ExecutionTarget
  - Arc::ExecutionTarget, 155
- Export
  - Arc::MessageAuth, 216
  - Arc::MultiSecAttr, 221
  - Arc::SecAttr, 281
- extend
  - Arc::Counter, 84
  - Arc::CounterTicket, 89
  - Arc::IntraProcessCounter, 181
- extract\_body\_information
  - Arc::ConfusaParserUtils, 80
- factory\_
  - Arc::Loader, 191
- FaultTo
  - Arc::WSAHeader, 367
- File
  - Arc::FileCache, 162
- FileCache
  - Arc::FileCache, 160, 161
- FileCacheHash, 164
  - getHash, 164
  - maxLength, 164
- FileOpen
  - Arc, 38
- FillJobStore
  - Arc::JobController, 186
- Filter
  - Arc::InfoFilter, 171
  - Arc::MessageAuth, 216
- final\_xmlsec
  - Arc, 38
- find
  - Arc::ModuleManager, 219
- findLocation
  - Arc::ModuleManager, 219
- findSimpleSAMLInstallation
  - Arc::SAML2LoginClient, 275
- for\_read
  - Arc::DataBuffer, 103
- for\_write
  - Arc::DataBuffer, 103, 104
- force\_to\_meta
  - Arc::DataMover, 107
- FoundJobs
  - Arc::TargetGenerator, 312
- FoundTargets
  - Arc::TargetGenerator, 312
- FreeSlotsWithDuration
  - Arc::ExecutionTarget, 157
- From
  - Arc::WSAHeader, 368
- FullName
  - Arc::XMLNode, 396
- FullPath
  - Arc::URL, 326
- fullstr
  - Arc::URL, 326
  - Arc::URLLocation, 332
- Generate
  - Arc::DelegationConsumer, 137
- GenerateEECRequest
  - Arc::Credential, 93
- GenerateRequest
  - Arc::Credential, 93
- GENERIC\_ERROR
  - Arc, 36
- Get
  - Arc::InfoCacheInterface, 170
  - Arc::InformationContainer, 175
  - Arc::InformationInterface, 176
  - Arc::PayloadStream, 236
  - Arc::PayloadStreamInterface, 239
  - Arc::XMLNode, 396
- get
  - Arc::MessageAttributes, 214
- get\_array
  - Arc::MySQLQuery, 223
  - Arc::Query, 260
- get\_cert\_str
  - Arc, 38
- get\_doc
  - Arc::ConfusaParserUtils, 80
- get\_factory
  - Arc::PluginArgument, 251
- get\_key\_from\_certfile
  - Arc, 38

- get\_key\_from\_certstr
  - Arc, 38
- get\_key\_from\_keyfile
  - Arc, 38
- get\_key\_from\_keyst
  - Arc, 38
- get\_module
  - Arc::PluginArgument, 251
- get\_node
  - Arc, 38
- get\_num\_columns
  - Arc::MySQLQuery, 223
  - Arc::Query, 260
- get\_num\_rows
  - Arc::MySQLQuery, 224
  - Arc::Query, 260
- get\_plugin\_instance
  - Arc, 36
- get\_property
  - Arc, 38
- get\_row
  - Arc::MySQLQuery, 224
  - Arc::Query, 260, 261
- get\_row\_field
  - Arc::MySQLQuery, 224
  - Arc::Query, 261
- getAlgFactory
  - ArcSec::Evaluator, 151
- getalgId
  - ArcSec::CombiningAlg, 76
  - ArcSec::DenyOverridesCombiningAlg, 145
  - ArcSec::PermitOverridesCombiningAlg, 247
- getAll
  - Arc::MessageAttributes, 214
- getAttrFactory
  - ArcSec::Evaluator, 152
- getAttribute
  - ArcSec::AttributeProxy, 56
- GetBestTarget
  - Arc::Broker, 63
- GetBrokers
  - Arc::BrokerLoader, 65
- GetCert
  - Arc::Credential, 93
- GetCertNumofChain
  - Arc::Credential, 93
- GetCertReq
  - Arc::Credential, 94
- getCertRequestB64
  - Arc::ConfusaCertHandler, 79
- getComparisonOperatorList
  - Arc::SoftwareRequirement, 300
- getCounterTicket
  - Arc::Counter, 85
- GetCreated
  - Arc::FileCache, 162
- getCurrentTime
  - Arc::Counter, 85
- GetDN
  - Arc::Credential, 94
- GetDoc
  - Arc::XMLNode, 396
- getEffect
  - ArcSec::Policy, 255
- GetEndTime
  - Arc::Credential, 94
- GetEntry
  - Arc::ClientSOAP, 72
- getEvalName
  - ArcSec::Policy, 255
  - ArcSec::Request, 264
- getEvalResult
  - ArcSec::Policy, 255
- getEvaluator
  - ArcSec::EvaluatorLoader, 153, 154
- getExcess
  - Arc::Counter, 85
  - Arc::IntraProcessCounter, 182
- getExpirationReminder
  - Arc::Counter, 85
- getExpiryTime
  - Arc::Counter, 86
  - Arc::ExpirationReminder, 158
- getExplanation
  - Arc::MCC\_Status, 205
- GetFailureReason
  - Arc::DataPoint, 113
- getFamily
  - Arc::Software, 293
- getFileName
  - Arc::Config, 78
- getFnFactory
  - ArcSec::Evaluator, 152
- GetFormat
  - Arc::Time, 320
- getFormat
  - Arc::Credential, 94
- getFunctionName
  - ArcSec::EqualFunction, 148
  - ArcSec::MatchFunction, 201
- getHash
  - FileCacheHash, 164
- getID
  - Arc::Service, 286
- getId
  - ArcSec::AnyURIAttribute, 49
  - ArcSec::AttributeValue, 58
  - ArcSec::BooleanAttribute, 62

- ArcSec::DateAttribute, 135
- ArcSec::DateTimeAttribute, 136
- ArcSec::DurationAttribute, 147
- ArcSec::GenericAttribute, 168
- ArcSec::PeriodAttribute, 245
- ArcSec::StringAttribute, 307
- ArcSec::TimeAttribute, 322
- ArcSec::X500NameAttribute, 389
- GetIdentityName
  - Arc::Credential, 94
- GetJobControllers
  - Arc::JobControllerLoader, 188
  - Arc::JobSupervisor, 190
- getKind
  - Arc::MCC\_Status, 205
- getLevel
  - Arc::LogMessage, 198
- GetLifeTime
  - Arc::Credential, 94
- getLimit
  - Arc::Counter, 86
  - Arc::IntraProcessCounter, 182
- getName
  - Arc::Software, 293
  - ArcSec::Evaluator, 152
  - ArcSec::Policy, 255
  - ArcSec::Request, 264
- getOrigin
  - Arc::MCC\_Status, 205
- GetOverlay
  - Arc::BaseConfig, 61
- GetPeriod
  - Arc::Period, 244
- GetPlugins
  - Arc::ArcLocation, 51
- getPolicy
  - ArcSec::EvaluatorLoader, 154
- GetPrivKey
  - Arc::Credential, 94
- GetProxyPolicy
  - Arc::Credential, 94
- GetPubKey
  - Arc::Credential, 94
- GetRejectedServices
  - Arc::UserConfig, 345
- getRequest
  - ArcSec::EvaluatorLoader, 154
- getRequestItems
  - ArcSec::Request, 264
- getReservationID
  - Arc::ExpirationReminder, 158
- GetRoot
  - Arc::XMLNode, 396
- getRootLogger
  - Arc::Logger, 196
- GetSelectedServices
  - Arc::UserConfig, 346
- getSoftwareList
  - Arc::SoftwareRequirement, 300
- GetStartTime
  - Arc::Credential, 94
- GetSubmitter
  - Arc::ExecutionTarget, 156
- GetSubmitters
  - Arc::SubmitterLoader, 310
- GetTargetRetrievers
  - Arc::TargetRetrieverLoader, 315
- GetTargets
  - Arc::TargetGenerator, 312
  - Arc::TargetRetriever, 314
- getThreshold
  - Arc::Logger, 196
- GetTime
  - Arc::Time, 320
- GetType
  - Arc::Credential, 94
- getType
  - ArcSec::AnyURIAttribute, 49
  - ArcSec::AttributeValue, 58
  - ArcSec::BooleanAttribute, 62
  - ArcSec::DateAttribute, 135
  - ArcSec::DateTimeAttribute, 136
  - ArcSec::DurationAttribute, 147
  - ArcSec::GenericAttribute, 168
  - ArcSec::PeriodAttribute, 246
  - ArcSec::StringAttribute, 308
  - ArcSec::TimeAttribute, 322
  - ArcSec::X500NameAttribute, 389
- GetValid
  - Arc::FileCache, 162
- getValue
  - Arc::Counter, 86
  - Arc::IntraProcessCounter, 182
- GetVerification
  - Arc::Credential, 95
- getVersion
  - Arc::Software, 294
- GetXML
  - Arc::XMLNode, 396
- GREATERTHAN
  - Arc::Software, 292
- GREATERTHANOEQUAL
  - Arc::Software, 292
- GUID
  - Arc, 39
- handle\_
  - Arc::PayloadStream, 238

- handle\_redirect\_step
  - Arc::ConfusaParserUtils, 80
- hasMore
  - Arc::AttributeIterator, 54
- header\_allocated\_
  - Arc::WSAHeader, 369
- hold
  - Arc::DataSpeed, 130
- Host
  - Arc::URL, 326
- host
  - Arc::URL, 329
- HTTPOption
  - Arc::URL, 326
- HTTPOptions
  - Arc::URL, 327
- httpoptions
  - Arc::URL, 329
- ID
  - Arc::DelegationConsumer, 138
  - Arc::DelegationProviderSOAP, 143
- IdPName
  - Arc::UserConfig, 346, 347
- IDType
  - Arc::Counter, 83
- Import
  - Arc::SecAttr, 281
- InconsistentMetadataError
  - Arc::DataStatus, 133
- INFO\_TYPE\_ACCESS
  - Arc::DataPoint, 111
- INFO\_TYPE\_ALL
  - Arc::DataPoint, 111
- INFO\_TYPE\_CONTENT
  - Arc::DataPoint, 111
- INFO\_TYPE\_NAME
  - Arc::DataPoint, 111
- INFO\_TYPE\_REST
  - Arc::DataPoint, 111
- INFO\_TYPE\_STRUCT
  - Arc::DataPoint, 111
- INFO\_TYPE\_TIMES
  - Arc::DataPoint, 111
- INFO\_TYPE\_TYPE
  - Arc::DataPoint, 111
- InfoCache
  - Arc::InfoCache, 170
- InfoFilter
  - Arc::InfoFilter, 171
- InfoRegisters
  - Arc::InfoRegisters, 173
- InformationContainer
  - Arc::InformationContainer, 175
- InformationInterface
  - Arc::InformationInterface, 176
- InformationRequest
  - Arc::InformationRequest, 177
- InformationResponse
  - Arc::InformationResponse, 178
- Init
  - Arc::ArcLocation, 51
- init\_xmlsec
  - Arc, 39
- InitializeCredentials
  - Arc::UserConfig, 347
- InitProxyCertInfo
  - Arc::Credential, 95
- InquireRequest
  - Arc::Credential, 95
- Insert
  - Arc::PayloadRaw, 231
  - Arc::PayloadRawInterface, 233
- IntraProcessCounter
  - Arc::IntraProcessCounter, 180
- ip6addr
  - Arc::URL, 329
- is\_notwritten
  - Arc::DataBuffer, 104
- is\_owner\_
  - Arc::XMLNode, 403
- is\_read
  - Arc::DataBuffer, 104
- is\_temporary\_
  - Arc::XMLNode, 403
- is\_written
  - Arc::DataBuffer, 105
- isconnected
  - Arc::Database, 100
  - Arc::MySQLDatabase, 222
- isOk
  - Arc::MCC\_Status, 205
- IsReadingError
  - Arc::DataStatus, 133
- isRequiringAll
  - Arc::SoftwareRequirement, 301
- isResolved
  - Arc::SoftwareRequirement, 301
- isSatisfied
  - Arc::SoftwareRequirement, 301, 302
- IsSecureProtocol
  - Arc::URL, 327
- istr
  - Arc::Period, 244
- istring\_to\_level
  - Arc, 39
- isValid
  - Arc::CounterTicket, 89

- IsWritingError
  - Arc::DataStatus, 133
- Job
  - Arc::Job, 185
- JobControllerLoader
  - Arc::JobControllerLoader, 187
- JobListFile
  - Arc::UserConfig, 348
- JobSupervisor
  - Arc::JobSupervisor, 190
- KeepStderr
  - Arc::Run, 272
- KeepStdin
  - Arc::Run, 273
- KeepStdout
  - Arc::Run, 273
- key
  - Arc::AttributeIterator, 54
- KeyPassword
  - Arc::UserConfig, 349
- KeyPath
  - Arc::UserConfig, 349, 350
- KeySize
  - Arc::UserConfig, 350
- Kill
  - Arc::Run, 273
- LDAPAttributes
  - Arc::URL, 327
- ldapattributes
  - Arc::URL, 329
- LDAPFilter
  - Arc::URL, 327
- ldapfilter
  - Arc::URL, 330
- LDAPScope
  - Arc::URL, 327
- ldapscope
  - Arc::URL, 330
- length
  - Arc::PayloadRawBuf, 232
- LESSTHAN
  - Arc::Software, 292
- LESSTHANOREQUAL
  - Arc::Software, 292
- Limit
  - Arc::PayloadStream, 236
  - Arc::PayloadStreamInterface, 239
- Link
  - Arc::FileCache, 162
- List
  - Arc::DataPoint, 113
- ListError
  - Arc::DataStatus, 133
- Load
  - Arc::ClientSOAP, 72
- load
  - Arc::BrokerLoader, 65
  - Arc::JobControllerLoader, 188
  - Arc::ModuleManager, 219
  - Arc::PluginsFactory, 253
  - Arc::SubmitterLoader, 310
  - Arc::TargetRetrieverLoader, 315
- load\_key\_from\_certfile
  - Arc, 39
- load\_key\_from\_certstr
  - Arc, 39
- load\_key\_from\_keyfile
  - Arc, 39
- load\_trusted\_cert\_file
  - Arc, 39
- load\_trusted\_cert\_str
  - Arc, 40
- load\_trusted\_certs
  - Arc, 40
- LoadConfigurationFile
  - Arc::UserConfig, 351
- Loader
  - Arc::Loader, 191
- LocationAlreadyExistsError
  - Arc::DataStatus, 133
- Locations
  - Arc::URL, 327
- locations
  - Arc::URL, 330
- lock
  - Arc::SimpleCondition, 288
- lock\_
  - Arc::InformationInterface, 176
- log
  - Arc::LogFile, 194
  - Arc::LogStream, 200
- LogDestination
  - Arc::LogDestination, 192
- LogError
  - Arc::Credential, 95
  - Arc::DelegationConsumer, 138
- LogFile
  - Arc::LogFile, 193
- Logger
  - Arc::Logger, 195
  - Arc::LogMessage, 199
- logger
  - Arc::MCC, 204
  - Arc::Plexer, 248
  - Arc::Service, 286

- LogLevel
  - Arc, 36
- LogMessage
  - Arc::LogMessage, 198
- LogStream
  - Arc::LogStream, 200
- make\_policy
  - ArcSec::Policy, 256
- make\_request
  - ArcSec::Request, 264
- MakeConfig
  - Arc::BaseConfig, 61
  - Arc::MCCConfig, 207
- makePersistent
  - Arc::ModuleManager, 219, 220
- match
  - Arc::RegularExpression, 263
- MatchXMLName
  - Arc, 40
  - Arc::XMLNode, 402
- MatchXMLNamespace
  - Arc, 40
  - Arc::XMLNode, 402, 403
- max\_duration\_
  - Arc::DelegationContainerSOAP, 140
- max\_size\_
  - Arc::DelegationContainerSOAP, 140
- max\_usage\_
  - Arc::DelegationContainerSOAP, 141
- MaxDiskSpace
  - Arc::ExecutionTarget, 157
- maxLength
  - FileCacheHash, 164
- MaxMainMemory
  - Arc::ExecutionTarget, 157
- MaxVirtualMemory
  - Arc::ExecutionTarget, 157
- MCC
  - Arc::MCC, 203
- MCC\_Status
  - Arc::MCC\_Status, 205
- MCCLoader
  - Arc::MCCLoader, 209
- Message
  - Arc::Message, 211
- MessageAttributes
  - Arc::AttributeIterator, 55
  - Arc::MessageAttributes, 213
- MessageID
  - Arc::WSAHeader, 368
- MetaData
  - Arc::WSAEndpointReference, 366
- MetaDataOption
  - Arc::URL, 327
- MetaDataOptions
  - Arc::URL, 327
- metadataoptions
  - Arc::URL, 330
- Migrate
  - Arc::JobController, 186
  - Arc::Submitter, 308
- ModifyFoundTargets
  - Arc::TargetGenerator, 313
- ModuleManager
  - Arc::ModuleManager, 219
- Move
  - Arc::XMLNode, 396
- msg
  - Arc::Logger, 196, 197
- Name
  - Arc::URLLocation, 332
  - Arc::XMLNode, 397
- name
  - Arc::URLLocation, 332
- Namespace
  - Arc::XMLNode, 397
- NamespacePrefix
  - Arc::XMLNode, 397
- Namespaces
  - Arc::XMLNode, 397
- New
  - Arc::XMLNode, 397
- NewAttribute
  - Arc::XMLNode, 397, 398
- NewChild
  - Arc::XMLNode, 398
- NewReferenceParameter
  - Arc::WSAHeader, 368
- Next
  - Arc::MCC, 203
  - Arc::Plexer, 248
- next\_
  - Arc::MCC, 204
- NextLocation
  - Arc::DataPoint, 113
  - Arc::DataPointDirect, 120
  - Arc::DataPointIndex, 125
- Nodes
  - Arc::XMLNodeContainer, 404
- NoLocationError
  - Arc::DataStatus, 133
- NOTEQUAL
  - Arc::Software, 292
- NotInitializedError
  - Arc::DataStatus, 133
- NotSupportedForDirectDataPointsError

- Arc::DataStatus, 133
- OAuthConsumer
  - Arc::OAuthConsumer, 226
- OpenSSLInit
  - Arc, 40
- OperatingSystem
  - Arc::ExecutionTarget, 157
- operator AlgFactory \*
  - ArcSec::EvaluatorContext, 153
- operator AttributeFactory \*
  - ArcSec::EvaluatorContext, 153
- operator bool
  - Arc::CStringValue, 68
  - Arc::FileCache, 162
  - Arc::MCC\_Status, 206
  - Arc::MultiSecAttr, 221
  - Arc::PathIterator, 229
  - Arc::PayloadStream, 237
  - Arc::PayloadStreamInterface, 239
  - Arc::Run, 273
  - Arc::SAMLToken, 279
  - Arc::SecAttr, 281
  - Arc::SecAttrValue, 282
  - Arc::URL, 327
  - Arc::UserConfig, 352
  - Arc::UsernameToken, 362
  - Arc::WSRF, 371
  - Arc::X509Token, 391
  - Arc::XMLNode, 398
- operator FnFactory \*
  - ArcSec::EvaluatorContext, 153
- operator PluginsFactory \*
  - Arc::ChainContext, 66
- operator std::string
  - Arc::MCC\_Status, 206
  - Arc::Period, 244
  - Arc::Software, 294
  - Arc::Time, 320
  - Arc::XMLNode, 398
- operator XMLNode
  - Arc::WSAEndpointReference, 366
  - Arc::WSAHeader, 368
- operator <
  - Arc::ExpirationReminder, 158
  - Arc::Period, 244
  - Arc::Software, 295
  - Arc::Time, 320
  - Arc::URL, 328
- operator <<
  - Arc, 40, 41
  - Arc::LogMessage, 199
  - Arc::Software, 297
  - Arc::URL, 329
- operator <=
  - Arc::Period, 244
  - Arc::Software, 295
  - Arc::Time, 320
- operator >
  - Arc::Period, 244
  - Arc::Software, 296
  - Arc::Time, 321
- operator >=
  - Arc::Period, 244
  - Arc::Software, 296
  - Arc::Time, 321
- operator \*
  - Arc::AttributeIterator, 54
  - Arc::PathIterator, 229
- operator ()
  - Arc::Software, 294
- operator +
  - Arc::Time, 320
- operator ++
  - Arc::AttributeIterator, 54
  - Arc::PathIterator, 229
  - Arc::XMLNode, 399
- operator -
  - Arc::Time, 320
- operator >
  - Arc::AttributeIterator, 55
- operator --
  - Arc::PathIterator, 229
  - Arc::XMLNode, 399
- operator =
  - Arc::ExecutionTarget, 156
  - Arc::Message, 212
  - Arc::Period, 244
  - Arc::SoftwareRequirement, 303
  - Arc::Time, 320, 321
  - Arc::WSAEndpointReference, 366
  - Arc::XMLNode, 399
  - Arc::XMLNodeContainer, 404
- operator ==
  - Arc::FileCache, 162
  - Arc::Period, 244
  - Arc::SecAttr, 281
  - Arc::SecAttrValue, 282
  - Arc::Software, 295
  - Arc::Time, 321
  - Arc::URL, 328
  - Arc::XMLNode, 400
- operator []
  - Arc::MCCLoader, 209
  - Arc::PayloadRaw, 231
  - Arc::PayloadRawInterface, 233
  - Arc::XMLNode, 400
  - Arc::XMLNodeContainer, 404

- Option
  - Arc::URL, 328
- Options
  - Arc::URL, 328
- OptionString
  - Arc::URL, 328
- OutputCertificate
  - Arc::Credential, 95
- OutputCertificateChain
  - Arc::Credential, 95
- OutputPrivateKey
  - Arc::Credential, 96
- OutputPublicKey
  - Arc::Credential, 96
- OverlayFile
  - Arc::UserConfig, 353
- Parent
  - Arc::XMLNode, 400
- parse
  - Arc::Config, 78
- parseDN
  - Arc::OAuthConsumer, 226
  - Arc::SAML2SSOHTTPClient, 276
- ParseOptions
  - Arc::URL, 328
- parsePolicy
  - ArcSec::PolicyParser, 257
- parseVOMSAC
  - Arc, 41, 42
- PARSING\_ERROR
  - Arc, 37
- Passive
  - Arc::DataPoint, 113
  - Arc::DataPointDirect, 120
  - Arc::DataPointIndex, 125
- passphrase\_callback
  - Arc, 42
- Passwd
  - Arc::URL, 328
- passwd, 228
  - Arc::URL, 330
- Password
  - Arc::UserConfig, 353, 354
- PasswordType
  - Arc::UsernameToken, 361
- Path
  - Arc::URL, 328
  - Arc::XMLNode, 401
- path
  - Arc::URL, 330
- Path2BaseDN
  - Arc::URL, 328
- PathIterator
  - Arc::PathIterator, 228
- Payload
  - Arc::Message, 212
  - Arc::SOAPMessage, 290
- PayloadRaw
  - Arc::PayloadRaw, 230
- PayloadSOAP
  - Arc::PayloadSOAP, 235
- PayloadStream
  - Arc::PayloadStream, 236
- PayloadWSRF
  - Arc::PayloadWSRF, 241
- Period
  - Arc::Period, 243
- plainstr
  - Arc::URL, 328
- Plexer
  - Arc::Plexer, 248
- plugins\_table\_name
  - Arc, 43
- PluginsFactory
  - Arc::PluginsFactory, 253
- Policy
  - ArcSec::Policy, 255
- PolicyStore
  - ArcSec::PolicyStore, 257
- Port
  - Arc::URL, 329
- port
  - Arc::URL, 330
- Pos
  - Arc::PayloadStream, 237
  - Arc::PayloadStreamInterface, 240
- PossibleTargets
  - Arc::Broker, 64
- PostRegister
  - Arc::DataPoint, 113
  - Arc::DataPointDirect, 120
- PostRegisterError
  - Arc::DataStatus, 133
- PreFilterTargets
  - Arc::Broker, 63
- Prefix
  - Arc::XMLNode, 401
- PreRegister
  - Arc::DataPoint, 114
  - Arc::DataPointDirect, 120
- PreRegisterError
  - Arc::DataStatus, 133
- PreUnregister
  - Arc::DataPoint, 114
  - Arc::DataPointDirect, 121
- Print
  - Arc::ExecutionTarget, 156

- Arc::Job, 185
- print
  - Arc::Config, 78
- PrintJobStatus
  - Arc::JobController, 186
- PrintTargetInfo
  - Arc::TargetGenerator, 313
- process
  - Arc::ClientHTTPwithSAML2SSO, 70
  - Arc::ClientSOAP, 72
  - Arc::ClientSOAPwithSAML2SSO, 73
  - Arc::MCC, 203
  - Arc::MCCInterface, 207
  - Arc::Plexer, 248
  - Test::TestService, 317
- processConsent
  - Arc::HakaClient, 169
  - Arc::OpenIdpClient, 227
  - Arc::SAML2SSOHTTPClient, 276
- processIdP2Confusa
  - Arc::HakaClient, 169
  - Arc::OpenIdpClient, 227
  - Arc::SAML2SSOHTTPClient, 276
- processIdPLogin
  - Arc::HakaClient, 169
  - Arc::OpenIdpClient, 227
  - Arc::SAML2SSOHTTPClient, 276
- processLogin
  - Arc::OAuthConsumer, 226
  - Arc::SAML2LoginClient, 275
  - Arc::SAML2SSOHTTPClient, 276
- ProcessSecHandlers
  - Arc::MCC, 203
  - Arc::Service, 286
- Protocol
  - Arc::URL, 329
- protocol
  - Arc::URL, 330
- PROTOCOL\_RECOGNIZED\_ERROR
  - Arc, 37
- ProvidesMeta
  - Arc::DataPoint, 114
  - Arc::DataPointDirect, 121
  - Arc::DataPointIndex, 125
- ProxyPath
  - Arc::UserConfig, 354
- pushCSR
  - Arc::OAuthConsumer, 226
  - Arc::SAML2SSOHTTPClient, 276
- Put
  - Arc::PayloadStream, 237
  - Arc::PayloadStreamInterface, 240
- Query
  - Arc::Query, 259
- Range
  - Arc::DataPoint, 114
  - Arc::DataPointDirect, 121
  - Arc::DataPointIndex, 126
- RC\_DEFAULT\_PORT
  - URL.h, 410
- ReadAcquireError
  - Arc::DataStatus, 133
- ReadError
  - Arc::DataStatus, 133
- ReadFromFile
  - Arc::XMLNode, 401
- ReadFromStream
  - Arc::XMLNode, 401
- ReadOutOfOrder
  - Arc::DataPoint, 114
  - Arc::DataPointDirect, 121
  - Arc::DataPointIndex, 126
- ReadResolveError
  - Arc::DataStatus, 133
- ReadStartError
  - Arc::DataStatus, 133
- ReadStderr
  - Arc::Run, 273
- ReadStdout
  - Arc::Run, 273
- ReadStopError
  - Arc::DataStatus, 133
- ReferenceParameter
  - Arc::WSAHeader, 368
- ReferenceParameters
  - Arc::WSAEndpointReference, 366
- Registered
  - Arc::DataPoint, 115
  - Arc::DataPointDirect, 121
  - Arc::DataPointIndex, 126
- RegisteredService
  - Arc::RegisteredService, 262
- registration
  - Arc::InfoRegistrar, 174
- RegistrationCollector
  - Arc::Service, 286
- RelatesTo
  - Arc::WSAHeader, 368
- RelationshipType
  - Arc::WSAHeader, 368
- Release
  - Arc::FileCache, 162
- reload
  - Arc::ModuleManager, 220
- remove
  - Arc::MessageAttributes, 214

- removeAll
  - Arc::MessageAttributes, 214
- removeService
  - Arc::InfoRegisterContainer, 172
- Replace
  - Arc::XMLNode, 401
- ReplyTo
  - Arc::WSAHeader, 369
- report
  - Arc::PluginsFactory, 253
- Request
  - Arc::DelegationConsumer, 138
  - ArcSec::Request, 264
- RequestAttribute
  - ArcSec::RequestAttribute, 265
- RequestItem
  - ArcSec::RequestItem, 266
- reserve
  - Arc::Counter, 87
  - Arc::IntraProcessCounter, 183
- reset
  - Arc::SimpleCondition, 288
- Resolve
  - Arc::DataPoint, 115
  - Arc::DataPointDirect, 122
- Rest
  - Arc::PathIterator, 229
- Restore
  - Arc::DelegationConsumer, 138
- restricted\_
  - Arc::DelegationContainerSOAP, 141
- Result
  - Arc::InformationResponse, 178
  - Arc::Run, 273
- Run
  - Arc::Run, 271
- Running
  - Arc::Run, 273
- Same
  - Arc::XMLNode, 401
- SAML2LoginClient
  - Arc::SAML2LoginClient, 274
- SAMLToken
  - Arc::SAMLToken, 278
- SAMLVersion
  - Arc::SAMLToken, 278
- save
  - Arc::Config, 78
- SaveToFile
  - Arc::UserConfig, 354
  - Arc::XMLNode, 401
- SaveToStream
  - Arc::XMLNode, 401
- scan
  - Arc::PluginsFactory, 253
- Scope
  - Arc::URL, 324
- sechandlers\_
  - Arc::MCC, 204
  - Arc::Service, 286
- secure
  - Arc::DataMover, 107
- seekable\_
  - Arc::PayloadStream, 238
- selectSoftware
  - Arc::SoftwareRequirement, 303, 304
- Service
  - Arc::Service, 286
- ServiceCounter
  - Arc::TargetGenerator, 313
- SESSION\_CLOSE
  - Arc, 37
- Set
  - Arc::XMLNode, 401
- set
  - Arc::DataBuffer, 105
  - Arc::MessageAttributes, 215
- set\_base
  - Arc::DataSpeed, 130
- set\_default\_max\_inactivity\_time
  - Arc::DataMover, 107
- set\_default\_min\_average\_speed
  - Arc::DataMover, 107
- set\_default\_min\_speed
  - Arc::DataMover, 107
- set\_max\_data
  - Arc::DataSpeed, 130
- set\_max\_inactivity\_time
  - Arc::DataSpeed, 131
- set\_min\_average\_speed
  - Arc::DataSpeed, 131
- set\_min\_speed
  - Arc::DataSpeed, 131
- set\_namespaces
  - Arc::WSRF, 371
  - Arc::WSRFBaseFault, 372
  - Arc::WSRP, 375
- set\_progress\_indicator
  - Arc::DataSpeed, 131
- SetAdditionalChecks
  - Arc::DataPoint, 115
  - Arc::DataPointDirect, 122
  - Arc::DataPointIndex, 126
- setAttributeFactory
  - ArcSec::Request, 264
- setBackups
  - Arc::LogFile, 194

- setCfg
  - Arc::ModuleManager, 220
- setCombiningAlg
  - ArcSec::Evaluator, 152
- setEvalResult
  - ArcSec::Policy, 256
- setEvaluatorContext
  - ArcSec::Policy, 256
- setExcess
  - Arc::Counter, 87
  - Arc::IntraProcessCounter, 183
- setFileName
  - Arc::Config, 78
- SetFormat
  - Arc::Time, 321
- setIdentifier
  - Arc::LogMessage, 199
- SetLifeTime
  - Arc::Credential, 96
- setLimit
  - Arc::Counter, 87
  - Arc::IntraProcessCounter, 183
- setMaxSize
  - Arc::LogFile, 194
- SetMeta
  - Arc::DataPoint, 115
  - Arc::DataPointIndex, 126
- SetPeriod
  - Arc::Period, 245
- SetProxyPolicy
  - Arc::Credential, 96
- setReopen
  - Arc::LogFile, 194
- setRequestItems
  - ArcSec::Request, 265
- setRequirement
  - Arc::SoftwareRequirement, 304
- SetSecure
  - Arc::DataPoint, 116
  - Arc::DataPointDirect, 122
  - Arc::DataPointIndex, 127
- SetStartTime
  - Arc::Credential, 96
- setThreshold
  - Arc::Logger, 197
- SetTime
  - Arc::Time, 321
- SetValid
  - Arc::FileCache, 162
- shutdown
  - Arc::Database, 100
  - Arc::MySQLDatabase, 222
- signal
  - Arc::SimpleCondition, 288
- signal\_nonblock
  - Arc::SimpleCondition, 288
- SignEECRequest
  - Arc::Credential, 96
- SignNode
  - Arc::XMLSecNode, 406
- SignRequest
  - Arc::Credential, 96, 97
- Size
  - Arc::PayloadRaw, 231
  - Arc::PayloadRawInterface, 234
  - Arc::PayloadStream, 237
  - Arc::PayloadStreamInterface, 240
  - Arc::XMLNode, 401
  - Arc::XMLNodeContainer, 404
- size
  - Arc::PayloadRawBuf, 232
- SLCS
  - Arc::UserConfig, 355
- SOAP
  - Arc::InformationRequest, 177
  - Arc::WSRF, 371
- SOAPMessage
  - Arc::SOAPMessage, 289
- Software
  - Arc::Software, 292
- SoftwareRequirement
  - Arc::SoftwareRequirement, 298, 299
- SortLocations
  - Arc::DataPoint, 116
  - Arc::DataPointDirect, 122
  - Arc::DataPointIndex, 127
- SortTargets
  - Arc::Broker, 63
- Source
  - ArcSec::Source, 305
- STACK\_OF
  - Arc::Credential, 97
- StageError
  - Arc::DataStatus, 133
- Start
  - Arc::FileCache, 163
  - Arc::Run, 273
- StartReading
  - Arc::DataPoint, 116
  - Arc::DataPointIndex, 127
- StartWriting
  - Arc::DataPoint, 116
  - Arc::DataPointIndex, 127
- Stat
  - Arc::DataPoint, 117
- StatError
  - Arc::DataStatus, 133
- STATUS\_OK

- Arc, 36
- StatusKind
  - Arc, 36
- Stop
  - Arc::FileCache, 163
- StopAndDelete
  - Arc::FileCache, 163
- StopReading
  - Arc::DataPoint, 117
  - Arc::DataPointIndex, 128
- StopWriting
  - Arc::DataPoint, 117
  - Arc::DataPointIndex, 128
- storeCert
  - Arc::OAuthConsumer, 226
  - Arc::SAML2SSOHTTPClient, 276
- StoreDirectory
  - Arc::UserConfig, 355, 356
- str
  - Arc::Time, 321
  - Arc::URL, 329
  - Arc::URLLocation, 332
- string
  - Arc, 42
- Submit
  - Arc::Submitter, 309
- SubmitterLoader
  - Arc::SubmitterLoader, 309
- Success
  - Arc::DataStatus, 133
- SuccessCached
  - Arc::DataStatus, 133
- Swap
  - Arc::XMLNode, 402
- SYSCONFIG
  - Arc::UserConfig, 360
- SYSCONFIGARCLOC
  - Arc::UserConfig, 360
- SystemError
  - Arc::DataStatus, 133
- TargetGenerator
  - Arc::TargetGenerator, 311
- TargetRetriever
  - Arc::TargetRetriever, 314
- TargetRetrieverLoader
  - Arc::TargetRetrieverLoader, 315
- Test::TestMCC, 316
- Test::TestService, 317
  - process, 317
- thread\_stacksize
  - Arc, 43
- Time
  - Arc::Time, 319
- Timeout
  - Arc::PayloadStream, 238
  - Arc::PayloadStreamInterface, 240
  - Arc::UserConfig, 356
- TimeStamp
  - Arc, 42
- To
  - Arc::WSAHeader, 369
- toString
  - Arc::Software, 296
- Transfer
  - Arc::DataMover, 107, 108
- transfer
  - Arc::DataSpeed, 131
- TransferError
  - Arc::DataStatus, 133
- Truncate
  - Arc::PayloadRaw, 231
  - Arc::PayloadRawInterface, 234
- TryLoad
  - Arc::PluginsFactory, 253
- UnimplementedError
  - Arc::DataStatus, 133
- UNKNOWN\_SERVICE\_ERROR
  - Arc, 37
- UnknownError
  - Arc::DataStatus, 133
- Unlink
  - Arc::MCC, 203
- unload
  - Arc::ModuleManager, 220
- unlock
  - Arc::SimpleCondition, 288
- Unregister
  - Arc::DataPoint, 117
  - Arc::DataPointDirect, 122
- UnregisterError
  - Arc::DataStatus, 133
- Update
  - Arc::ExecutionTarget, 156
- UpdateCredentials
  - Arc::DelegationConsumerSOAP, 139
  - Arc::DelegationContainerSOAP, 140
  - Arc::DelegationProviderSOAP, 143, 144
- URL
  - Arc::URL, 325
- URL.h, 409
  - RC\_DEFAULT\_PORT, 410
- urlencode
  - Arc::ConfusaParserUtils, 80
- urlencode\_params
  - Arc::ConfusaParserUtils, 80
- URLLocation

- Arc::URLLocation, 331, 332
- urloptions
  - Arc::URL, 330
- UserConfig
  - Arc::UserConfig, 336, 337
- UserName
  - Arc::UserConfig, 357
- Username
  - Arc::URL, 329
  - Arc::UsernameToken, 362
- username
  - Arc::URL, 330
- UsernameToken
  - Arc::UsernameToken, 361
- UtilsDirPath
  - Arc::UserConfig, 357
- valid
  - Arc::URL, 330
- valid\_
  - Arc::WSRF, 371
- valid\_url\_options
  - Arc::DataPoint, 118
- Validate
  - Arc::XMLNode, 402
- verbose
  - Arc::DataMover, 108
  - Arc::DataSpeed, 131, 132
- Verbosity
  - Arc::UserConfig, 358
- VerifyNode
  - Arc::XMLSecNode, 406
- VERSIONTOKENS
  - Arc::Software, 297
- VOMSDecode
  - Arc, 42
- VOMSServerPath
  - Arc::UserConfig, 358, 359
- VOMSTrustList
  - Arc::VOMSTrustList, 363, 364
- Wait
  - Arc::Run, 274
- wait
  - Arc::SimpleCondition, 288
  - Arc::SimpleCounter, 289
- wait\_any
  - Arc::DataBuffer, 105
- wait\_nonblock
  - Arc::SimpleCondition, 288
- WaitForExit
  - Arc::ThreadRegistry, 318
- WaitOrCancel
  - Arc::ThreadRegistry, 318
- WriteAcquireError
  - Arc::DataStatus, 133
- WriteError
  - Arc::DataStatus, 133
- WriteOutOfOrder
  - Arc::DataPoint, 117
  - Arc::DataPointDirect, 123
  - Arc::DataPointIndex, 128
- WriteResolveError
  - Arc::DataStatus, 133
- WriteStartError
  - Arc::DataStatus, 133
- WriteStdin
  - Arc::Run, 274
- WriteStopError
  - Arc::DataStatus, 133
- WSAEndpointReference
  - Arc::WSAEndpointReference, 365
- WSAFault
  - Arc, 37
- WSAFaultAssign
  - Arc, 42
- WSAFaultExtract
  - Arc, 42
- WSAFaultInvalidAddressingHeader
  - Arc, 37
- WSAFaultUnknown
  - Arc, 37
- WSAHeader
  - Arc::WSAHeader, 367
- WSRF
  - Arc::WSRF, 370
- WSRFBaseFault
  - Arc::WSRFBaseFault, 372
- WSRP
  - Arc::WSRP, 374
- WSRPFault
  - Arc::WSRPFault, 377
- WSRPResourcePropertyChangeFailure
  - Arc::WSRPResourcePropertyChangeFailure, 385
- X509Token
  - Arc::X509Token, 390
- X509TokenType
  - Arc::X509Token, 390
- XMLNode
  - Arc::XMLNode, 394, 395
- XMLNodeContainer
  - Arc::XMLNodeContainer, 404
- XMLSecNode
  - Arc::XMLSecNode, 405
- XPathLookup
  - Arc::XMLNode, 402