

IrisSupportLib

Version 1.0

Reference Guide



1 IrisSupportLib Reference Guide	1
2 IrisSupportLib NAMESPACE macros	5
3 Module Index	7
3.1 Modules	7
4 Hierarchical Index	9
4.1 Class Hierarchy	9
5 Class Index	11
5.1 Class List	11
6 File Index	13
6.1 File List	13
7 Module Documentation	15
7.1 Instance Flags	15
7.1.1 Detailed Description	15
7.2 IrisInstanceBuilder resource APIs	15
7.2.1 Detailed Description	16
7.2.2 Function Documentation	16
7.2.2.1 addNoValueRegister()	17
7.2.2.2 addParameter()	17
7.2.2.3 addRegister()	17
7.2.2.4 addStringParameter()	18
7.2.2.5 addStringRegister()	19
7.2.2.6 beginResourceGroup()	19
7.2.2.7 enhanceParameter()	20
7.2.2.8 enhanceRegister()	20
7.2.2.9 getResourceInfo()	20
7.2.2.10 setDefaultResourceDelegates()	21
7.2.2.11 setDefaultResourceReadDelegate() [1/3]	21
7.2.2.12 setDefaultResourceReadDelegate() [2/3]	21
7.2.2.13 setDefaultResourceReadDelegate() [3/3]	22
7.2.2.14 setDefaultResourceWriteDelegate() [1/3]	22
7.2.2.15 setDefaultResourceWriteDelegate() [2/3]	23
7.2.2.16 setDefaultResourceWriteDelegate() [3/3]	23
7.2.2.17 setNextSubRscId()	24
7.2.2.18 setPropertyCanonicalRnScheme()	24
7.2.2.19 setTag()	24
7.3 IrisInstanceBuilder event APIs	24
7.3.1 Detailed Description	25
7.3.2 Function Documentation	26

7.3.2.1 addEventSource() [1/2]	26
7.3.2.2 addEventSource() [2/2]	26
7.3.2.3 deleteEventSource()	26
7.3.2.4 enhanceEventSource()	27
7.3.2.5 finalizeRegisterReadEvent()	27
7.3.2.6 finalizeRegisterUpdateEvent()	27
7.3.2.7 getIrisInstanceEvent()	27
7.3.2.8 hasEventSource()	27
7.3.2.9 renameEventSource()	28
7.3.2.10 resetRegisterReadEvent()	28
7.3.2.11 resetRegisterUpdateEvent()	28
7.3.2.12 setDefaultEsCreateDelegate() [1/3]	28
7.3.2.13 setDefaultEsCreateDelegate() [2/3]	28
7.3.2.14 setDefaultEsCreateDelegate() [3/3]	29
7.3.2.15 setRegisterReadEvent() [1/2]	29
7.3.2.16 setRegisterReadEvent() [2/2]	30
7.3.2.17 setRegisterUpdateEvent() [1/2]	30
7.3.2.18 setRegisterUpdateEvent() [2/2]	31
7.4 IrisInstanceBuilder breakpoint APIs	32
7.4.1 Detailed Description	32
7.4.2 Function Documentation	33
7.4.2.1 getBreakpointInfo()	33
7.4.2.2 notifyBreakpointHit()	33
7.4.2.3 notifyBreakpointHitData()	33
7.4.2.4 notifyBreakpointHitRegister()	34
7.4.2.5 setBreakpointDeleteDelegate() [1/3]	34
7.4.2.6 setBreakpointDeleteDelegate() [2/3]	34
7.4.2.7 setBreakpointDeleteDelegate() [3/3]	35
7.4.2.8 setBreakpointSetDelegate() [1/3]	35
7.4.2.9 setBreakpointSetDelegate() [2/3]	35
7.4.2.10 setBreakpointSetDelegate() [3/3]	36
7.4.2.11 setHandleBreakpointHitDelegate() [1/3]	36
7.4.2.12 setHandleBreakpointHitDelegate() [2/3]	37
7.4.2.13 setHandleBreakpointHitDelegate() [3/3]	37
7.5 IrisInstanceBuilder memory APIs	37
7.5.1 Detailed Description	38
7.5.2 Function Documentation	39
7.5.2.1 addAddressTranslation()	39
7.5.2.2 addMemorySpace()	39
7.5.2.3 setDefaultAddressTranslateDelegate() [1/3]	39
7.5.2.4 setDefaultAddressTranslateDelegate() [2/3]	40
7.5.2.5 setDefaultAddressTranslateDelegate() [3/3]	40

7.5.2.6 setDefaultGetMemorySidebandInfoDelegate() [1/3]	41
7.5.2.7 setDefaultGetMemorySidebandInfoDelegate() [2/3]	41
7.5.2.8 setDefaultGetMemorySidebandInfoDelegate() [3/3]	42
7.5.2.9 setDefaultMemoryReadDelegate() [1/3]	42
7.5.2.10 setDefaultMemoryReadDelegate() [2/3]	43
7.5.2.11 setDefaultMemoryReadDelegate() [3/3]	43
7.5.2.12 setDefaultMemoryWriteDelegate() [1/3]	44
7.5.2.13 setDefaultMemoryWriteDelegate() [2/3]	44
7.5.2.14 setDefaultMemoryWriteDelegate() [3/3]	44
7.5.2.15 setPropertyCanonicalMsnScheme()	45
7.6 IrisInstanceBuilder image loading APIs	45
7.6.1 Detailed Description	46
7.6.2 Function Documentation	46
7.6.2.1 setLoadImageDataDelegate() [1/3]	46
7.6.2.2 setLoadImageDataDelegate() [2/3]	46
7.6.2.3 setLoadImageDataDelegate() [3/3]	47
7.6.2.4 setLoadImageFileDelegate() [1/3]	47
7.6.2.5 setLoadImageFileDelegate() [2/3]	47
7.6.2.6 setLoadImageFileDelegate() [3/3]	48
7.7 IrisInstanceBuilder image readData callback APIs	48
7.7.1 Detailed Description	48
7.7.2 Function Documentation	48
7.7.2.1 openImage()	48
7.8 IrisInstanceBuilder execution stepping APIs	49
7.8.1 Detailed Description	49
7.8.2 Function Documentation	49
7.8.2.1 setRemainingStepGetDelegate() [1/3]	49
7.8.2.2 setRemainingStepGetDelegate() [2/3]	50
7.8.2.3 setRemainingStepGetDelegate() [3/3]	50
7.8.2.4 setRemainingStepSetDelegate() [1/3]	50
7.8.2.5 setRemainingStepSetDelegate() [2/3]	51
7.8.2.6 setRemainingStepSetDelegate() [3/3]	51
7.8.2.7 setStepCountGetDelegate() [1/3]	51
7.8.2.8 setStepCountGetDelegate() [2/3]	52
7.8.2.9 setStepCountGetDelegate() [3/3]	52
7.9 Disassembler delegate functions	53
7.9.1 Detailed Description	53
7.9.2 Typedef Documentation	53
7.9.2.1 DisassembleOpcodeDelegate	53
7.9.2.2 GetCurrentDisassemblyModeDelegate	54
7.9.2.3 GetDisassemblyDelegate	54
7.9.3 Function Documentation	54

7.9.3.1 addDisassemblyMode()	54
7.9.3.2 attachTo()	54
7.9.3.3 IrisInstanceDisassembler()	54
7.9.3.4 setDisassembleOpcodeDelegate()	55
7.9.3.5 setGetCurrentModeDelegate()	55
7.9.3.6 setGetDisassemblyDelegate()	55
7.10 Semihosting data request flag constants	55
7.10.1 Detailed Description	55
8 Class Documentation	57
8.1 iris::IrisInstanceBuilder::AddressTranslationBuilder Class Reference	57
8.1.1 Detailed Description	57
8.1.2 Member Function Documentation	57
8.1.2.1 setTranslateDelegate() [1/3]	57
8.1.2.2 setTranslateDelegate() [2/3]	58
8.1.2.3 setTranslateDelegate() [3/3]	58
8.2 iris::IrisInstanceMemory::AddressTranslationInfoAndAccess Struct Reference	59
8.2.1 Detailed Description	59
8.3 iris::BreakpointHitInfo Struct Reference	59
8.4 iris::IrisInstanceBuilder::EventSourceBuilder Class Reference	59
8.4.1 Detailed Description	60
8.4.2 Member Function Documentation	60
8.4.2.1 addEnumElement() [1/2]	60
8.4.2.2 addEnumElement() [2/2]	60
8.4.2.3 addField()	61
8.4.2.4 addOption()	61
8.4.2.5 hasSideEffects()	62
8.4.2.6 removeEnumElement()	62
8.4.2.7 renameEnumElement()	62
8.4.2.8 setCounter()	63
8.4.2.9 setDescription()	63
8.4.2.10 setEventStreamCreateDelegate() [1/2]	63
8.4.2.11 setEventStreamCreateDelegate() [2/2]	64
8.4.2.12 setFormat()	64
8.4.2.13 setHidden()	64
8.4.2.14 setName()	65
8.5 iris::IrisInstanceEvent::EventSourceInfoAndDelegate Struct Reference	65
8.5.1 Detailed Description	65
8.6 iris::EventStream Class Reference	65
8.6.1 Detailed Description	68
8.6.2 Member Function Documentation	68
8.6.2.1 action()	68

8.6.2.2 addField() [1/5]	68
8.6.2.3 addField() [2/5]	69
8.6.2.4 addField() [3/5]	69
8.6.2.5 addField() [4/5]	69
8.6.2.6 addField() [5/5]	69
8.6.2.7 addFieldSlow() [1/5]	70
8.6.2.8 addFieldSlow() [2/5]	70
8.6.2.9 addFieldSlow() [3/5]	70
8.6.2.10 addFieldSlow() [4/5]	71
8.6.2.11 addFieldSlow() [5/5]	71
8.6.2.12 checkRangePc()	71
8.6.2.13 disable()	71
8.6.2.14 emitEventBegin() [1/2]	72
8.6.2.15 emitEventBegin() [2/2]	72
8.6.2.16 emitEventEnd()	72
8.6.2.17 enable()	72
8.6.2.18 flush()	73
8.6.2.19 getCountVal()	73
8.6.2.20 getEcInstId()	73
8.6.2.21 getEsId()	73
8.6.2.22 getEventSourceId()	73
8.6.2.23 getEventSourceInfo()	74
8.6.2.24 getProxiedByInstancId()	74
8.6.2.25 getState()	74
8.6.2.26 isCounter()	74
8.6.2.27 isEnabled()	74
8.6.2.28 IsProxiedByOtherInstance()	74
8.6.2.29 IsProxyForOtherInstance()	75
8.6.2.30 selfRelease()	75
8.6.2.31 setCounter()	75
8.6.2.32 setOptions()	75
8.6.2.33 setProperties()	76
8.6.2.34 setProxiedByInstancId()	76
8.6.2.35 setRanges()	76
8.6.3 Member Data Documentation	77
8.6.3.1 counter	77
8.6.3.2 irisInstance	77
8.6.3.3 proxiedByInstancId	77
8.7 iris::IrisInstanceBuilder::FieldBuilder Class Reference	77
8.7.1 Detailed Description	79
8.7.2 Member Function Documentation	79
8.7.2.1 addEnum()	79

8.7.2.2 addField()	80
8.7.2.3 addLogicalField()	80
8.7.2.4 addStringEnum()	80
8.7.2.5 getRscld() [1/2]	80
8.7.2.6 getRscld() [2/2]	81
8.7.2.7 parent()	81
8.7.2.8 setAddressOffset()	81
8.7.2.9 setBitWidth()	81
8.7.2.10 setBreakpointSupportInfo()	81
8.7.2.11 setCanonicalRn()	82
8.7.2.12 setCanonicalRnElfDwarf()	82
8.7.2.13 setCname()	82
8.7.2.14 setDescription()	83
8.7.2.15 setFormat()	83
8.7.2.16 setLsbOffset()	83
8.7.2.17 setName()	83
8.7.2.18 setParentRscld()	84
8.7.2.19 setReadDelegate() [1/3]	84
8.7.2.20 setReadDelegate() [2/3]	84
8.7.2.21 setReadDelegate() [3/3]	85
8.7.2.22 setResetData() [1/2]	85
8.7.2.23 setResetData() [2/2]	86
8.7.2.24 setResetDataFromContainer()	86
8.7.2.25 setResetString()	86
8.7.2.26 setRwMode()	87
8.7.2.27 setSubRscld()	87
8.7.2.28 setTag() [1/2]	87
8.7.2.29 setTag() [2/2]	87
8.7.2.30 setType()	89
8.7.2.31 setWriteDelegate() [1/3]	89
8.7.2.32 setWriteDelegate() [2/3]	89
8.7.2.33 setWriteDelegate() [3/3]	90
8.7.2.34 setWriteMask() [1/2]	90
8.7.2.35 setWriteMask() [2/2]	91
8.7.2.36 setWriteMaskFromContainer()	91
8.8 iris::IrisCConnection Class Reference	91
8.8.1 Detailed Description	92
8.9 iris::IrisClient Class Reference	92
8.9.1 Constructor & Destructor Documentation	93
8.9.1.1 IrisClient()	94
8.9.2 Member Function Documentation	94
8.9.2.1 connect() [1/2]	94

8.9.2.2 connect() [2/2]	94
8.9.2.3 connectCommandLine()	94
8.9.2.4 connectSocketFd()	95
8.9.2.5 disconnect()	95
8.9.2.6 disconnectAndWaitForChildToExit()	95
8.9.2.7 getConnectCommandLineHelp()	95
8.9.2.8 getIrisInstance()	95
8.9.2.9 initServiceServer()	95
8.9.2.10 loadPlugin()	95
8.9.2.11 processEvents()	95
8.9.2.12 setInstanceName()	96
8.9.2.13 setSleepOnDestructionMs()	96
8.9.2.14 spawnAndConnect()	96
8.9.2.15 stopWaitForEvent()	96
8.9.2.16 waitForEvent()	96
8.9.2.17 waitpidWithTimeout()	96
8.9.3 Member Data Documentation	97
8.9.3.1 connectionHelpStr	97
8.10 iris::IrisCommandLineParser Class Reference	97
8.10.1 Detailed Description	98
8.10.2 Constructor & Destructor Documentation	98
8.10.2.1 IrisCommandLineParser()	99
8.10.3 Member Function Documentation	99
8.10.3.1 addOption() [1/2]	99
8.10.3.2 addOption() [2/2]	99
8.10.3.3 clear()	99
8.10.3.4 defaultMessageFunc()	99
8.10.3.5 getDbI()	99
8.10.3.6 getHelpMessage()	100
8.10.3.7 getInt()	100
8.10.3.8 getMap()	100
8.10.3.9 getNonOptionArguments()	100
8.10.3.10 getUInt()	100
8.10.3.11 isSpecified()	100
8.10.3.12 noNonOptionArguments()	100
8.10.3.13 parseCommandLine()	100
8.10.3.14 pleaseSpecifyOneOf()	101
8.10.3.15 printErrorAndExit() [1/2]	101
8.10.3.16 printErrorAndExit() [2/2]	101
8.10.3.17 printMessage()	101
8.10.3.18 setMessageFunc()	101
8.10.3.19 setValue()	101

8.10.3.20 unsetValue()	102
8.11 iris::IrisEventEmitter< ARGS > Class Template Reference	102
8.11.1 Detailed Description	102
8.11.2 Member Function Documentation	102
8.11.2.1 operator()	102
8.12 iris::IrisEventRegistry Class Reference	103
8.12.1 Detailed Description	103
8.12.2 Member Function Documentation	103
8.12.2.1 addField()	103
8.12.2.2 addFieldSlow()	104
8.12.2.3 begin()	104
8.12.2.4 emitEventEnd()	104
8.12.2.5 empty()	104
8.12.2.6 end()	105
8.12.2.7 forEach()	105
8.12.2.8 registerEventStream()	105
8.12.2.9 unregisterEventStream()	105
8.13 iris::IrisEventStream Class Reference	106
8.13.1 Detailed Description	106
8.13.2 Member Function Documentation	106
8.13.2.1 disable()	106
8.13.2.2 enable()	106
8.14 iris::IrisGlobalInstance Class Reference	107
8.14.1 Member Function Documentation	107
8.14.1.1 getIrisInstance()	107
8.14.1.2 registerChannel()	107
8.14.1.3 registerIrisInterfaceChannel()	108
8.14.1.4 setLogMessageFunction()	108
8.14.1.5 unregisterIrisInterfaceChannel()	108
8.15 iris::IrisInstance Class Reference	108
8.15.1 Member Typedef Documentation	112
8.15.1.1 EventCallbackFunction	112
8.15.2 Constructor & Destructor Documentation	112
8.15.2.1 IrisInstance() [1/2]	112
8.15.2.2 IrisInstance() [2/2]	113
8.15.3 Member Function Documentation	113
8.15.3.1 addCallback_IRIS_INSTANCE_REGISTRY_CHANGED()	113
8.15.3.2 destroyAllEventStreams()	113
8.15.3.3 disableEvent()	113
8.15.3.4 enableEvent() [1/2]	114
8.15.3.5 enableEvent() [2/2]	114
8.15.3.6 eventBufferDestroyed()	115

8.15.3.7 findEventSources()	115
8.15.3.8 findEventSourcesAndFields()	115
8.15.3.9 findInstanceInfos()	116
8.15.3.10 getBuilder()	116
8.15.3.11 getInstanceId()	116
8.15.3.12 getInstanceInfo() [1/2]	116
8.15.3.13 getInstanceInfo() [2/2]	117
8.15.3.14 getInstanceList()	117
8.15.3.15 getInstanceName() [1/2]	117
8.15.3.16 getInstanceName() [2/2]	117
8.15.3.17 getInstId()	117
8.15.3.18 getLocalIrisInterface()	118
8.15.3.19 getMemorySpaceId()	118
8.15.3.20 getMemorySpaceInfo()	118
8.15.3.21 getPropertyMap()	118
8.15.3.22 getRemoteIrisInterface()	118
8.15.3.23 getResourceId()	118
8.15.3.24 irisCall()	119
8.15.3.25 irisCallNoThrow()	119
8.15.3.26 irisCallThrow()	119
8.15.3.27 isRegistered()	119
8.15.3.28 isValidEvBufId()	119
8.15.3.29 notifyStateChanged()	119
8.15.3.30 publishCppInterface()	119
8.15.3.31 registerEventBufferCallback() [1/3]	120
8.15.3.32 registerEventBufferCallback() [2/3]	120
8.15.3.33 registerEventBufferCallback() [3/3]	120
8.15.3.34 registerEventCallback() [1/3]	121
8.15.3.35 registerEventCallback() [2/3]	121
8.15.3.36 registerEventCallback() [3/3]	122
8.15.3.37 registerFunction()	122
8.15.3.38 registerInstance()	122
8.15.3.39 resourceRead()	123
8.15.3.40 resourceReadCrn()	123
8.15.3.41 resourceReadStr()	124
8.15.3.42 resourceWrite()	124
8.15.3.43 resourceWriteCrn()	124
8.15.3.44 resourceWriteStr()	124
8.15.3.45 sendRequest()	124
8.15.3.46 sendResponse()	124
8.15.3.47 setCallback_IRIS_SHUTDOWN_LEAVE()	125
8.15.3.48 setCallback_IRIS_SIMULATION_TIME_EVENT()	125

8.15.3.49 setConnectionInterface()	125
8.15.3.50 setPendingSyncStepResponse()	125
8.15.3.51 setProperty()	125
8.15.3.52 setSyncStepEventBufferId()	126
8.15.3.53 setThrowOnError()	126
8.15.3.54 simulationTimeDisableEvents()	126
8.15.3.55 simulationTimeIsRunning()	126
8.15.3.56 simulationTimeRun()	126
8.15.3.57 simulationTimeRunUntilStop()	126
8.15.3.58 simulationTimeStop()	127
8.15.3.59 simulationTimeWaitForStop()	127
8.15.3.60 unpublishCppInterface()	127
8.15.3.61 unregisterInstance()	127
8.16 iris::IrisInstanceBreakpoint Class Reference	127
8.16.1 Detailed Description	128
8.16.2 Member Function Documentation	128
8.16.2.1 addCondition()	128
8.16.2.2 attachTo()	129
8.16.2.3 getBreakpointInfo()	129
8.16.2.4 handleBreakpointHit()	129
8.16.2.5 notifyBreakpointHit()	129
8.16.2.6 notifyBreakpointHitData()	130
8.16.2.7 notifyBreakpointHitRegister()	130
8.16.2.8 setBreakpointDeleteDelegate()	131
8.16.2.9 setBreakpointSetDelegate()	131
8.16.2.10 setEventHandler()	131
8.16.2.11 setHandleBreakpointHitDelegate()	131
8.17 iris::IrisInstanceBuilder Class Reference	131
8.17.1 Detailed Description	138
8.17.2 Constructor & Destructor Documentation	138
8.17.2.1 IrisInstanceBuilder()	138
8.17.3 Member Function Documentation	138
8.17.3.1 addTable()	138
8.17.3.2 enableSemihostingAndGetManager()	139
8.17.3.3 setDbgStateDelegates()	139
8.17.3.4 setDbgStateGetAcknowledgeDelegate() [1/3]	139
8.17.3.5 setDbgStateGetAcknowledgeDelegate() [2/3]	140
8.17.3.6 setDbgStateGetAcknowledgeDelegate() [3/3]	140
8.17.3.7 setDbgStateSetRequestDelegate() [1/3]	140
8.17.3.8 setDbgStateSetRequestDelegate() [2/3]	141
8.17.3.9 setDbgStateSetRequestDelegate() [3/3]	141
8.17.3.10 setDefaultTableReadDelegate() [1/3]	142

8.17.3.11 setDefaultTableReadDelegate() [2/3]	142
8.17.3.12 setDefaultTableReadDelegate() [3/3]	142
8.17.3.13 setDefaultTableWriteDelegate() [1/3]	143
8.17.3.14 setDefaultTableWriteDelegate() [2/3]	143
8.17.3.15 setDefaultTableWriteDelegate() [3/3]	144
8.17.3.16 setExecutionStateGetDelegate() [1/3]	144
8.17.3.17 setExecutionStateGetDelegate() [2/3]	144
8.17.3.18 setExecutionStateGetDelegate() [3/3]	145
8.17.3.19 setExecutionStateSetDelegate() [1/3]	145
8.17.3.20 setExecutionStateSetDelegate() [2/3]	146
8.17.3.21 setExecutionStateSetDelegate() [3/3]	146
8.17.3.22 setCurrentDisassemblyModeDelegate()	146
8.18 iris::IrisInstanceCheckpoint Class Reference	147
8.18.1 Detailed Description	147
8.18.2 Member Function Documentation	147
8.18.2.1 attachTo()	147
8.18.2.2 setCheckpointRestoreDelegate()	147
8.18.2.3 setCheckpointSaveDelegate()	147
8.19 iris::IrisInstanceDebuggableState Class Reference	148
8.19.1 Detailed Description	148
8.19.2 Member Function Documentation	148
8.19.2.1 attachTo()	148
8.19.2.2 setGetAcknowledgeDelegate()	148
8.19.2.3 setSetRequestDelegate()	148
8.20 iris::IrisInstanceDisassembler Class Reference	149
8.20.1 Detailed Description	149
8.21 iris::IrisInstanceEvent Class Reference	149
8.21.1 Detailed Description	150
8.21.2 Constructor & Destructor Documentation	150
8.21.2.1 IrisInstanceEvent()	151
8.21.3 Member Function Documentation	151
8.21.3.1 addEventSource() [1/2]	151
8.21.3.2 addEventSource() [2/2]	151
8.21.3.3 attachTo()	151
8.21.3.4 deleteEventSource()	152
8.21.3.5 destroyAllEventStreams()	152
8.21.3.6 destroyEventStream()	152
8.21.3.7 enhanceEventSource()	152
8.21.3.8 eventBufferClear()	152
8.21.3.9 eventBufferGetSyncStepResponse()	153
8.21.3.10 getEventSourceInfo()	153
8.21.3.11 hasEventSource()	153

8.21.3.12 isValidEvBufId()	153
8.21.3.13 renameEventSource()	154
8.21.3.14 setDefaultEsCreateDelegate()	154
8.22 iris::IrisInstanceFactoryBuilder Class Reference	154
8.22.1 Detailed Description	155
8.22.2 Constructor & Destructor Documentation	155
8.22.2.1 IrisInstanceFactoryBuilder()	155
8.22.3 Member Function Documentation	155
8.22.3.1 addBoolParameter()	155
8.22.3.2 addHiddenBoolParameter()	156
8.22.3.3 addHiddenParameter()	156
8.22.3.4 addHiddenStringParameter()	156
8.22.3.5 addParameter()	157
8.22.3.6 addStringParameter()	157
8.22.3.7 getHiddenParameterInfo()	157
8.22.3.8 getParameterInfo()	157
8.23 iris::IrisInstanceImage Class Reference	158
8.23.1 Detailed Description	158
8.23.2 Constructor & Destructor Documentation	158
8.23.2.1 IrisInstanceImage()	158
8.23.3 Member Function Documentation	159
8.23.3.1 attachTo()	159
8.23.3.2 readFileData()	159
8.23.3.3 setLoadImageDataDelegate()	159
8.23.3.4 setLoadImageFileDelegate()	159
8.24 iris::IrisInstanceImage_Callback Class Reference	160
8.24.1 Detailed Description	160
8.24.2 Constructor & Destructor Documentation	160
8.24.2.1 IrisInstanceImage_Callback()	160
8.24.3 Member Function Documentation	160
8.24.3.1 attachTo()	160
8.24.3.2 openImage()	161
8.25 iris::IrisInstanceMemory Class Reference	161
8.25.1 Detailed Description	162
8.25.2 Constructor & Destructor Documentation	162
8.25.2.1 IrisInstanceMemory()	162
8.25.3 Member Function Documentation	162
8.25.3.1 addAddressTranslation()	162
8.25.3.2 addMemorySpace()	163
8.25.3.3 attachTo()	163
8.25.3.4 setDefaultGetSidebandInfoDelegate()	163
8.25.3.5 setDefaultReadDelegate()	163

8.25.3.6 setDefaultTranslateDelegate()	164
8.25.3.7 setDefaultWriteDelegate()	164
8.26 iris::IrisInstancePerInstanceExecution Class Reference	164
8.26.1 Detailed Description	164
8.26.2 Constructor & Destructor Documentation	164
8.26.2.1 IrisInstancePerInstanceExecution()	165
8.26.3 Member Function Documentation	166
8.26.3.1 attachTo()	166
8.26.3.2 setExecutionStateGetDelegate()	166
8.26.3.3 setExecutionStateSetDelegate()	166
8.27 iris::IrisInstanceResource Class Reference	166
8.27.1 Detailed Description	167
8.27.2 Constructor & Destructor Documentation	167
8.27.2.1 IrisInstanceResource()	167
8.27.3 Member Function Documentation	168
8.27.3.1 addResource()	168
8.27.3.2 attachTo()	168
8.27.3.3 beginResourceGroup()	168
8.27.3.4 calcHierarchicalNames()	169
8.27.3.5 getResourceInfo()	169
8.27.3.6 makeNamesHierarchical()	169
8.27.3.7 setNextSubRscId()	170
8.27.3.8 setTag()	170
8.28 iris::IrisInstanceSemihosting Class Reference	170
8.28.1 Member Function Documentation	171
8.28.1.1 attachTo()	171
8.28.1.2 readData()	171
8.28.1.3 semihostedCall()	171
8.28.1.4 setEventHandler()	173
8.29 iris::IrisInstanceSimulation Class Reference	173
8.29.1 Detailed Description	174
8.29.2 Constructor & Destructor Documentation	174
8.29.2.1 IrisInstanceSimulation()	175
8.29.3 Member Function Documentation	175
8.29.3.1 attachTo()	175
8.29.3.2 enterPostInstantiationPhase()	175
8.29.3.3 getSimulationPhaseDescription()	175
8.29.3.4 getSimulationPhaseName()	175
8.29.3.5 notifySimPhase()	175
8.29.3.6 registerSimEventsOnGlobalInstance()	176
8.29.3.7 setConnectionInterface()	176
8.29.3.8 setEventHandler()	176

8.29.3.9 setGetParameterInfoDelegate() [1/3]	176
8.29.3.10 setGetParameterInfoDelegate() [2/3]	177
8.29.3.11 setGetParameterInfoDelegate() [3/3]	177
8.29.3.12 setInstantiateDelegate() [1/3]	177
8.29.3.13 setInstantiateDelegate() [2/3]	177
8.29.3.14 setInstantiateDelegate() [3/3]	178
8.29.3.15 setLogLevel()	178
8.29.3.16 setRequestShutdownDelegate() [1/3]	178
8.29.3.17 setRequestShutdownDelegate() [2/3]	178
8.29.3.18 setRequestShutdownDelegate() [3/3]	179
8.29.3.19 setResetDelegate() [1/3]	179
8.29.3.20 setResetDelegate() [2/3]	179
8.29.3.21 setResetDelegate() [3/3]	179
8.29.3.22 setSetParameterValueDelegate() [1/3]	180
8.29.3.23 setSetParameterValueDelegate() [2/3]	180
8.29.3.24 setSetParameterValueDelegate() [3/3]	180
8.30 iris::IrisInstanceSimulationTime Class Reference	181
8.30.1 Detailed Description	181
8.30.2 Constructor & Destructor Documentation	181
8.30.2.1 IrisInstanceSimulationTime()	182
8.30.3 Member Function Documentation	182
8.30.3.1 attachTo()	182
8.30.3.2 registerSimTimeEventsOnGlobalInstance()	182
8.30.3.3 setEventHandler()	182
8.30.3.4 setSimTimeGetDelegate() [1/3]	182
8.30.3.5 setSimTimeGetDelegate() [2/3]	183
8.30.3.6 setSimTimeGetDelegate() [3/3]	183
8.30.3.7 setSimTimeNotifyStateChanged()	183
8.30.3.8 setSimTimeRunDelegate() [1/3]	184
8.30.3.9 setSimTimeRunDelegate() [2/3]	184
8.30.3.10 setSimTimeRunDelegate() [3/3]	184
8.30.3.11 setSimTimeStopDelegate() [1/3]	184
8.30.3.12 setSimTimeStopDelegate() [2/3]	185
8.30.3.13 setSimTimeStopDelegate() [3/3]	185
8.31 iris::IrisInstanceStep Class Reference	185
8.31.1 Detailed Description	186
8.31.2 Constructor & Destructor Documentation	186
8.31.2.1 IrisInstanceStep()	186
8.31.3 Member Function Documentation	186
8.31.3.1 attachTo()	186
8.31.3.2 setRemainingStepGetDelegate()	186
8.31.3.3 setRemainingStepSetDelegate()	186

8.31.3.4 setStepCountGetDelegate()	187
8.32 iris::IrisInstanceTable Class Reference	187
8.32.1 Detailed Description	187
8.32.2 Constructor & Destructor Documentation	187
8.32.2.1 IrisInstanceTable()	187
8.32.3 Member Function Documentation	188
8.32.3.1 addTableInfo()	188
8.32.3.2 attachTo()	188
8.32.3.3 setDefaultReadDelegate()	188
8.32.3.4 setDefaultWriteDelegate()	188
8.33 iris::IrisInstantiationContext Class Reference	189
8.33.1 Detailed Description	189
8.33.2 Member Function Documentation	189
8.33.2.1 error()	190
8.33.2.2 getBoolParameter()	190
8.33.2.3 getConnectionInterface()	190
8.33.2.4 getInstanceName()	190
8.33.2.5 getParameter() [1/3]	191
8.33.2.6 getParameter() [2/3]	191
8.33.2.7 getParameter() [3/3]	191
8.33.2.8 getRecommendedInstanceFlags()	191
8.33.2.9 getS64Parameter()	192
8.33.2.10 getStringParameter()	192
8.33.2.11 getSubcomponentContext()	192
8.33.2.12 getU64Parameter()	192
8.33.2.13 parameterError()	193
8.33.2.14 parameterWarning()	193
8.33.2.15 warning()	194
8.34 iris::IrisNonFactoryPlugin< PLUGIN_CLASS > Class Template Reference	194
8.34.1 Detailed Description	194
8.35 iris::IrisParameterBuilder Class Reference	194
8.35.1 Detailed Description	196
8.35.2 Constructor & Destructor Documentation	196
8.35.2.1 IrisParameterBuilder()	196
8.35.3 Member Function Documentation	196
8.35.3.1 addEnum()	196
8.35.3.2 addStringEnum()	197
8.35.3.3 setBitWidth()	197
8.35.3.4 setDefault() [1/3]	197
8.35.3.5 setDefault() [2/3]	198
8.35.3.6 setDefault() [3/3]	198
8.35.3.7 setDefaultFloat()	198

8.35.3.8	setDefaultSigned() [1/2]	199
8.35.3.9	setDefaultSigned() [2/2]	199
8.35.3.10	setDescr()	199
8.35.3.11	setFormat()	199
8.35.3.12	setHidden()	200
8.35.3.13	setInitOnly()	200
8.35.3.14	setMax() [1/2]	200
8.35.3.15	setMax() [2/2]	200
8.35.3.16	setMaxFloat()	202
8.35.3.17	setMaxSigned() [1/2]	202
8.35.3.18	setMaxSigned() [2/2]	202
8.35.3.19	setMin() [1/2]	203
8.35.3.20	setMin() [2/2]	203
8.35.3.21	setMinFloat()	203
8.35.3.22	setMinSigned() [1/2]	203
8.35.3.23	setMinSigned() [2/2]	204
8.35.3.24	setName()	204
8.35.3.25	setRange() [1/2]	204
8.35.3.26	setRange() [2/2]	205
8.35.3.27	setRangeFloat()	205
8.35.3.28	setRangeSigned() [1/2]	205
8.35.3.29	setRangeSigned() [2/2]	206
8.35.3.30	setRwMode()	206
8.35.3.31	setSubRsclD()	206
8.35.3.32	setTag() [1/2]	206
8.35.3.33	setTag() [2/2]	207
8.35.3.34	setTopology()	207
8.35.3.35	setType()	207
8.36	iris::IrisPluginFactory< PLUGIN_CLASS > Class Template Reference	208
8.37	iris::IrisPluginFactoryBuilder Class Reference	208
8.37.1	Detailed Description	208
8.37.2	Constructor & Destructor Documentation	208
8.37.2.1	IrisPluginFactoryBuilder()	208
8.37.3	Member Function Documentation	209
8.37.3.1	getDefaultInstanceName()	209
8.37.3.2	getInstanceNamePrefix()	209
8.37.3.3	getPluginName()	209
8.37.3.4	setDefaultInstanceName()	209
8.37.3.5	setInstanceNamePrefix()	209
8.37.3.6	setPluginName()	210
8.38	iris::IrisRegisterReadEventEmitter< REG_T, ARGS > Class Template Reference	210
8.38.1	Detailed Description	210

8.38.2 Member Function Documentation	211
8.38.2.1 operator()()	211
8.39 iris::IrisRegisterUpdateEventEmitter< REG_T, ARGS > Class Template Reference	211
8.39.1 Detailed Description	211
8.39.2 Member Function Documentation	212
8.39.2.1 operator()()	212
8.40 iris::IrisSimulationResetContext Class Reference	212
8.40.1 Detailed Description	213
8.40.2 Member Function Documentation	213
8.40.2.1 getAllowPartialReset()	213
8.41 iris::IrisInstanceBuilder::MemorySpaceBuilder Class Reference	213
8.41.1 Detailed Description	214
8.41.2 Member Function Documentation	214
8.41.2.1 addAttribute()	214
8.41.2.2 getSpaceId()	215
8.41.2.3 setAttributeDefault()	215
8.41.2.4 setAttributes()	215
8.41.2.5 setCanonicalMsn()	215
8.41.2.6 setDescription()	216
8.41.2.7 setEndianness()	216
8.41.2.8 setMaxAddr()	216
8.41.2.9 setMinAddr()	216
8.41.2.10 setName()	217
8.41.2.11 setReadDelegate() [1/3]	217
8.41.2.12 setReadDelegate() [2/3]	217
8.41.2.13 setReadDelegate() [3/3]	218
8.41.2.14 setSidebandDelegate() [1/3]	218
8.41.2.15 setSidebandDelegate() [2/3]	219
8.41.2.16 setSidebandDelegate() [3/3]	219
8.41.2.17 setSupportedByteWidths()	219
8.41.2.18 setWriteDelegate() [1/3]	220
8.41.2.19 setWriteDelegate() [2/3]	220
8.41.2.20 setWriteDelegate() [3/3]	221
8.42 iris::IrisCommandLineParser::Option Struct Reference	221
8.42.1 Detailed Description	221
8.42.2 Member Function Documentation	221
8.42.2.1 setList()	222
8.43 iris::IrisInstanceBuilder::ParameterBuilder Class Reference	222
8.43.1 Detailed Description	223
8.43.2 Member Function Documentation	223
8.43.2.1 addEnum()	224
8.43.2.2 addStringEnum()	224

8.43.2.3	getRscId() [1/2]	224
8.43.2.4	getRscId() [2/2]	224
8.43.2.5	setBitWidth()	225
8.43.2.6	setName()	225
8.43.2.7	setDefaultData() [1/2]	225
8.43.2.8	setDefaultData() [2/2]	225
8.43.2.9	setDefaultDataFromContainer()	226
8.43.2.10	setDefaultString()	226
8.43.2.11	setDescription()	226
8.43.2.12	setFormat()	227
8.43.2.13	setHidden()	227
8.43.2.14	setInitOnly()	227
8.43.2.15	setMax() [1/2]	228
8.43.2.16	setMax() [2/2]	228
8.43.2.17	setMaxFromContainer()	228
8.43.2.18	setMin() [1/2]	229
8.43.2.19	setMin() [2/2]	229
8.43.2.20	setMinFromContainer()	229
8.43.2.21	setName()	230
8.43.2.22	setParentRscId()	230
8.43.2.23	setReadDelegate() [1/3]	230
8.43.2.24	setReadDelegate() [2/3]	230
8.43.2.25	setReadDelegate() [3/3]	231
8.43.2.26	setRwMode()	231
8.43.2.27	setSubRscId()	232
8.43.2.28	setTag() [1/2]	232
8.43.2.29	setTag() [2/2]	232
8.43.2.30	setType()	232
8.43.2.31	setWriteDelegate() [1/3]	233
8.43.2.32	setWriteDelegate() [2/3]	233
8.43.2.33	setWriteDelegate() [3/3]	233
8.44	iris::IrisInstanceEvent::ProxyEventInfo Struct Reference	234
8.44.1	Detailed Description	234
8.45	iris::IrisInstanceBuilder::RegisterBuilder Class Reference	234
8.45.1	Detailed Description	236
8.45.2	Member Function Documentation	236
8.45.2.1	addEnum()	236
8.45.2.2	addField()	237
8.45.2.3	addLogicalField()	237
8.45.2.4	addStringEnum()	237
8.45.2.5	getRscId() [1/2]	238
8.45.2.6	getRscId() [2/2]	238

8.45.2.7 setAddressOffset()	238
8.45.2.8 setBitWidth()	238
8.45.2.9 setBreakpointSupportInfo()	239
8.45.2.10 setCanonicalRn()	239
8.45.2.11 setCanonicalRnElfDwarf()	239
8.45.2.12 setCname()	239
8.45.2.13 setDescription()	240
8.45.2.14 setFormat()	240
8.45.2.15 setLsbOffset()	240
8.45.2.16 setName()	241
8.45.2.17 setParentRscld()	241
8.45.2.18 setReadDelegate() [1/3]	241
8.45.2.19 setReadDelegate() [2/3]	241
8.45.2.20 setReadDelegate() [3/3]	242
8.45.2.21 setResetData() [1/2]	242
8.45.2.22 setResetData() [2/2]	243
8.45.2.23 setResetDataFromContainer()	243
8.45.2.24 setResetString()	243
8.45.2.25 setRwMode()	244
8.45.2.26 setSubRscld()	244
8.45.2.27 setTag() [1/2]	244
8.45.2.28 setTag() [2/2]	245
8.45.2.29 setType()	245
8.45.2.30 setWriteDelegate() [1/3]	245
8.45.2.31 setWriteDelegate() [2/3]	245
8.45.2.32 setWriteDelegate() [3/3]	246
8.45.2.33 setWriteMask() [1/2]	246
8.45.2.34 setWriteMask() [2/2]	247
8.45.2.35 setWriteMaskFromContainer()	247
8.46 iris::IrisInstanceResource::ResourceInfoAndAccess Struct Reference	247
8.46.1 Detailed Description	248
8.47 iris::ResourceWriteValue Struct Reference	248
8.47.1 Detailed Description	248
8.48 iris::IrisInstanceBuilder::SemihostingManager Class Reference	248
8.48.1 Detailed Description	248
8.48.2 Member Function Documentation	249
8.48.2.1 readData()	249
8.48.2.2 semihostedCall()	249
8.49 iris::IrisInstanceMemory::SpaceInfoAndAccess Struct Reference	249
8.49.1 Detailed Description	250
8.50 iris::IrisInstanceBuilder::TableBuilder Class Reference	250
8.50.1 Detailed Description	251

8.50.2 Member Function Documentation	251
8.50.2.1 addColumn()	251
8.50.2.2 addColumnInfo()	251
8.50.2.3 setDescription()	251
8.50.2.4 setFormatLong()	252
8.50.2.5 setFormatShort()	252
8.50.2.6 setIndexFormatHint()	252
8.50.2.7 setMaxIndex()	252
8.50.2.8 setMinIndex()	253
8.50.2.9 setName()	253
8.50.2.10 setReadDelegate() [1/3]	253
8.50.2.11 setReadDelegate() [2/3]	254
8.50.2.12 setReadDelegate() [3/3]	254
8.50.2.13 setWriteDelegate() [1/3]	254
8.50.2.14 setWriteDelegate() [2/3]	255
8.50.2.15 setWriteDelegate() [3/3]	255
8.51 iris::IrisInstanceBuilder::TableColumnBuilder Class Reference	256
8.51.1 Detailed Description	256
8.51.2 Member Function Documentation	256
8.51.2.1 addColumn()	257
8.51.2.2 addColumnInfo()	257
8.51.2.3 endColumn()	257
8.51.2.4 setBitWidth()	258
8.51.2.5 setDescription()	259
8.51.2.6 setFormat()	259
8.51.2.7 setFormatLong()	259
8.51.2.8 setFormatShort()	260
8.51.2.9 setName()	260
8.51.2.10 setRwMode()	260
8.51.2.11 setType()	260
8.52 iris::IrisInstanceTable::TableInfoAndAccess Struct Reference	261
8.52.1 Detailed Description	261
9 File Documentation	263
9.1 IrisCanonicalMsnArm.h File Reference	263
9.1.1 Detailed Description	263
9.2 IrisCanonicalMsnArm.h	263
9.3 IrisCConnection.h File Reference	264
9.3.1 Detailed Description	264
9.4 IrisCConnection.h	264
9.5 IrisClient.h File Reference	266
9.5.1 Detailed Description	267

9.6 IrisClient.h	267
9.7 IrisCommandLineParser.h File Reference	285
9.7.1 Detailed Description	285
9.8 IrisCommandLineParser.h	286
9.9 IrisElfDwarfArm.h File Reference	288
9.9.1 Detailed Description	289
9.10 IrisElfDwarfArm.h	289
9.11 IrisEventEmitter.h File Reference	291
9.11.1 Detailed Description	291
9.12 IrisEventEmitter.h	291
9.13 IrisGlobalInstance.h File Reference	292
9.13.1 Detailed Description	292
9.14 IrisGlobalInstance.h	292
9.15 IrisInstance.h File Reference	296
9.15.1 Detailed Description	296
9.15.2 Typedef Documentation	297
9.15.2.1 EventCallbackDelegate	297
9.16 IrisInstance.h	297
9.17 IrisInstanceBreakpoint.h File Reference	305
9.17.1 Detailed Description	305
9.17.2 Typedef Documentation	305
9.17.2.1 BreakpointDeleteDelegate	305
9.17.2.2 BreakpointSetDelegate	305
9.17.2.3 HandleBreakpointHitDelegate	306
9.18 IrisInstanceBreakpoint.h	306
9.19 IrisInstanceBuilder.h File Reference	307
9.19.1 Detailed Description	308
9.20 IrisInstanceBuilder.h	308
9.21 IrisInstanceCheckpoint.h File Reference	334
9.21.1 Detailed Description	334
9.21.2 Typedef Documentation	334
9.21.2.1 CheckpointRestoreDelegate	334
9.21.2.2 CheckpointSaveDelegate	335
9.22 IrisInstanceCheckpoint.h	335
9.23 IrisInstanceDebuggableState.h File Reference	335
9.23.1 Detailed Description	336
9.23.2 Typedef Documentation	336
9.23.2.1 DebuggableStateGetAcknowledgeDelegate	336
9.23.2.2 DebuggableStateSetRequestDelegate	336
9.24 IrisInstanceDebuggableState.h	336
9.25 IrisInstanceDisassembler.h File Reference	337
9.25.1 Detailed Description	337

9.26 IrisInstanceDisassembler.h	337
9.27 IrisInstanceEvent.h File Reference	338
9.27.1 Detailed Description	339
9.27.2 Typedef Documentation	339
9.27.2.1 EventStreamCreateDelegate	339
9.28 IrisInstanceEvent.h	339
9.29 IrisInstanceFactoryBuilder.h File Reference	347
9.29.1 Detailed Description	348
9.30 IrisInstanceFactoryBuilder.h	348
9.31 IrisInstanceImage.h File Reference	349
9.31.1 Detailed Description	350
9.31.2 Typedef Documentation	350
9.31.2.1 ImageLoadDataDelegate	350
9.31.2.2 ImageLoadFileDelegate	350
9.32 IrisInstanceImage.h	351
9.33 IrisInstanceMemory.h File Reference	352
9.33.1 Detailed Description	352
9.33.2 Typedef Documentation	353
9.33.2.1 MemoryAddressTranslateDelegate	353
9.33.2.2 MemoryGetSidebandInfoDelegate	353
9.33.2.3 MemoryReadDelegate	353
9.33.2.4 MemoryWriteDelegate	353
9.34 IrisInstanceMemory.h	354
9.35 IrisInstancePerInstanceExecution.h File Reference	355
9.35.1 Detailed Description	356
9.35.2 Typedef Documentation	356
9.35.2.1 PerInstanceExecutionStateGetDelegate	356
9.35.2.2 PerInstanceExecutionStateSetDelegate	356
9.36 IrisInstancePerInstanceExecution.h	356
9.37 IrisInstanceResource.h File Reference	357
9.37.1 Detailed Description	357
9.37.2 Typedef Documentation	357
9.37.2.1 ResourceReadDelegate	358
9.37.2.2 ResourceWriteDelegate	358
9.37.3 Function Documentation	358
9.37.3.1 resourceReadBitField()	358
9.37.3.2 resourceWriteBitField()	358
9.38 IrisInstanceResource.h	359
9.39 IrisInstanceSemihosting.h File Reference	360
9.39.1 Detailed Description	360
9.40 IrisInstanceSemihosting.h	360
9.41 IrisInstanceSimulation.h File Reference	362

9.41.1 Detailed Description	363
9.41.2 Typedef Documentation	363
9.41.2.1 SimulationGetParameterInfoDelegate	363
9.41.2.2 SimulationInstantiateDelegate	363
9.41.2.3 SimulationRequestShutdownDelegate	363
9.41.2.4 SimulationResetDelegate	363
9.41.2.5 SimulationSetParameterValueDelegate	364
9.42 IrisInstanceSimulation.h	364
9.43 IrisInstanceSimulationTime.h File Reference	367
9.43.1 Detailed Description	367
9.43.2 Typedef Documentation	368
9.43.2.1 SimulationTimeGetDelegate	368
9.43.2.2 SimulationTimeRunDelegate	368
9.43.2.3 SimulationTimeStopDelegate	368
9.43.3 Enumeration Type Documentation	368
9.43.3.1 TIME_EVENT_REASON	368
9.44 IrisInstanceSimulationTime.h	368
9.45 IrisInstanceStep.h File Reference	370
9.45.1 Detailed Description	370
9.45.2 Typedef Documentation	371
9.45.2.1 RemainingStepGetDelegate	371
9.45.2.2 RemainingStepSetDelegate	371
9.45.2.3 StepCountGetDelegate	371
9.46 IrisInstanceStep.h	371
9.47 IrisInstanceTable.h File Reference	372
9.47.1 Detailed Description	372
9.47.2 Typedef Documentation	372
9.47.2.1 TableReadDelegate	372
9.47.2.2 TableWriteDelegate	373
9.48 IrisInstanceTable.h	373
9.49 IrisInstantiationContext.h File Reference	373
9.49.1 Detailed Description	374
9.50 IrisInstantiationContext.h	374
9.51 IrisParameterBuilder.h File Reference	375
9.51.1 Detailed Description	376
9.52 IrisParameterBuilder.h	376
9.53 IrisPluginFactory.h File Reference	379
9.53.1 Detailed Description	380
9.53.2 Macro Definition Documentation	380
9.53.2.1 IRIS_NON_FACTORY_PLUGIN	380
9.53.2.2 IRIS_PLUGIN_FACTORY	380
9.54 IrisPluginFactory.h	381

9.55 IrisRegisterEventEmitter.h File Reference	385
9.55.1 Detailed Description	385
9.56 IrisRegisterEventEmitter.h	385
9.57 IrisTcpClient.h File Reference	386
9.57.1 Detailed Description	386
9.58 IrisTcpClient.h	386

Chapter 1

IrisSupportLib Reference Guide

Copyright © 2018-2023 Arm Limited or its affiliates. All rights reserved.

About this book

This book contains API reference documentation for IrisSupportLib. It was generated from the source code using Doxygen.

The IrisSupportLib library contains the code to create an IrisInstance object and helper classes to add functionality to the instance. It also contains the code to communicate with the Iris system using U64JSON and general support code used by the library, for example thread abstraction.

IrisSupportLib is built as a static library. It must be linked in to any executable or DSO that needs to connect to Iris. The library is provided pre-compiled in \$IRIS_HOME/<OS_Compiler>/libIrisSupport.a|IrisSupport.lib. Headers are provided in the directory \$IRIS_HOME/include/iris/ and the source code is provided in the directory \$IRIS_HOME/↔ IrisSupportLib/.

Other information

For more information about Iris, see the [Iris User Guide](#).
See the following locations for examples of Iris clients and plug-ins:

- \$IRIS_HOME/Examples/Client/ for Iris C++ client examples.
- \$IRIS_HOME/Python/Examples/ for Iris Python client examples.
- \$IRIS_HOME/Examples/Plugin/ for Iris plug-in examples.

Feedback

Feedback on this product If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content If you have any comments on content, send an e-mail to errata@arm.com. Give:

- The title *IrisSupportLib Reference Guide*.
- The number 101319_0100_18_en.
- If applicable, the relevant page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm.

No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with © or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2018-2023 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

<http://www.arm.com>.

Release Information

Document History			
Issue	Date	Confidentiality	Change

Document History			
0100-00	23 Nov 2018	Non-Confidential	New document for Fast Models v11.5.
0100-01	26 Feb 2019	Non-Confidential	Update for v11.6.
0100-02	17 May 2019	Non-Confidential	Update for v11.7.
0100-03	05 Sep 2019	Non-Confidential	Update for v11.8.
0100-04	28 Nov 2019	Non-Confidential	Update for v11.9.
0100-05	12 Mar 2020	Non-Confidential	Update for v11.10.
0100-06	22 Sep 2020	Non-Confidential	Update for v11.12.
0100-07	09 Dec 2020	Non-Confidential	Update for v11.13.
0100-08	17 Mar 2021	Non-Confidential	Update for v11.14.
0100-09	29 Jun 2021	Non-Confidential	Update for v11.15.
0100-10	06 Oct 2021	Non-Confidential	Update for v11.16.
0100-11	16 Feb 2022	Non-Confidential	Update for v11.17.
0100-12	15 Jun 2022	Non-Confidential	Update for v11.18.
0100-13	14 Sept 2022	Non-Confidential	Update for v11.19.
0100-14	07 Dec 2022	Non-Confidential	Update for v11.20.
0100-15	22 Mar 2023	Non-Confidential	Update for v11.21.
0100-16	14 Jun 2023	Non-Confidential	Update for v11.22.
0100-17	13 Sep 2023	Non-Confidential	Update for v11.23.
0100-18	06 Dec 2023	Non-Confidential	Update for v11.23.

Chapter 2

IrisSupportLib NAMESPACE macros

To allow multiple different versions of IrisSupportLib to be used by different components in the same executable, all IrisSupportLib code is defined in a hidden inner namespace. This namespace is constructed from the revision and fork from `iris/detail/IrisSupportLibRevision.h`. For example, if revision=0 and fork=master, this means IrisSupportLib code is in the namespace `iris::r0master`.

This is then imported into the namespace `iris` so all Iris code can be used without the hidden internal namespace.

Make sure you include the Iris `NAMESPACE_` macros in any new source files, for example:

```
...
#ifndef ARM_INCLUDE_MyHeader_h
#define ARM_INCLUDE_MyHeader_h

#include "iris/detail/IrisCommon.h"

NAMESPACE_IRIS_START

// Code goes here

NAMESPACE_IRIS_END

#endif // ARM_INCLUDE_MyHeader_h
```


Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Instance Flags	15
IrisInstanceBuilder resource APIs	15
IrisInstanceBuilder event APIs	24
IrisInstanceBuilder breakpoint APIs	32
IrisInstanceBuilder memory APIs	37
IrisInstanceBuilder image loading APIs	45
IrisInstanceBuilder image readData callback APIs.	48
IrisInstanceBuilder execution stepping APIs	49
Disassembler delegate functions	53
Semihosting data request flag constants	55

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

iris::IrisInstanceBuilder::AddressTranslationBuilder	57
iris::IrisInstanceMemory::AddressTranslationInfoAndAccess	59
iris::BreakpointHitInfo	59
iris::IrisInstanceBuilder::EventSourceBuilder	59
iris::IrisInstanceEvent::EventSourceInfoAndDelegate	65
iris::EventStream	65
iris::IrisEventStream	106
iris::IrisInstanceBuilder::FieldBuilder	77
iris::IrisCommandLineParser	97
IrisConnectionInterface	
iris::IrisCConnection	91
iris::IrisClient	92
iris::IrisGlobalInstance	107
IrisEventEmitterBase	
iris::IrisEventEmitter< ARGS >	102
iris::IrisEventRegistry	103
iris::IrisInstance	108
iris::IrisInstanceBreakpoint	127
iris::IrisInstanceBuilder	131
iris::IrisInstanceCheckpoint	147
iris::IrisInstanceDebuggableState	148
iris::IrisInstanceDisassembler	149
iris::IrisInstanceEvent	149
iris::IrisInstanceFactoryBuilder	154
iris::IrisPluginFactoryBuilder	208
iris::IrisInstanceImage	158
iris::IrisInstanceImage_Callback	160
iris::IrisInstanceMemory	161
iris::IrisInstancePerInstanceExecution	164
iris::IrisInstanceResource	166
iris::IrisInstanceSemihosting	170
iris::IrisInstanceSimulation	173
iris::IrisInstanceSimulationTime	181
iris::IrisInstanceStep	185
iris::IrisInstanceTable	187
iris::IrisInstantiationContext	189
IrisInterface	
iris::IrisClient	92
iris::IrisGlobalInstance	107
iris::IrisNonFactoryPlugin< PLUGIN_CLASS >	194

iris::IrisParameterBuilder	194
iris::IrisPluginFactory< PLUGIN_CLASS >	208
impl::IrisProcessEventsInterface	
iris::IrisClient	92
IrisRegisterEventEmitterBase	
iris::IrisRegisterReadEventEmitter< REG_T, ARGS >	210
iris::IrisRegisterUpdateEventEmitter< REG_T, ARGS >	211
iris::IrisSimulationResetContext	212
iris::IrisInstanceBuilder::MemorySpaceBuilder	213
iris::IrisCommandLineParser::Option	221
iris::IrisInstanceBuilder::ParameterBuilder	222
iris::IrisInstanceEvent::ProxyEventInfo	234
iris::IrisInstanceBuilder::RegisterBuilder	234
iris::IrisInstanceResource::ResourceInfoAndAccess	247
iris::ResourceWriteValue	248
iris::IrisInstanceBuilder::SemihostingManager	248
iris::IrisInstanceMemory::SpaceInfoAndAccess	249
iris::IrisInstanceBuilder::TableBuilder	250
iris::IrisInstanceBuilder::TableColumnBuilder	256
iris::IrisInstanceTable::TableInfoAndAccess	261

Chapter 5

Class Index

5.1 Class List

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

iris::IrisInstanceBuilder::AddressTranslationBuilder	57
Used to set metadata for an address translation	
iris::IrisInstanceMemory::AddressTranslationInfoAndAccess	59
Contains static address translation information	
iris::BreakpointHitInfo	59
iris::IrisInstanceBuilder::EventSourceBuilder	
Used to set metadata on an EventSource	59
iris::IrisInstanceEvent::EventSourceInfoAndDelegate	
Contains the metadata and delegates for a single EventSource	65
iris::EventStream	
Base class for event streams	65
iris::IrisInstanceBuilder::FieldBuilder	
Used to set metadata on a register field resource	77
iris::IrisCConnection	
Provide an IrisConnectionInterface which loads an IrisC library	91
iris::IrisClient	92
iris::IrisCommandLineParser	97
iris::IrisEventEmitter< ARGS >	
A helper class for generating Iris events	102
iris::IrisEventRegistry	
Class to register Iris event streams for an event	103
iris::IrisEventStream	
Event stream class for Iris-specific events	106
iris::IrisGlobalInstance	107
iris::IrisInstance	108
iris::IrisInstanceBreakpoint	
Breakpoint add-on for IrisInstance	127
iris::IrisInstanceBuilder	
Builder interface to populate an IrisInstance with registers, memory etc	131
iris::IrisInstanceCheckpoint	
Checkpoint add-on for IrisInstance	147
iris::IrisInstanceDebuggableState	
Debuggable-state add-on for IrisInstance	148
iris::IrisInstanceDisassembler	
Disassembler add-on for IrisInstance	149
iris::IrisInstanceEvent	
Event add-on for IrisInstance	149
iris::IrisInstanceFactoryBuilder	
A builder class to construct instantiation parameter metadata	154

iris::IrisInstanceImage	
Image loading add-on for IrisInstance	158
iris::IrisInstanceImage_Callback	
Image loading add-on for IrisInstance clients implementing <code>image_loadDataRead()</code>	160
iris::IrisInstanceMemory	
Memory add-on for IrisInstance	161
iris::IrisInstancePerInstanceExecution	
Per-instance execution control add-on for IrisInstance	164
iris::IrisInstanceResource	
Resource add-on for IrisInstance	166
iris::IrisInstanceSemihosting	170
iris::IrisInstanceSimulation	
An IrisInstance add-on that adds simulation functions for the SimulationEngine instance	173
iris::IrisInstanceSimulationTime	
Simulation time add-on for IrisInstance	181
iris::IrisInstanceStep	
Step add-on for IrisInstance	185
iris::IrisInstanceTable	
Table add-on for IrisInstance	187
iris::IrisInstantiationContext	
Provides context when instantiating an Iris instance from a factory	189
iris::IrisNonFactoryPlugin< PLUGIN_CLASS >	
Wrapper to instantiate a non-factory plugin	194
iris::IrisParameterBuilder	
Helper class to construct instantiation parameters	194
iris::IrisPluginFactory< PLUGIN_CLASS >	208
iris::IrisPluginBuilderFactory	
Set meta data for instantiating a plug-in instance	208
iris::IrisRegisterReadEventEmitter< REG_T, ARGS >	
An EventEmitter class for register read events	210
iris::IrisRegisterUpdateEventEmitter< REG_T, ARGS >	
An EventEmitter class for register update events	211
iris::IrisSimulationResetContext	
Provides context to a reset delegate call	212
iris::IrisInstanceBuilder::MemorySpaceBuilder	
Used to set metadata for a memory space	213
iris::IrisCommandLineParser::Option	
Option container	221
iris::IrisInstanceBuilder::ParameterBuilder	
Used to set metadata on a parameter	222
iris::IrisInstanceEvent::ProxyEventInfo	
Contains information for a single proxy EventSource	234
iris::IrisInstanceBuilder::RegisterBuilder	
Used to set metadata on a register resource	234
iris::IrisInstanceResource::ResourceInfoAndAccess	
Entry in 'resourceInfos'	247
iris::ResourceWriteValue	248
iris::IrisInstanceBuilder::SemihostingManager	
Semihosting_apis IrisInstanceBuilder semihosting APIs	248
iris::IrisInstanceMemory::SpaceInfoAndAccess	
Entry in 'spaceInfos'	249
iris::IrisInstanceBuilder::TableBuilder	
Used to set metadata for a table	250
iris::IrisInstanceBuilder::TableColumnBuilder	
Used to set metadata for a table column	256
iris::IrisInstanceTable::TableInfoAndAccess	
Entry in 'tableInfos'	261

Chapter 6

File Index

6.1 File List

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

IrisCanonicalMsnArm.h	Constants for the memory.canonicalMsnScheme arm.com/memoryspaces	263
IrisCConnection.h	IrisConnectionInterface implementation based on IrisC	264
IrisClient.h	Iris client which supports multiple methods to connect to other Iris executables	266
IrisCommandLineParser.h	Generic command line parser	285
IrisElfDwarfArm.h	Constants for the register.canonicalRnScheme "ElfDwarf" for architecture Arm	288
IrisEventEmitter.h	A utility class for emitting Iris events	291
IrisGlobalInstance.h	Central instance which lives in the simulation engine and distributes all Iris messages	292
IrisInstance.h	Boilerplate code for an Iris instance, including clients and components	296
IrisInstanceBreakpoint.h	Breakpoint add-on to IrisInstance	305
IrisInstanceBuilder.h	A high level interface to build up functionality on an IrisInstance	307
IrisInstanceCheckpoint.h	Checkpoint add-on to IrisInstance	334
IrisInstanceDebuggableState.h	IrisInstance add-on to implement debuggableState functions	335
IrisInstanceDisassembler.h	Disassembler add-on to IrisInstance	337
IrisInstanceEvent.h	Event add-on to IrisInstance	338
IrisInstanceFactoryBuilder.h	A helper class to build instantiation parameter metadata	347
IrisInstanceImage.h	Image-loading add-on to IrisInstance and image-loading callback add-on to the caller	349
IrisInstanceMemory.h	Memory add-on to IrisInstance	352
IrisInstancePerInstanceExecution.h	Per-instance execution control add-on to IrisInstance	355
IrisInstanceResource.h	Resource add-on to IrisInstance	357
IrisInstanceSemihosting.h	IrisInstance add-on to implement semihosting functionality	360

IrisInstanceSimulation.h	
IrisInstance add-on to implement simulation_* functions	362
IrisInstanceSimulationTime.h	
IrisInstance add-on to implement simulationTime functions	367
IrisInstanceStep.h	
Stepping-related add-on to an IrisInstance	370
IrisInstanceTable.h	
Table add-on to IrisInstance	372
IrisInstantiationContext.h	
Helper class used to instantiate Iris instances from generic factories	373
IrisParameterBuilder.h	
Helper class to construct instantiation parameters	375
IrisPluginFactory.h	
A generic plug-in factory for instantiating plug-in instances	379
IrisRegisterEventEmitter.h	
Utility classes for emitting register read and register update events	385
IrisTcpClient.h	
IrisTcpClient Type alias for IrisClient	386

Chapter 7

Module Documentation

7.1 Instance Flags

Flags that can be set when registering an [IrisInstance](#).

Variables

- static const uint64_t [iris::IrisInstance::DEFAULT_FLAGS](#) = [THROW_ON_ERROR](#)
Default flags used if not otherwise specified.
- static const bool [iris::IrisInstance::SYNCHRONOUS](#) = true
Cause [enableEvent\(\)](#) callback to be called back synchronously (i.e. the caller is blocked until the callback function returns).
- static const uint64_t [iris::IrisInstance::THROW_ON_ERROR](#) = (1 << 1)
Throw an exception when an Iris call returns an error response.
- static const uint64_t [iris::IrisInstance::UNQUIFY](#) = (1 << 0)
Uniquify instance name when registering.

7.1.1 Detailed Description

Flags that can be set when registering an [IrisInstance](#).

7.2 IrisInstanceBuilder resource APIs

Set up resource and register metadata and delegates.

Classes

- class [iris::IrisInstanceBuilder::FieldBuilder](#)
Used to set metadata on a register field resource.
- class [iris::IrisInstanceBuilder::ParameterBuilder](#)
Used to set metadata on a parameter.
- class [iris::IrisInstanceBuilder::RegisterBuilder](#)
Used to set metadata on a register resource.

Functions

- [RegisterBuilder](#) [iris::IrisInstanceBuilder::addNoValueRegister](#) (const std::string &name, const std::string &description, const std::string &format)
Add metadata for one noValue resource.
- [ParameterBuilder](#) [iris::IrisInstanceBuilder::addParameter](#) (const std::string &name, uint64_t bitWidth, const std::string &description)

- Add numeric parameter.*

 - [RegisterBuilder iris::IrisInstanceBuilder::addRegister](#) (const std::string &name, uint64_t bitWidth, const std::string &description, uint64_t addressOffset=IRIS_UINT64_MAX, uint64_t canonicalRn=IRIS_UINT64_MAX)

Add metadata for one numeric register resource.

 - [ParameterBuilder iris::IrisInstanceBuilder::addStringParameter](#) (const std::string &name, const std::string &description)

Add string parameter.

 - [RegisterBuilder iris::IrisInstanceBuilder::addStringRegister](#) (const std::string &name, const std::string &description)

Add metadata for one string register resource.

 - void [iris::IrisInstanceBuilder::beginResourceGroup](#) (const std::string &name, const std::string &description, uint64_t subRscldStart=IRIS_UINT64_MAX, const std::string &cname=std::string())

Begin a new resource group.

 - [ParameterBuilder iris::IrisInstanceBuilder::enhanceParameter](#) (ResourceId rscld)

Get [ParameterBuilder](#) to enhance a parameter.

 - [RegisterBuilder iris::IrisInstanceBuilder::enhanceRegister](#) (ResourceId rscld)

Get [RegisterBuilder](#) to enhance register.

 - const ResourceInfo & [iris::IrisInstanceBuilder::getResourceInfo](#) (ResourceId rscld)

Get ResourceInfo of a previously added register.

 - template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) READER, IrisErrorCode(T::*)(const ResourceInfo &, const [ResourceWriteValue](#) &) WRITER>
void [iris::IrisInstanceBuilder::setDefaultResourceDelegates](#) (T *instance)

Set both read and write resource delegates if they are defined in the same class.

 - template<IrisErrorCode(*)(const ResourceInfo &, ResourceReadResult &) FUNC>
void [iris::IrisInstanceBuilder::setDefaultResourceReadDelegate](#) ()

Set default read access function for all subsequently added resources.

 - void [iris::IrisInstanceBuilder::setDefaultResourceReadDelegate](#) ([ResourceReadDelegate](#) delegate=[ResourceReadDelegate](#)())

Set default read access function for all subsequently added resources.

 - template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>
void [iris::IrisInstanceBuilder::setDefaultResourceReadDelegate](#) (T *instance)

Set default read access function for all subsequently added resources.

 - template<IrisErrorCode(*)(const ResourceInfo &, const [ResourceWriteValue](#) &) FUNC>
void [iris::IrisInstanceBuilder::setDefaultResourceWriteDelegate](#) ()

Set default write access function for all subsequently added resources.

 - void [iris::IrisInstanceBuilder::setDefaultResourceWriteDelegate](#) ([ResourceWriteDelegate](#) delegate=[ResourceWriteDelegate](#)())

Set default write access function for all subsequently added resources.

 - template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const [ResourceWriteValue](#) &) METHOD>
void [iris::IrisInstanceBuilder::setDefaultResourceWriteDelegate](#) (T *instance)

Set default write access function for all subsequently added resources.

 - void [iris::IrisInstanceBuilder::setNextSubRscld](#) (uint64_t nextSubRscld)

Set the rscld that will be used for the next resource to be added.

 - void [iris::IrisInstanceBuilder::setPropertyCanonicalRnScheme](#) (const std::string &canonicalRnScheme)

Set the register.canonicalRnScheme instance property.

 - void [iris::IrisInstanceBuilder::setTag](#) (ResourceId rscld, const std::string &tag)

Set a tag for a specific resource.

7.2.1 Detailed Description

Set up resource and register metadata and delegates.

7.2.2 Function Documentation

7.2.2.1 addNoValueRegister()

```
RegisterBuilder iris::IrisInstanceBuilder::addNoValueRegister (
    const std::string & name,
    const std::string & description,
    const std::string & format )
```

Add metadata for one noValue resource.

Resource group: [beginResourceGroup\(\)](#) must have been called before calling this function. The added resource is automatically added to the last group added by [beginResourceGroup\(\)](#).

Type: The added resource is of type 'noValue'. Use [addRegister\(\)](#) to add a register of type 'numeric' or 'numericFp'. Use [addStringRegister\(\)](#) to add a register of type 'string'.

The returned builder object is only valid until another resource is added. It is only intended to modify the resource that was added last.

Parameters

<i>name</i>	Name of the resource. This is the same as the 'name' field of ResourceInfo.
<i>description</i>	Human readable description of the resource. This is the same as the 'description' field of ResourceInfo.
<i>format</i>	The format used to display this resource.

Returns

A [RegisterBuilder](#) object that can be used to set additional metadata for this resource.

7.2.2.2 addParameter()

```
ParameterBuilder iris::IrisInstanceBuilder::addParameter (
    const std::string & name,
    uint64_t bitWidth,
    const std::string & description )
```

Add numeric parameter.

Resource group: [beginResourceGroup\(\)](#) must have been called before calling this function. The added parameter is automatically added to the last group added by [beginResourceGroup\(\)](#).

Type: The added parameter is of type 'numeric'. Call setType("numericFp") on the returned [ParameterBuilder](#) to add a 'numericFp' (pure floating point) parameter. Use [addStringParameter\(\)](#) to add a parameter of type 'string'.

The returned builder object is only valid until another resource is added. It is only intended to modify the resource that was added last.

Parameters

<i>name</i>	Name of the parameter. This is the same as the 'name' field of ResourceInfo.
<i>bitWidth</i>	Width of the parameter in bits. This is the same as the 'bitWidth' field of ResourceInfo.
<i>description</i>	Human readable description of the parameter. This is the same as the 'description' field of ResourceInfo.

Returns

A [ParameterBuilder](#) object that can be used to set additional metadata for this parameter.

7.2.2.3 addRegister()

```
RegisterBuilder iris::IrisInstanceBuilder::addRegister (
    const std::string & name,
    uint64_t bitWidth,
```

```
const std::string & description,
uint64_t addressOffset = IRIS_UINT64_MAX,
uint64_t canonicalRn = IRIS_UINT64_MAX )
```

Add metadata for one numeric register resource.

Resource group: [beginResourceGroup\(\)](#) must have been called before calling this function. The added resource is automatically added to the last group added by [beginResourceGroup\(\)](#).

Type: The added resource is of type 'numeric'. Call `setType("numericFp")` on the returned [RegisterBuilder](#) to add a 'numericFp' (pure floating-point) register. Use [addStringRegister\(\)](#) to add a register of type 'string'.

The returned builder object is only valid until another resource is added. It is only intended to modify the resource that was added last.

Parameters

<i>name</i>	Name of the register. This is the same as the 'name' field of ResourceInfo.
<i>bitWidth</i>	Width of the resource in bits. This is the same as the 'bitWidth' field of ResourceInfo.
<i>description</i>	Human readable description of the resource. This is the same as the 'description' field of ResourceInfo.
<i>addressOffset</i>	The address offset of this register inside the parent device. This is the same as the 'addressOffset' field of RegisterInfo.
<i>canonicalRn</i>	Canonical Register Number. This is the same as the 'canonicalRn' field of RegisterInfo.

Returns

A [RegisterBuilder](#) object that can be used to set additional metadata for this register resource.

Remarks

A value of $2^{*}64-1$ (0xFFFFFFFFFFFFFFFF) for the arguments *addressOffset* and *canonicalRn* (the default value) is used to indicate that the field is not set. To set an addressOffset of $2^{*}64-1$ use

```
addRegister(...).setAddressOffset(iris::IRIS_UINT64_MAX);
```

To set a canonicalRn of $2^{*}64-1$ use

```
addRegister(...).setCanonicalRn(iris::IRIS_UINT64_MAX);
```

7.2.2.4 addStringParameter()

```
ParameterBuilder iris::IrisInstanceBuilder::addStringParameter (
    const std::string & name,
    const std::string & description )
```

Add string parameter.

Resource group: [beginResourceGroup\(\)](#) must have been called before calling this function. The added parameter is automatically added to the last group added by [beginResourceGroup\(\)](#).

Type: The added parameter is of type 'string'. Use [addParameter\(\)](#) to add a parameter of a type 'numeric' or 'numericFp'.

The returned builder object is only valid until another resource is added. It is only intended to modify the resource that was added last.

Parameters

<i>name</i>	Name of the parameter. This is the same as the 'name' field of ResourceInfo.
<i>description</i>	Human readable description of the parameter. This is the same as the 'description' field of ResourceInfo.

Returns

A [ParameterBuilder](#) object that can be used to set additional metadata for this parameter.

7.2.2.5 addStringRegister()

```
RegisterBuilder iris::IrisInstanceBuilder::addStringRegister (
    const std::string & name,
    const std::string & description )
```

Add metadata for one string register resource.

Resource group: [beginResourceGroup\(\)](#) must have been called before calling this function. The added resource is automatically added to the last group added by [beginResourceGroup\(\)](#).

Type: The added resource is of type 'string'. Use [addRegister\(\)](#) to add a register of type 'numeric'.

The returned builder object is only valid until another resource is added. It is only intended to modify the resource that was added last.

Parameters

<i>name</i>	Name of the register. This is the same as the 'name' field of ResourceInfo.
<i>description</i>	Human readable description of the resource. This is the same as the 'description' field of ResourceInfo.

Returns

A [RegisterBuilder](#) object that can be used to set additional metadata for this register resource.

7.2.2.6 beginResourceGroup()

```
void iris::IrisInstanceBuilder::beginResourceGroup (
    const std::string & name,
    const std::string & description,
    uint64_t subRscIdStart = IRIS_UINT64_MAX,
    const std::string & cname = std::string() )
```

Begin a new resource group.

This has the following effects:

- Add a resource group if it does not yet exist. (If it already exists under 'name' all other parameters are ignored.)
- Assign all resources that are added by subsequent [addRegister\(\)](#) or [addParameter\(\)](#) calls to this group.

This function must be called before the first resource is added.

Parameters

<i>name</i>	Name of the resource group.
<i>description</i>	Description of the resource group.
<i>subRscIdStart</i>	If not IRIS_UINT64_MAX, start counting from this subRscId when new resources are added.
<i>cname</i>	C identifier-style name to use for this group if it is different from <i>name</i> .

See also

[addParameter](#)
[addStringParameter](#)
[addRegister](#)
[addStringRegister](#)
[addNoValueRegister](#)

7.2.2.7 enhanceParameter()

```
ParameterBuilder iris::IrisInstanceBuilder::enhanceParameter (
    ResourceId rscId ) [inline]
```

Get [ParameterBuilder](#) to enhance a parameter.

This function can be used to add/set meta info to an existing parameter. There is no strong use case for this function as all meta info can be set/added by using chained calls to the set...()/add...() functions directly after adding the parameter.

Usage: irisInstance.getBuilder().enhanceParameter(rscId).setFoo(...).setBar(...);

The returned builder object is only valid until another resource is added. It is only intended to modify the specified resource and to add fields to this resource.

Parameters

<i>rscId</i>	ResourceId of the parameter which is to be modified.
--------------	--

Returns

A [ParameterBuilder](#) object that can be used to set additional metadata for this parameter.

7.2.2.8 enhanceRegister()

```
RegisterBuilder iris::IrisInstanceBuilder::enhanceRegister (
    ResourceId rscId ) [inline]
```

Get [RegisterBuilder](#) to enhance register.

This function can be used to add sub-fields to register fields which is not possible in a chained call. The rscId can be retrieved by using getRscId() in the chained call. This function does not add any resource and does not modify any state.

Usage: irisInstance.getBuilder().enhanceRegister(rscId).setFoo(...).setBar(...).addField(...);

See DummyComponent.h for an example.

The returned builder object is only valid until another resource is added. It is only intended to modify the specified resource and to add fields to this resource.

Parameters

<i>rscId</i>	ResourceId of the resource which is to be modified or to which fields are to be added.
--------------	--

Returns

A [RegisterBuilder](#) object that can be used to set additional metadata for this resource.

7.2.2.9 getResourceInfo()

```
const ResourceInfo & iris::IrisInstanceBuilder::getResourceInfo (
    ResourceId rscId ) [inline]
```

Get ResourceInfo of a previously added register.

The returned reference will only be valid until more resources are added.

Parameters

<i>rscId</i>	Resource Id of the resource.
--------------	------------------------------

7.2.2.10 setDefaultResourceDelegates()

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) READER,
IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &) WRITER>
void iris::IrisInstanceBuilder::setDefaultResourceDelegates (
    T * instance ) [inline]
```

Set both read and write resource delegates if they are defined in the same class.

See also

[setDefaultResourceReadDelegate](#)

[setDefaultResourceWriteDelegate](#)

Template Parameters

<i>T</i>	Class that defines resource read and write delegate methods.
<i>READER</i>	A method of class T which is a resource read delegate.
<i>WRITER</i>	A method of class T which is a resource write delegate.

Parameters

<i>instance</i>	An instance of class T on which READER and WRITER should be called.
-----------------	---

7.2.2.11 setDefaultResourceReadDelegate() [1/3]

```
template<IrisErrorCode(*) (const ResourceInfo &, ResourceReadResult &) FUNC>
void iris::IrisInstanceBuilder::setDefaultResourceReadDelegate ( ) [inline]
```

Set default read access function for all subsequently added resources.

Resources that do not explicitly override the access function using

[addRegister\(...\).setReadDelegate\(...\)](#)

will use this delegate.

Usage: Pass in a global function to delegate resource reading to that function:

```
iris::IrisErrorCode myReadFunction(const iris::ResourceInfo &resourceInfo,
                                   iris::ResourceReadResult &result);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultReadDelegate<myReadFunction>();
builder->addRegister(...); // Uses myReadFunction
```

Template Parameters

<i>FUNC</i>	A function which is a resource read delegate.
-------------	---

7.2.2.12 setDefaultResourceReadDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setDefaultResourceReadDelegate (
    ResourceReadDelegate delegate = ResourceReadDelegate() ) [inline]
```

Set default read access function for all subsequently added resources.

Resources that do not explicitly override the access function using

[addRegister\(...\).setReadDelegate\(...\)](#)

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↔` not_implemented for all resources.

Usage: Pass an instance of ResourceReadDelegate into this function to delegate reading to any class T:

```

class MyClass
{
    ...
    iris::IrisErrorCode myReadFunction(const iris::ResourceInfo &resourceInfo,
                                      iris::ResourceReadResult &result);
};
MyClass myInstanceOfMyClass;
ResourceReadDelegate delegate =
    ResourceReadDelegate::make<MyClass, &MyClass::myReadFunction>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultReadDelegate(delegate);
builder->addRegister(...); // Uses myReadFunction

```

Parameters

<i>delegate</i>	Delegate object which will be called to read resources.
-----------------	---

7.2.2.13 setDefaultResourceReadDelegate() [3/3]

```

template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>
void iris::IrisInstanceBuilder::setDefaultResourceReadDelegate (
    T * instance ) [inline]

```

Set default read access function for all subsequently added resources.

Resources that do not explicitly override the access function using

`addRegister(...).setReadDelegate(...)`

will use this delegate.

Usage: Pass an instance of class T where T::METHOD() is a resource read method:

```

class MyClass
{
    ...
    iris::IrisErrorCode myReadFunction(const iris::ResourceInfo &resourceInfo,
                                      iris::ResourceReadResult &result);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultReadDelegate<MyClass, &MyClass::myReadFunction>(myInstanceOfMyClass);
builder->addRegister(...); // Uses myReadFunction

```

Template Parameters

<i>T</i>	Class that defines a resource read delegate method.
<i>METHOD</i>	A method of class T which is a resource read delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.2.2.14 setDefaultResourceWriteDelegate() [1/3]

```

template<IrisErrorCode(*) (const ResourceInfo &, const ResourceWriteValue &) FUNC>
void iris::IrisInstanceBuilder::setDefaultResourceWriteDelegate ( ) [inline]

```

Set default write access function for all subsequently added resources.

Resources that do not explicitly override the access function using

`addRegister(...).setWriteDelegate(...)`

will use this delegate.

Usage: Pass in a global function to delegate resource writing to that function:

```

iris::IrisErrorCode myWriteFunction(const iris::ResourceInfo &resourceInfo, const uint64_t *data);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultWriteDelegate<myWriteFunction>();
builder->addRegister(...); // Uses myWriteFunction

```

Template Parameters

<i>FUNC</i>	A function that is a resource write delegate.
-------------	---

7.2.2.15 setDefaultResourceWriteDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setDefaultResourceWriteDelegate (
    ResourceWriteDelegate delegate = ResourceWriteDelegate() ) [inline]
```

Set default write access function for all subsequently added resources.

Resources that do not explicitly override the access function using

```
addRegister(...).setWriteDelegate(...)
```

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_not_implemented` for all resources.

Usage: Pass an instance of class T where `T::METHOD()` is a resource write method:

```
class MyClass
{
    ...
    iris::IrisErrorCode myWriteFunction(const iris::ResourceInfo &resourceInfo, const uint64_t *data);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
iris::ResourceWriteDelegate delegate =
    iris::ResourceWriteDelegate::make<MyClass, &MyClass::myWriteFunction>(myInstanceOfMyClass);
builder->setDefaultWriteDelegate(delegate);
builder->addRegister(...); // Uses myWriteFunction
```

Parameters

<i>delegate</i>	Delegate object which will be called to write resources.
-----------------	--

7.2.2.16 setDefaultResourceWriteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &)
METHOD>
```

```
void iris::IrisInstanceBuilder::setDefaultResourceWriteDelegate (
    T * instance ) [inline]
```

Set default write access function for all subsequently added resources.

Resources that do not explicitly override the access function using

```
addRegister(...).setWriteDelegate(...)
```

will use this delegate.

Usage: Pass an instance of class T where `T::METHOD()` is a resource write method:

```
class MyClass
{
    ...
    iris::IrisErrorCode myWriteFunction(const iris::ResourceInfo &resourceInfo, const uint64_t *data);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultWriteDelegate<MyClass, &MyClass::myWriteFunction>(myInstanceOfMyClass);
builder->addRegister(...); // Uses myWriteFunction
```

Template Parameters

<i>T</i>	Class that defines a resource write delegate method.
<i>METHOD</i>	A method of class T which is a resource write delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.2.2.17 setNextSubRscId()

```
void iris::IrisInstanceBuilder::setNextSubRscId (
    uint64_t nextSubRscId ) [inline]
```

Set the rscId that will be used for the next resource to be added.

Resources that are added following this call are assigned subRscIds starting at nextSubRscId.

Parameters

<i>nextSubRscId</i>	The subRscId that is used for the next resource to be added.
---------------------	--

7.2.2.18 setPropertyCanonicalRnScheme()

```
void iris::IrisInstanceBuilder::setPropertyCanonicalRnScheme (
    const std::string & canonicalRnScheme )
```

Set the register.canonicalRnScheme instance property.

This property is visible in the list of properties returned by instance_getProperties().

This property defines the scheme used by the 'canonicalRn' member of the RegisterInfo object. This should be called upon initialization, before other instances have a chance to call instance_getProperties().

When using the function setCanonicalRnElfDwarf() the property is set automatically to "ElfDwarf" and it is not necessary to call this function.

When not calling setCanonicalRn() for any register it is not necessary to call this function. In this case the property will not exist which is ok.

Custom scheme names (other than ElfDwarf) should always be of the form <comnapy-name>.com/<scheme-name> to avoid conflicts.

Parameters

<i>canonicalRnScheme</i>	Name of the canonical register number scheme used by this instance.
--------------------------	---

7.2.2.19 setTag()

```
void iris::IrisInstanceBuilder::setTag (
    ResourceId rscId,
    const std::string & tag )
```

Set a tag for a specific resource.

Parameters

<i>rscId</i>	Resource Id for the resource that will have this tag set.
<i>tag</i>	Name of the boolean tag that will be set to true.

See also

[ResourceBuilder::setTag](#)

[RegisterBuilder::setTag](#)

7.3 IrisInstanceBuilder event APIs

Set up event source metadata and event stream delegates.

Classes

- class `iris::IrisInstanceBuilder::EventSourceBuilder`

Used to set metadata on an EventSource.

Functions

- `EventSourceBuilder iris::IrisInstanceBuilder::addEventSource` (const std::string &name, bool isHidden=false)
Add metadata for an event source.
- `EventSourceBuilder iris::IrisInstanceBuilder::addEventSource` (const std::string &name, IrisEventEmitterBase &event_emitter, bool isHidden=false)
Add metadata for an event source that uses an `IrisEventEmitter`.
- `void iris::IrisInstanceBuilder::deleteEventSource` (const std::string &name)
Delete event source.
- `EventSourceBuilder iris::IrisInstanceBuilder::enhanceEventSource` (const std::string &name)
Enhance existing event source.
- `void iris::IrisInstanceBuilder::finalizeRegisterReadEvent` ()
Finalize set up of an `IrisEventEmitter`.
- `void iris::IrisInstanceBuilder::finalizeRegisterUpdateEvent` ()
Finalize set up of an `IrisEventEmitter`.
- `IrisInstanceEvent * iris::IrisInstanceBuilder::getIrisInstanceEvent` ()
- `bool iris::IrisInstanceBuilder::hasEventSource` (const std::string &name)
Check whether event source already exists.
- `void iris::IrisInstanceBuilder::renameEventSource` (const std::string &name, const std::string &newName)
Rename existing event source.
- `void iris::IrisInstanceBuilder::resetRegisterReadEvent` ()
Reset the active register read event.
- `void iris::IrisInstanceBuilder::resetRegisterUpdateEvent` ()
Reset the active register update event.
- `template<IrisErrorCode(*)(<EventStream *>, const EventSourceInfo &, const std::vector< std::string > &) FUNC>`
`void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate` ()
Set the delegate that helps to create a new event stream for the simulation-specific event.
- `void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate` (`EventStreamCreateDelegate` delegate)
Set the delegate that helps to create a new event stream for the simulation-specific event.
- `template<typename T , IrisErrorCode(T::*)(<EventStream *>, const EventSourceInfo &, const std::vector< std::string > &) METHOD>`
`void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate` (T *instance)
Set the delegate that helps to create a new event stream for the simulation-specific event.
- `EventSourceBuilder iris::IrisInstanceBuilder::setRegisterReadEvent` (const std::string &name, const std::string &description=std::string())
Add a new register read event source.
- `EventSourceBuilder iris::IrisInstanceBuilder::setRegisterReadEvent` (const std::string &name, IrisRegisterEventEmitterBase &event_emitter)
Add a new register read event source.
- `EventSourceBuilder iris::IrisInstanceBuilder::setRegisterUpdateEvent` (const std::string &name, const std::string &description=std::string())
Add a new register update event source.
- `EventSourceBuilder iris::IrisInstanceBuilder::setRegisterUpdateEvent` (const std::string &name, IrisRegisterEventEmitterBase &event_emitter)
Add a new register update event source.

7.3.1 Detailed Description

Set up event source metadata and event stream delegates.

7.3.2 Function Documentation

7.3.2.1 addEventSource() [1/2]

```
EventSourceBuilder iris::IrisInstanceBuilder::addEventSource (
    const std::string & name,
    bool isHidden = false ) [inline]
```

Add metadata for an event source.

Consider using `addEventSource(const std::string& name, IrisEventEmitterBase& event_emitter)` instead. Only use this if you want to implement a non-trivial trace source with its own event emitter handling.

Parameters

<i>name</i>	The name of the new event source.
<i>isHidden</i>	If true, the event source is hidden.

See also

[EventSourceBuilder::setHidden](#)

Returns

An [EventSourceBuilder](#) object that can be used to set additional attributes for this event source. The returned [EventSourceBuilder](#) is only valid until the next call to [addEventSource\(\)](#).

7.3.2.2 addEventSource() [2/2]

```
EventSourceBuilder iris::IrisInstanceBuilder::addEventSource (
    const std::string & name,
    IrisEventEmitterBase & event_emitter,
    bool isHidden = false ) [inline]
```

Add metadata for an event source that uses an [IrisEventEmitter](#).

Parameters

<i>name</i>	The name of the new event source.
<i>event_emitter</i>	The IrisEventEmitter for this event source.
<i>isHidden</i>	If true, the event source is hidden.

See also

[EventSourceBuilder::setHidden](#)

Returns

An [EventSourceBuilder](#) object that can be used to set additional attributes for this event source. The returned [EventSourceBuilder](#) is only valid until the next call to [addEventSource\(\)](#), [setRegisterReadEvent\(\)](#), or [setRegisterWriteEvent\(\)](#).

7.3.2.3 deleteEventSource()

```
void iris::IrisInstanceBuilder::deleteEventSource (
    const std::string & name ) [inline]
```

Delete event source.

Parameters

<i>name</i>	The name of the event source.
-------------	-------------------------------

7.3.2.4 enhanceEventSource()

```
EventSourceBuilder iris::IrisInstanceBuilder::enhanceEventSource (
    const std::string & name ) [inline]
```

Enhance existing event source.

Parameters

<i>name</i>	The name of the event source.
-------------	-------------------------------

Returns

An [EventSourceBuilder](#) object that can be used to set additional attributes for this event source. The returned [EventSourceBuilder](#) is only valid until the next call to [addEventSource\(\)](#), [setRegisterReadEvent\(\)](#), or [setRegisterWriteEvent\(\)](#).

7.3.2.5 finalizeRegisterReadEvent()

```
void iris::IrisInstanceBuilder::finalizeRegisterReadEvent ( )
```

Finalize the setup of an [IrisEventEmitter](#).

When all the registers associated with all the read events have been added, call [finalizeRegisterReadEvent\(\)](#) to add the event sources to the [IrisInstance](#).

7.3.2.6 finalizeRegisterUpdateEvent()

```
void iris::IrisInstanceBuilder::finalizeRegisterUpdateEvent ( )
```

Finalize set up of an [IrisEventEmitter](#).

When all the registers associated with all the write events have been added, call [finalizeRegisterUpdateEvent\(\)](#) to add the event sources to the [IrisInstance](#).

7.3.2.7 getIrisInstanceEvent()

```
IrisInstanceEvent * iris::IrisInstanceBuilder::getIrisInstanceEvent ( ) [inline]
```

Direct access to [IrisInstanceEvent](#).

Do not use! This will be removed! Use the event api of [IrisInstanceBuilder](#) instead. This is a temporary hack.

7.3.2.8 hasEventSource()

```
bool iris::IrisInstanceBuilder::hasEventSource (
    const std::string & name ) [inline]
```

Check whether event source already exists.

Parameters

<i>name</i>	The name of the event source.
-------------	-------------------------------

Returns

True iff the event source already exists.

7.3.2.9 renameEventSource()

```
void iris::IrisInstanceBuilder::renameEventSource (
    const std::string & name,
    const std::string & newName ) [inline]
```

Rename existing event source.

Parameters

<i>name</i>	The old name of the event source.
<i>newName</i>	The new name of the event source.

7.3.2.10 resetRegisterReadEvent()

```
void iris::IrisInstanceBuilder::resetRegisterReadEvent ( )
```

Reset the active register read event.

setRegisterReadEvent and resetRegisterReadEvent should be called in pair to scope the registers being added to be associated with a certain read event.

7.3.2.11 resetRegisterUpdateEvent()

```
void iris::IrisInstanceBuilder::resetRegisterUpdateEvent ( )
```

Reset the active register update event.

setRegisterUpdateEvent and resetRegisterUpdateEvent should be called in pair to scope the registers being added to be associated with a certain update event.

7.3.2.12 setDefaultEsCreateDelegate() [1/3]

```
template<IrisErrorCode(*) (EventStream *&, const EventSourceInfo &, const std::vector< std::string > &) FUNC>
```

```
void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate ( ) [inline]
```

Set the delegate that helps to create a new event stream for the simulation-specific event.

Consider using addEventSource(const std::string& name, IrisEventEmitterBase& event_emitter) instead. Only use this if you want to implement a non-trivial trace source with its own event emitter handling.

Event sources that do not explicitly override the access function using

```
addEventSource(...).setEventStreamCreateDelegate(...)
```

use this delegate.

Usage: Pass in a global function to which to delegate event stream creation:

```
iris::IrisErrorCode createEventStream(iris::EventStream*&, const iris::EventSourceInfo&,
                                     const std::vector<std::string>&)>
builder->setDefaultEsCreateDelegate<MyClass::createEventStream>();
builder->addEventSource(...); // Uses createEventStream
```

Template Parameters

<i>FUNC</i>	Global function to which to delegate event stream creation.
-------------	---

7.3.2.13 setDefaultEsCreateDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate (
    EventStreamCreateDelegate delegate ) [inline]
```

Set the delegate that helps to create a new event stream for the simulation-specific event.

Consider using addEventSource(const std::string& name, IrisEventEmitterBase& event_emitter) instead. Only use this if you want to implement a non-trivial trace source with its own event emitter handling.

Event sources that do not explicitly override the access function using

```
addEventSource(...).setEventStreamCreateDelegate(...)
```

use this delegate.

Usage: Pass an instance of class T where T::METHOD() is an event stream creation method:

```
class MyClass
{
    ...
    iris::IrisErrorCode createEventStream(iris::EventStream*&, const iris::EventSourceInfo&,
                                         const std::vector<std::string>&)>
};
MyClass myInstanceOfMyClass;
EventStreamCreateDelegate delegate = EventStreamCreateDelegate::make<MyClass,
    &MyClass::createEventStream>(myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultEsCreateDelegateC(delegate);
builder->addEventSource(...); // Uses createEventStream
```

Parameters

<i>delegate</i>	Delegate object that will be called to create an event stream.
-----------------	--

7.3.2.14 setDefaultEsCreateDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(EventStream *amp;, const EventSourceInfo &, const std::vector< std::string > &) METHOD>
void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate (
    T * instance ) [inline]
```

Set the delegate that helps to create a new event stream for the simulation-specific event.

Consider using addEventSource(const std::string& name, IrisEventEmitterBase& event_emitter) instead. Only use this if you want to implement a non-trivial trace source with its own event emitter handling.

Event sources that do not explicitly override the access function using addEventSource(...).setEventStreamCreateDelegate(...)

use this delegate.

Usage: Pass an instance of class T where T::METHOD() is an event stream creation method:

```
class MyClass
{
    ...
    iris::IrisErrorCode createEventStream(iris::EventStream*&, const iris::EventSourceInfo&,
                                         const std::vector<std::string>&)>
};
MyClass myInstanceOfMyClass;
builder->setDefaultEsCreateDelegate<MyClass, &MyClass::createEventStream>(myInstanceOfMyClass);
builder->addEventSource(...); // Uses createEventStream
```

Template Parameters

<i>T</i>	Class that defines an event stream creation method.
<i>METHOD</i>	A method of class T which is an event stream creation method.

Parameters

<i>instance</i>	The instance of class T on which METHOD should be called.
-----------------	---

7.3.2.15 setRegisterReadEvent() [1/2]

```
EventSourceBuilder iris::IrisInstanceBuilder::setRegisterReadEvent (
    const std::string & name,
    const std::string & description = std::string() )
```

Add a new register read event source.

Any registers added after calling setRegisterReadEvent() and before the next call to setRegisterReadEvent() or finalizeRegisterReadEvent() are associated with this event.

A call to setRegisterReadEvent() implicitly calls finalizeRegisterReadEvent() on the event source with name name iff an event emitter object (type IrisRegisterEventEmitterBase) is provided as an argument.

If the register read event source already exists (identified by name), the active register read event source simply switches to it.

Register read events have three standard fields:

Field	Description
REGISTER	The Iris rsclid of the register accessed.
DEBUG	True if the read originated from a debug access.
VALUE	The value that was read.

Parameters

<i>name</i>	Name of the event source.
<i>description</i>	Description of the event source.

Returns

An [EventSourceBuilder](#) for the event allowing extra custom fields to be added.

7.3.2.16 setRegisterReadEvent() [2/2]

```
EventSourceBuilder iris::IrisInstanceBuilder::setRegisterReadEvent (
    const std::string & name,
    IrisRegisterEventEmitterBase & event_emitter )
```

Add a new register read event source.

Any registers added after calling [setRegisterReadEvent\(\)](#) and before the next call to [setRegisterReadEvent\(\)](#) or [finalizeRegisterReadEvent\(\)](#) are associated with this event.

A call to [setRegisterReadEvent\(\)](#) implicitly calls [finalizeRegisterReadEvent\(\)](#) on the event source with name *name* iff an event emitter object (type `IrisRegisterEventEmitterBase`) is provided as an argument.

If the register read event source already exists (identified by name), the active register read event source simply switches to it.

Register read events have three standard fields:

Field	Description
REGISTER	The Iris rsclid of the register accessed.
DEBUG	True if the read originated from a debug access.
VALUE	The value that was read.

Parameters

<i>name</i>	Name of the event source.
<i>event_emitter</i>	The <i>event_emitter</i> to associate with this event source.

Returns

An [EventSourceBuilder](#) for the event allowing extra custom fields to be added.

7.3.2.17 setRegisterUpdateEvent() [1/2]

```
EventSourceBuilder iris::IrisInstanceBuilder::setRegisterUpdateEvent (
    const std::string & name,
    const std::string & description = std::string() )
```

Add a new register update event source.

Any registers added after calling [setRegisterUpdateEvent\(\)](#) and before the next call to [setRegisterUpdateEvent\(\)](#) or [finalizeRegisterUpdateEvent\(\)](#) are associated with this event.

A call to [setRegisterUpdateEvent](#) implicitly calls [finalizeRegisterUpdateEvent\(\)](#) on the event source with name `name` iff an event emitter object (type `IrisRegisterEventEmitterBase`) is provided as an argument.

If the register update event source (identified by name) already exists, the active register update event source simply switches to it.

Register update events have four standard fields:

Field	Description
REGISTER	The Iris rscl of the register accessed.
DEBUG	True if the update originated from a debug access.
OLD_VALUE	The value that would have been read before the access was made.
NEW_VALUE	The value that would be read after the access was made.

Parameters

<i>name</i>	Name of the event source.
<i>description</i>	Description of the event source.

Returns

An [EventSourceBuilder](#) for the event allowing extra custom fields to be added.

7.3.2.18 setRegisterUpdateEvent() [2/2]

```
EventSourceBuilder iris::IrisInstanceBuilder::setRegisterUpdateEvent (
    const std::string & name,
    IrisRegisterEventEmitterBase & event_emitter )
```

Add a new register update event source.

Any registers added after calling [setRegisterUpdateEvent\(\)](#) and before the next call to [setRegisterUpdateEvent\(\)](#) or [finalizeRegisterUpdateEvent\(\)](#) are associated with this event.

A call to [setRegisterUpdateEvent](#) implicitly calls [finalizeRegisterUpdateEvent\(\)](#) on the event source with name `name` iff an event emitter object (type `IrisRegisterEventEmitterBase`) is provided as an argument.

If the register update event source (identified by name) already exists, the active register update event source simply switches to it.

Register update events have four standard fields:

Field	Description
REGISTER	The Iris rscl of the register accessed.
DEBUG	True if the update originated from a debug access.
OLD_VALUE	The value that would have been read before the access was made.
NEW_VALUE	The value that would be read after the access was made.

Parameters

<i>name</i>	Name of the event source.
<i>event_emitter</i>	The event_emitter to associate with this event source.

Returns

An [EventSourceBuilder](#) for the event allowing extra custom fields to be added.

7.4 IrisInstanceBuilder breakpoint APIs

Set up breakpoint hit notifications and breakpoint delegates.

Functions

- void [iris::IrisInstanceBuilder::addBreakpointCondition](#) (const std::string &name, const std::string &type, const std::string &description, const std::vector< std::string > bpt_types=std::vector< std::string >())
Add an optional component-specific condition.
- const BreakpointInfo * [iris::IrisInstanceBuilder::getBreakpointInfo](#) (BreakpointId bptId)
Get the breakpoint information for a given breakpoint.
- void [iris::IrisInstanceBuilder::notifyBreakpointHit](#) (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId)
Notify clients that a code breakpoint was hit.
- void [iris::IrisInstanceBuilder::notifyBreakpointHitData](#) (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId, uint64_t accessAddr, uint64_t accessSize, const std::string &accessRw, const std::vector< uint64_t > &data)
Notify clients that a data breakpoint was hit (IRIS_BREAKPOINT_HIT).
- void [iris::IrisInstanceBuilder::notifyBreakpointHitRegister](#) (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId, const std::string &accessRw, const std::vector< uint64_t > &data)
Notify clients that a register breakpoint was hit (IRIS_BREAKPOINT_HIT).
- template<IrisErrorCode(*) (const BreakpointInfo &) FUNC>
void [iris::IrisInstanceBuilder::setBreakpointDeleteDelegate](#) ()
Set the delegate that is called when a breakpoint is deleted.
- void [iris::IrisInstanceBuilder::setBreakpointDeleteDelegate](#) (BreakpointDeleteDelegate delegate)
Set the delegate that is called when a breakpoint is deleted.
- template<typename T, IrisErrorCode(T::*)(const BreakpointInfo &) METHOD>
void [iris::IrisInstanceBuilder::setBreakpointDeleteDelegate](#) (T *instance)
Set the delegate that is called when a breakpoint is deleted.
- template<IrisErrorCode(*) (BreakpointInfo &) FUNC>
void [iris::IrisInstanceBuilder::setBreakpointSetDelegate](#) ()
Set the delegate that is called when a breakpoint is set.
- void [iris::IrisInstanceBuilder::setBreakpointSetDelegate](#) (BreakpointSetDelegate delegate)
Set the delegate that is called when a breakpoint is set.
- template<typename T, IrisErrorCode(T::*)(BreakpointInfo &) METHOD>
void [iris::IrisInstanceBuilder::setBreakpointSetDelegate](#) (T *instance)
Set the delegate that is called when a breakpoint is set.
- template<IrisErrorCode(*) (const BreakpointHitInfo &) FUNC>
void [iris::IrisInstanceBuilder::setHandleBreakpointHitDelegate](#) ()
Set the delegate that is called when a breakpoint is hit.
- void [iris::IrisInstanceBuilder::setHandleBreakpointHitDelegate](#) (HandleBreakpointHitDelegate delegate)
Set the delegate that is called when a breakpoint is hit.
- template<typename T, IrisErrorCode(T::*)(const BreakpointHitInfo &) METHOD>
void [iris::IrisInstanceBuilder::setHandleBreakpointHitDelegate](#) (T *instance)
Set the delegate that is called when a breakpoint is hit.

7.4.1 Detailed Description

Set up breakpoint hit notifications and breakpoint delegates.

7.4.2 Function Documentation

7.4.2.1 getBreakpointInfo()

```
const BreakpointInfo * iris::IrisInstanceBuilder::getBreakpointInfo (
    BreakpointId bptId ) [inline]
```

Get the breakpoint information for a given breakpoint.

Parameters

<i>bptId</i> ↔ <i>Id</i>	The breakpoint id of the breakpoint for which information is being requested.
-----------------------------	---

Returns

The breakpoint information for the requested breakpoint. This returns nullptr if *bptId* is invalid.

7.4.2.2 notifyBreakpointHit()

```
void iris::IrisInstanceBuilder::notifyBreakpointHit (
    BreakpointId bptId,
    uint64_t time,
    uint64_t pc,
    MemorySpaceId pcSpaceId ) [inline]
```

Notify clients that a code breakpoint was hit.

This emits an (IRIS_BREAKPOINT_HIT) event.

Parameters

<i>bptId</i>	Breakpoint id for the breakpoint that was hit.
<i>time</i>	Simulation time at which the breakpoint was hit.
<i>pc</i>	Value of the program counter when the breakpoint was hit.
<i>pc</i> ↔ <i>SpaceId</i>	Memory space id for the PC when the breakpoint was hit.

7.4.2.3 notifyBreakpointHitData()

```
void iris::IrisInstanceBuilder::notifyBreakpointHitData (
    BreakpointId bptId,
    uint64_t time,
    uint64_t pc,
    MemorySpaceId pcSpaceId,
    uint64_t accessAddr,
    uint64_t accessSize,
    const std::string & accessRw,
    const std::vector< uint64_t > & data ) [inline]
```

Notify clients that a data breakpoint was hit (IRIS_BREAKPOINT_HIT).

This emits an (IRIS_BREAKPOINT_HIT) event.

Parameters

<i>bptId</i>	Breakpoint id for the breakpoint that was hit.
--------------	--

Parameters

<i>time</i>	Simulation time at which the breakpoint was hit.
<i>pc</i>	Value of the program counter when the breakpoint was hit.
<i>pcSpaceId</i>	Memory space id for the PC when the breakpoint was hit.
<i>accessAddr</i>	Address of the access that hit.
<i>accessSize</i>	Size in bytes of the access that hit.
<i>accessRw</i>	Access direction. Should be "r" for a read access or "w" for a write access.
<i>data</i>	The data transferred by the access that hit.

7.4.2.4 notifyBreakpointHitRegister()

```
void iris::IrisInstanceBuilder::notifyBreakpointHitRegister (
    BreakpointId bptId,
    uint64_t time,
    uint64_t pc,
    MemorySpaceId pcSpaceId,
    const std::string & accessRw,
    const std::vector< uint64_t > & data ) [inline]
```

Notify clients that a register breakpoint was hit (IRIS_BREAKPOINT_HIT).

This emits an (IRIS_BREAKPOINT_HIT) event.

Parameters

<i>bptId</i>	Breakpoint id for the breakpoint that was hit.
<i>time</i>	Simulation time at which the breakpoint was hit.
<i>pc</i>	Value of the program counter when the breakpoint was hit.
<i>pc↔ SpaceId</i>	Memory space id for the PC when the breakpoint was hit.
<i>accessRw</i>	Access direction. Should be "r" for a read access or "w" for a write access.
<i>data</i>	The data transferred by the access that hit.

7.4.2.5 setBreakpointDeleteDelegate() [1/3]

```
template<IrisErrorCode(*) (const BreakpointInfo &) FUNC>
void iris::IrisInstanceBuilder::setBreakpointDeleteDelegate ( ) [inline]
```

Set the delegate that is called when a breakpoint is deleted.

Usage: Pass in a global function to call when a breakpoint is deleted:

```
iris::IrisErrorCode deleteBreakpoint(const iris::BreakpointInfo&);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setBreakpointDeleteDelegate(&deleteBreakpoint);
```

Template Parameters

<i>FUNC</i>	Global function to call when a breakpoint is deleted.
-------------	---

7.4.2.6 setBreakpointDeleteDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setBreakpointDeleteDelegate (
    BreakpointDeleteDelegate delegate ) [inline]
```

Set the delegate that is called when a breakpoint is deleted.

Usage: Pass a breakpoint delete delegate:

```
class MyClass
{
    ...
    iris::IrisErrorCode deleteBreakpoint(const iris::BreakpointInfo&);
};
MyClass myInstanceOfMyClass;
BreakpointDeleteDelegate delegate = BreakpointDeleteDelegate::make<MyClass,
    &MyClass::deleteBreakpoint>(myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setBreakpointDeleteDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object which will be called to delete a breakpoint.
-----------------	--

7.4.2.7 setBreakpointDeleteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const BreakpointInfo &) METHOD>
void iris::IrisInstanceBuilder::setBreakpointDeleteDelegate (
    T * instance ) [inline]
```

Set the delegate that is called when a breakpoint is deleted.

Usage: Pass an instance of class T, where T::METHOD() is a breakpoint delete delegate:

```
class MyClass
{
    ...
    iris::IrisErrorCode deleteBreakpoint(const iris::BreakpointInfo&);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setBreakpointDeleteDelegate<MyClass, &MyClass::deleteBreakpoint>(myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines a breakpoint delete method.
<i>METHOD</i>	A method of class T which is a breakpoint delete delegate method.

Parameters

<i>instance</i>	The instance of class T on which METHOD should be called.
-----------------	---

7.4.2.8 setBreakpointSetDelegate() [1/3]

```
template<IrisErrorCode(*) (BreakpointInfo &) FUNC>
void iris::IrisInstanceBuilder::setBreakpointSetDelegate ( ) [inline]
```

Set the delegate that is called when a breakpoint is set.

Usage: Pass in a global function to call when a breakpoint is set:

```
iris::IrisErrorCode setBreakpoint(iris::BreakpointInfo&);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setBreakpointSetDelegate<&setBreakpoint>();
```

Template Parameters

<i>FUNC</i>	Global function to call when a breakpoint is set.
-------------	---

7.4.2.9 setBreakpointSetDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setBreakpointSetDelegate (
```

```
BreakpointSetDelegate delegate ) [inline]
```

Set the delegate that is called when a breakpoint is set.

Usage: Pass a breakpoint set delegate:

```
class MyClass
{
    ...
    iris::IrisErrorCode setBreakpoint(iris::BreakpointInfo&);
};
MyClass myInstanceOfMyClass;
BreakpointSetDelegate delegate = BreakpointSetDelegate::make<MyClass,
    &MyClass::setBreakpoint>(myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setBreakpointSetDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object which will be called to set a breakpoint.
-----------------	---

7.4.2.10 setBreakpointSetDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(BreakpointInfo &) METHOD>
void iris::IrisInstanceBuilder::setBreakpointSetDelegate (
    T * instance ) [inline]
```

Set the delegate that is called when a breakpoint is set.

Usage: Pass an instance of class T, where T::METHOD() is a breakpoint set delegate:

```
class MyClass
{
    ...
    iris::IrisErrorCode setBreakpoint(iris::BreakpointInfo&);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setBreakpointSetDelegate<MyClass, &MyClass::setBreakpoint>(myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines a breakpoint set method.
<i>METHOD</i>	A method of class T which is a breakpoint set delegate method.

Parameters

<i>instance</i>	The instance of class T on which METHOD should be called.
-----------------	---

7.4.2.11 setHandleBreakpointHitDelegate() [1/3]

```
template<IrisErrorCode(*) (const BreakpointHitInfo &) FUNC>
void iris::IrisInstanceBuilder::setHandleBreakpointHitDelegate ( ) [inline]
```

Set the delegate that is called when a breakpoint is hit.

Usage: Pass in a global function to call when a breakpoint is hit.

```
iris::IrisErrorCode handleBreakpointHit(const iris::BreakpointHitInfo&);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setHandleBreakpointHitDelegate(&handleBreakpointHit);
```

Template Parameters

<i>FUNC</i>	Global function to call when a breakpoint is hit.
-------------	---

7.4.2.12 setHandleBreakpointHitDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setHandleBreakpointHitDelegate (
    HandleBreakpointHitDelegate delegate ) [inline]
```

Set the delegate that is called when a breakpoint is hit.

Usage: Pass a handle breakpoint hit delegate:

```
class MyClass
{
    ...
    iris::IrisErrorCode handleBreakpointHit (const iris::BreakpointHitInfo&);
};
MyClass myInstanceOfMyClass;
HandleBreakpointHitDelegate delegate = HandleBreakpointHitDelegate::make<MyClass,
    &MyClass::handleBreakpointHit>(myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setHandleBreakpointHitDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object which will be called to handle a breakpoint hit.
-----------------	--

7.4.2.13 setHandleBreakpointHitDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const BreakpointHitInfo &) METHOD>
void iris::IrisInstanceBuilder::setHandleBreakpointHitDelegate (
    T * instance ) [inline]
```

Set the delegate that is called when a breakpoint is hit.

Usage: Pass an instance of class T, where T::METHOD() is a handle breakpoint hit delegate:

```
class MyClass
{
    ...
    iris::IrisErrorCode handleBreakpointHit (const iris::BreakpointHitInfo&);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setHandleBreakpointHitDelegate<MyClass, &MyClass::handleBreakpointHit>(myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines a handle breakpoint hit method.
<i>METHOD</i>	A method of class T which is a handle breakpoint hit delegate method.

Parameters

<i>instance</i>	The instance of class T on which METHOD should be called.
-----------------	---

7.5 IrisInstanceBuilder memory APIs

Set up address translation and memory space metadata and delegates.

Classes

- class [iris::IrisInstanceBuilder::AddressTranslationBuilder](#)
Used to set metadata for an address translation.
- class [iris::IrisInstanceBuilder::MemorySpaceBuilder](#)
Used to set metadata for a memory space.

Functions

- [AddressTranslationBuilder](#) [iris::IrisInstanceBuilder::addAddressTranslation](#) (MemorySpaceId inSpaceId, MemorySpaceId outSpaceId, const std::string &description)
Add an address translation.
- [MemorySpaceBuilder](#) [iris::IrisInstanceBuilder::addMemorySpace](#) (const std::string &name)
Add metadata for one memory space.
- `template<IrisErrorCode(*)>(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) FUNC>`
`void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate ()`
Set the default address translation function for all subsequently added memory spaces.
- `void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate (MemoryAddressTranslateDelegate delegate=MemoryAddressTranslateDelegate())`
Set the default address translation function for all subsequently added memory spaces.
- `template<typename T, IrisErrorCode(T::*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) METHOD>`
`void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate (T *instance)`
Set the default address translation function for all subsequently added memory spaces.
- `template<IrisErrorCode(*)>(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) FUNC>`
`void iris::IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate ()`
Set the default sideband info function for all subsequently added memory spaces.
- `void iris::IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate (MemoryGetSidebandInfoDelegate delegate)`
Set the default sideband info function for all subsequently added memory spaces.
- `template<typename T, IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) METHOD>`
`void iris::IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate (T *instance)`
Set the default sideband info function for all subsequently added memory spaces.
- `template<IrisErrorCode(*)>(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) FUNC>`
`void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate ()`
Set the default read function for all subsequently added memory spaces.
- `void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate (MemoryReadDelegate delegate=MemoryReadDelegate())`
Set the default read function for all subsequently added memory spaces.
- `template<typename T, IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) METHOD>`
`void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate (T *instance)`
Set the default read function for all subsequently added memory spaces.
- `template<IrisErrorCode(*)>(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) FUNC>`
`void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate ()`
Set default write function for all subsequently added memory spaces.
- `void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate (MemoryWriteDelegate delegate=MemoryWriteDelegate())`
Set the default write function for all subsequently added memory spaces.
- `template<typename T, IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>`
`void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate (T *instance)`
Set the default write function for all subsequently added memory spaces.
- `void iris::IrisInstanceBuilder::setPropertyCanonicalMsnScheme (const std::string &canonicalMsnScheme)`
Set the memory.canonicalMsnScheme instance property.

7.5.1 Detailed Description

Set up address translation and memory space metadata and delegates.

7.5.2 Function Documentation

7.5.2.1 addAddressTranslation()

```
AddressTranslationBuilder iris::IrisInstanceBuilder::addAddressTranslation (
    MemorySpaceId inSpaceId,
    MemorySpaceId outSpaceId,
    const std::string & description ) [inline]
```

Add an address translation.

Add metadata for the address translation from the memory space indicated by *inSpaceId* to the memory space indicated by *outSpaceId*.

By explicitly adding an address translation using this function, the Iris instance can tell clients which address translations are supported and a component can provide a specific delegate function to perform that translation.

Parameters

<i>inSpaceId</i>	Memory space id for the input memory space of this translation.
<i>outSpaceId</i>	Memory space id for the output memory space of this translation.
<i>description</i>	A human readable description of this translation. return An AddressTranslationBuilder object which allows additional configuration of this translation.

7.5.2.2 addMemorySpace()

```
MemorySpaceBuilder iris::IrisInstanceBuilder::addMemorySpace (
    const std::string & name ) [inline]
```

Add metadata for one memory space.

Typical use pattern:

```
addMemorySpace("name")
    .setDescription("description")
    .setMinAddr(...)
    .setMaxAddr(...)
    .setEndianness(...)
    .addAttribute(...)
    .addAttributeDefault(...);
```

Parameters

<i>name</i>	Name of the memory space to add.
-------------	----------------------------------

Returns

A [MemorySpaceBuilder](#) object which can be used to configure metadata for the memory space.

7.5.2.3 setDefaultAddressTranslateDelegate() [1/3]

```
template<IrisErrorCode(*) (uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &)
FUNC>
```

```
void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate ( ) [inline]
```

Set the default address translation function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the access function using

```
addMemorySpace(...).setTranslationDelegate(...)
```

will use this delegate.

Usage:

```
iris::IrisErrorCode translateAddress(MemorySpaceId inSpaceId, uint64_t address,
    MemorySpaceId outSpaceId,
```



```

        iris::MemoryAddressTranslationResult &result);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultAddressTranslateDelegate(&translateAddress);
builder->addMemorySpace(...); // Uses translateAddress

```

Template Parameters

<i>FUNC</i>	Global function to call to translate addresses.
-------------	---

7.5.2.4 setDefaultAddressTranslateDelegate() [2/3]

```

void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate (
    MemoryAddressTranslateDelegate delegate = MemoryAddressTranslateDelegate() )

```

[inline]

Set the default address translation function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the access function using

`addMemorySpace(...).setTranslationDelegate(...)`

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_not_implemented` for all requests.

Usage:

```

class MyClass
{
    ...
    iris::IrisErrorCode translateAddress(MemorySpaceId inSpaceId, uint64_t address,
                                        MemorySpaceId outSpaceId,
                                        iris::MemoryAddressTranslationResult &result);
};
MyClass myInstanceOfMyClass;
iris::MemoryAddressTranslateDelegate delegate =
    iris::MemoryAddressTranslateDelegate::make<MyClass, &MyClass::translateAddress>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultAddressTranslateDelegate(delegate);
builder->addMemorySpace(...); // Uses translateAddress

```

Parameters

<i>delegate</i>	Delegate object which will be called to translate addresses.
-----------------	--

7.5.2.5 setDefaultAddressTranslateDelegate() [3/3]

```

template<typename T , IrisErrorCode(T::*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) METHOD>

```

```

void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate (
    T * instance ) [inline]

```

Set the default address translation function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the access function using

`addMemorySpace(...).setTranslationDelegate(...)`

will use this delegate.

Usage:

```

class MyClass
{
    ...
    iris::IrisErrorCode translateAddress(MemorySpaceId inSpaceId, uint64_t address,
                                        MemorySpaceId outSpaceId,
                                        iris::MemoryAddressTranslationResult &result);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultAddressTranslateDelegate<MyClass, &MyClass::translateAddress>(&myInstanceOfMyClass);
builder->addMemorySpace(...); // Uses translateAddress

```

Template Parameters

<i>T</i>	Class that defines an address translation delegate method.
<i>METHOD</i>	A method of class T which is an address translation delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.5.2.6 setDefaultGetMemorySidebandInfoDelegate() [1/3]

```
template<IrisErrorCode(*) (const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) FUNC>
```

```
void iris::IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate ( ) [inline]
```

Set the default sideband info function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the sideband function using

```
addMemorySpace(...).setSidebandDelegate(...)
```

will use this delegate.

Usage:

```
iris::IrisErrorCode getSidebandInfo(const iris::MemorySpaceInfo &spaceInfo, uint64_t address,
                                   const iris::IrisValueMap &attrib,
                                   const std::vector<std::string> &request,
                                   iris::IrisValueMap &result);

iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultGetMemorySidebandInfoDelegate(&getSidebandInfo);
builder->addMemorySpace(...); // Uses getSidebandInfo
```

Template Parameters

<i>FUNC</i>	Global function to call to get sideband info.
-------------	---

7.5.2.7 setDefaultGetMemorySidebandInfoDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate (
    MemoryGetSidebandInfoDelegate delegate ) [inline]
```

Set the default sideband info function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the sideband function using

```
addMemorySpace(...).setSidebandDelegate(...)
```

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns E_↵ not implemented for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getSidebandInfo(const iris::MemorySpaceInfo &spaceInfo, uint64_t address,
                                       const iris::IrisValueMap &attrib,
                                       const std::vector<std::string> &request,
                                       iris::IrisValueMap &result);
};

MyClass myInstanceOfMyClass;
iris::MemoryAddressTranslateDelegate delegate =
    iris::MemoryAddressTranslateDelegate::make<MyClass, &MyClass::getSidebandInfo>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultGetMemorySidebandInfoDelegate(delegate);
builder->addMemorySpace(...); // Uses getSidebandInfo
```

Parameters

<i>delegate</i>	Delegate object which will be called to get sideband info.
-----------------	--

7.5.2.8 setDefaultGetMemorySidebandInfoDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, const IrisValue↵
Map &, const std::vector< std::string > &, IrisValueMap &) METHOD>
void iris::IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate (
    T * instance ) [inline]
```

Set the default sideband info function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the sideband function using

`addMemorySpace(...).setSidebandDelegate(...)`

will use this delegate.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getSidebandInfo(const iris::MemorySpaceInfo &spaceInfo, uint64_t address,
                                        const iris::IrisValueMap &attrib,
                                        const std::vector<std::string> &request,
                                        iris::IrisValueMap &result);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultGetMemorySidebandInfoDelegate<MyClass, &MyClass::getSidebandInfo>(&myInstanceOfMyClass);
builder->addMemorySpace(...); // Uses getSidebandInfo
```

Template Parameters

<i>T</i>	Class that defines a sideband info delegate method.
<i>METHOD</i>	A method of class T which is a sideband info delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.5.2.9 setDefaultMemoryReadDelegate() [1/3]

```
template<IrisErrorCode(*) (const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const Attribute↵
ValueMap &, MemoryReadResult &) FUNC>
void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate ( ) [inline]
```

Set the default read function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the access function using

`addMemorySpace(...).setReadDelegate(...)`

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns E_↵

not_implemented for all requests.

Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

```
iris::IrisErrorCode readMemory(const iris::MemorySpaceInfo &spaceInfo, uint64_t address,
                              uint64_t byteWidth, uint64_t count,
                              const iris::IrisValueMap &attrib,
                              iris::MemoryReadResult &result);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultMemoryReadDelegate<readMemory>();
builder->addMemorySpace(...); // Uses readMemory
```

Template Parameters

<i>FUNC</i>	A memory read delegate function.
-------------	----------------------------------

7.5.2.10 setDefaultMemoryReadDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate (
    MemoryReadDelegate delegate = MemoryReadDelegate() ) [inline]
```

Set the default read function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the access function using

```
addMemorySpace(...).setReadDelegate(...)
```

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_not_implemented` for all requests.

Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

```
class MyClass
{
    ...
    iris::IrisErrorCode readMemory(const iris::MemorySpaceInfo &spaceInfo, uint64_t address,
                                   uint64_t byteWidth, uint64_t count,
                                   const iris::IrisValueMap &attrib,
                                   iris::MemoryReadResult &result);
};

MyClass myInstanceOfMyClass;
iris::MemoryReadDelegate delegate =
    iris::MemoryReadDelegate::make<MyClass, &MyClass::readMemory>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultMemoryReadDelegate(delegate);
builder->addMemorySpace(...); // Uses readMemory
```

Parameters

<i>delegate</i>	Delegate object which will be called to read memory.
-----------------	--

7.5.2.11 setDefaultMemoryReadDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t,
    _t, const AttributeValueMap &, MemoryReadResult &) METHOD>
void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate (
    T * instance ) [inline]
```

Set the default read function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the access function using

```
addMemorySpace(...).setReadDelegate(...)
```

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_not_implemented` for all requests.

Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

```
class MyClass
{
    ...
    iris::IrisErrorCode readMemory(const iris::MemorySpaceInfo &spaceInfo, uint64_t address,
                                   uint64_t byteWidth, uint64_t count,
                                   const iris::IrisValueMap &attrib,
                                   iris::MemoryReadResult &result);
};

MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultMemoryReadDelegate<MyClass, &MyClass::readMemory>(myInstanceOfMyClass);
builder->addMemorySpace(...); // Uses readMemory
```

Template Parameters

<i>T</i>	Class that defines a memory read delegate method.
<i>METHOD</i>	A method of class T which is a memory read delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.5.2.12 setDefaultMemoryWriteDelegate() [1/3]

```
template<IrisErrorCode(*) (const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) FUNC>
```

```
void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate ( ) [inline]
```

Set default write function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the access function using

```
addMemorySpace(...).setWriteDelegate(...)
```

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_not_implemented` for all requests.

Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

```
iris::IrisErrorCode writeMemory(const iris::MemorySpaceInfo &spaceInfo, uint64_t address,
                               uint64_t byteWidth, uint64_t count,
                               const iris::IrisValueMap &attrib,
                               const uint64_t *data,
                               iris::MemoryWriteResult &result);

iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultMemoryWriteDelegate(&writeMemory);
builder->addMemorySpace(...); // Uses writeMemory
```

Template Parameters

<i>FUNC</i>	Global function to call to write memory.
-------------	--

7.5.2.13 setDefaultMemoryWriteDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate (
    MemoryWriteDelegate delegate = MemoryWriteDelegate() ) [inline]
```

Set the default write function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the access function using

```
addMemorySpace(...).setWriteDelegate(...)
```

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_not_implemented` for all requests.

Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

```
class MyClass
{
    ...
    iris::IrisErrorCode writeMemory(const iris::MemorySpaceInfo &spaceInfo, uint64_t address,
                                    uint64_t byteWidth, uint64_t count,
                                    const iris::IrisValueMap &attrib,
                                    const uint64_t *data,
                                    iris::MemoryWriteResult &result);
};

MyClass myInstanceOfMyClass;
iris::MemoryReadDelegate delegate =
    iris::MemoryWriteDelegate::make<MyClass, &MyClass::writeMemory>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultMemoryWriteDelegate(delegate);
builder->addMemorySpace(...); // Uses writeMemory
```

Parameters

<i>delegate</i>	Delegate object which will be called to write memory.
-----------------	---

7.5.2.14 setDefaultMemoryWriteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t,
    _t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>
```

```
void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate (
    T * instance ) [inline]
```

Set the default write function for all subsequently added memory spaces.

Memory spaces that do not explicitly override the access function using

```
addMemorySpace(...).setWriteDelegate(...)
```

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↔` not_implemented for all requests.

Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

```
class MyClass
{
    ...
    iris::IrisErrorCode writeMemory(const iris::MemorySpaceInfo &spaceInfo, uint64_t address,
                                    uint64_t byteWidth, uint64_t count,
                                    const iris::IrisValueMap &attrib,
                                    const uint64_t *data,
                                    iris::MemoryWriteResult &result);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultMemoryWriteDelegate<MyClass, &MyClass::writeMemory>(&myInstanceOfMyClass);
builder->addMemorySpace(...); // Uses writeMemory
```

Template Parameters

<i>T</i>	Class that defines a memory read delegate method.
<i>METHOD</i>	A method of class T which is a memory read delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.5.2.15 setPropertyCanonicalMsnScheme()

```
void iris::IrisInstanceBuilder::setPropertyCanonicalMsnScheme (
    const std::string & canonicalMsnScheme )
```

Set the memory.canonicalMsnScheme instance property.

This property is visible in the list of properties returned by `instance_getProperties()`.

This property defines the scheme used by the 'canonicalMsn' member of the MemorySpaceInfo object. The default is 'arm.com/memoriespaces' which is used by all Arm components. This default can be overridden by calling this function. This should be called upon initialisation, before other instances have a chance to call `instance_get↔Properties()`.

Parameters

<i>canonicalMsnScheme</i>	Name of the canonical memory space number scheme used by this instance.
---------------------------	---

7.6 IrisInstanceBuilder image loading APIs

Set up image-loading delegates.

Functions

- `template<IrisErrorCode(*)>(const std::vector< uint8_t > &) FUNC<`
`void iris::IrisInstanceBuilder::setLoadImageDataDelegate ()`
Set the delegate to load an image from the data provided.
- `void iris::IrisInstanceBuilder::setLoadImageDataDelegate (ImageLoadDataDelegate delegate=ImageLoadDataDelegate())`

Set the delegate to load an image from the data provided.

- `template<typename T, IrisErrorCode(T::*)(const std::vector< uint8_t > &) METHOD>`
void `iris::IrisInstanceBuilder::setLoadImageDataDelegate` (T *instance)

Set the delegate to load an image from the data provided.

- `template<IrisErrorCode(*)(const std::string &) FUNC>`
void `iris::IrisInstanceBuilder::setLoadImageFileDelegate` ()

Set the delegate to load an image from a file.

- void `iris::IrisInstanceBuilder::setLoadImageFileDelegate` (ImageLoadFileDelegate delegate=ImageLoadFileDelegate())

Set the delegate to load an image from a file.

- `template<typename T, IrisErrorCode(T::*)(const std::string &) METHOD>`
void `iris::IrisInstanceBuilder::setLoadImageFileDelegate` (T *instance)

Set the delegate to load an image from a file.

7.6.1 Detailed Description

Set up image-loading delegates.

7.6.2 Function Documentation

7.6.2.1 setLoadImageDataDelegate() [1/3]

```
template<IrisErrorCode(*) (const std::vector< uint8_t > &) FUNC>
void iris::IrisInstanceBuilder::setLoadImageDataDelegate ( ) [inline]
```

Set the delegate to load an image from the data provided.

Usage:

```
iris::IrisErrorCode loadImageData(const std::vector<uint64_t> &data, uint64_t dataSizeInBytes);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setLoadImageDataDelegate(&loadImageData);
```

Template Parameters

<i>FUNC</i>	Global function to call for image loading.
--------------------	--

7.6.2.2 setLoadImageDataDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setLoadImageDataDelegate (
    ImageLoadDataDelegate delegate = ImageLoadDataDelegate() ) [inline]
```

Set the delegate to load an image from the data provided.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↵` not_implemented for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode loadImageData(const std::vector<uint64_t> &data, uint64_t dataSizeInBytes);
};
MyClass myInstanceOfMyClass;
iris::MemoryAddressTranslateDelegate delegate =
    iris::MemoryAddressTranslateDelegate::make<MyClass, &MyClass::loadImageData>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setLoadImageDataDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object to call for image loading.
------------------------	--

7.6.2.3 setLoadImageDataDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const std::vector< uint8_t > &) METHOD>
void iris::IrisInstanceBuilder::setLoadImageDataDelegate (
    T * instance ) [inline]
```

Set the delegate to load an image from the data provided.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode loadImageData(const std::vector<uint64_t> &data, uint64_t dataSizeInBytes);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setLoadImageDataDelegate<MyClass, &MyClass::loadImageData>(&myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines an image-loading delegate method.
<i>METHOD</i>	A method of class T which is an image-loading delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.6.2.4 setLoadImageFileDelegate() [1/3]

```
template<IrisErrorCode(*) (const std::string &) FUNC>
void iris::IrisInstanceBuilder::setLoadImageFileDelegate ( ) [inline]
```

Set the delegate to load an image from a file.

Usage:

```
iris::IrisErrorCode loadImageFile(const std::string &path);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setLoadImageFileDelegate<&loadImageFile>();
```

Template Parameters

<i>FUNC</i>	Global function to call for image loading.
-------------	--

7.6.2.5 setLoadImageFileDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setLoadImageFileDelegate (
    ImageLoadFileDelegate delegate = ImageLoadFileDelegate() ) [inline]
```

Set the delegate to load an image from a file.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↔` not_implemented for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode loadImageFile(const std::string &path);
};
MyClass myInstanceOfMyClass;
iris::MemoryAddressTranslateDelegate delegate =
    iris::MemoryAddressTranslateDelegate::make<MyClass, &MyClass::loadImageFile>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setLoadImageFileDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object to call for image loading.
-----------------	--

7.6.2.6 setLoadImageFileDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const std::string &) METHOD>
void iris::IrisInstanceBuilder::setLoadImageFileDelegate (
    T * instance ) [inline]
```

Set the delegate to load an image from a file.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode loadImageFile(const std::string &path);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setLoadImageFileDelegate<MyClass, &MyClass::loadImageFile>(&myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines an image-loading delegate method.
<i>METHOD</i>	A method of class T which is an image-loading delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.7 IrisInstanceBuilder image readData callback APIs.

Open images for reading.

Functions

- uint64_t [iris::IrisInstanceBuilder::openImage](#) (const std::string &filename)
Open an image to be read using [image_loadDataPull\(\)](#) or [image_loadDataRead\(\)](#).

7.7.1 Detailed Description

Open images for reading.

7.7.2 Function Documentation

7.7.2.1 openImage()

```
uint64_t iris::IrisInstanceBuilder::openImage (
    const std::string & filename ) [inline]
```

Open an image to be read using [image_loadDataPull\(\)](#) or [image_loadDataRead\(\)](#).

Parameters

<i>filename</i>	The name of the file to be read.
-----------------	----------------------------------

Returns

The tag number to use when calling `image_loadDataPull()`.

7.8 IrisInstanceBuilder execution stepping APIs

Set up delegates to set and get the step count and the remaining steps.

Functions

- `template<IrisErrorCode(*) (uint64_t &, const std::string &) FUNC>`
`void iris::IrisInstanceBuilder::setRemainingStepGetDelegate ()`
Set the delegate to get the remaining steps for this instance.
- `void iris::IrisInstanceBuilder::setRemainingStepGetDelegate (RemainingStepGetDelegate delegate)`
Set the delegate to get the remaining steps for this instance.
- `template<typename T, IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>`
`void iris::IrisInstanceBuilder::setRemainingStepGetDelegate (T *instance)`
Set the delegate to get the remaining steps for this instance.
- `template<IrisErrorCode(*) (uint64_t &, const std::string &) FUNC>`
`void iris::IrisInstanceBuilder::setRemainingStepSetDelegate ()`
Set the delegate to set the remaining steps for this instance.
- `void iris::IrisInstanceBuilder::setRemainingStepSetDelegate (RemainingStepSetDelegate delegate=RemainingStepSetDelegate())`
Set the delegate to set the remaining steps for this instance.
- `template<typename T, IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>`
`void iris::IrisInstanceBuilder::setRemainingStepSetDelegate (T *instance)`
Set the delegate to set the remaining steps for this instance.
- `template<IrisErrorCode(*) (uint64_t &, const std::string &) FUNC>`
`void iris::IrisInstanceBuilder::setStepCountGetDelegate ()`
Set the delegate to get the step count for this instance.
- `void iris::IrisInstanceBuilder::setStepCountGetDelegate (StepCountGetDelegate delegate=StepCountGetDelegate())`
Set the delegate to get the step count for this instance.
- `template<typename T, IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>`
`void iris::IrisInstanceBuilder::setStepCountGetDelegate (T *instance)`
Set the delegate to get the step count for this instance.

7.8.1 Detailed Description

Set up delegates to set and get the step count and the remaining steps.

7.8.2 Function Documentation

7.8.2.1 setRemainingStepGetDelegate() [1/3]

```
template<IrisErrorCode(*) (uint64_t &, const std::string &) FUNC>
void iris::IrisInstanceBuilder::setRemainingStepGetDelegate () [inline]
```

Set the delegate to get the remaining steps for this instance.

Usage:

```
iris::IrisErrorCode getRemainingSteps(uint64_t &steps, const std::string &unit);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setRemainingStepGetDelegate(&getRemainingSteps());
```

Template Parameters

<i><code>FUNC</code></i>	Global function to call to get the remaining steps.
---------------------------------	---

7.8.2.2 setRemainingStepGetDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setRemainingStepGetDelegate (
    RemainingStepGetDelegate delegate ) [inline]
```

Set the delegate to get the remaining steps for this instance.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↔ not_implemented` for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getRemainingSteps(uint64_t &steps, const std::string &unit);
};
MyClass myInstanceOfMyClass;
iris::RemainingStepGetDelegate::make<MyClass, &MyClass::getRemainingSteps>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setRemainingStepGetDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object to call to get the remaining steps.
-----------------	---

7.8.2.3 setRemainingStepGetDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>
void iris::IrisInstanceBuilder::setRemainingStepGetDelegate (
    T * instance ) [inline]
```

Set the delegate to get the remaining steps for this instance.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getRemainingSteps(uint64_t &steps, const std::string &unit);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setRemainingStepGetDelegate<MyClass, &MyClass::getRemainingSteps>(&myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines a get remaining steps delegate method.
<i>METHOD</i>	A method of class T that is a get remaining steps delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.8.2.4 setRemainingStepSetDelegate() [1/3]

```
template<IrisErrorCode(*) (uint64_t, const std::string &) FUNC>
void iris::IrisInstanceBuilder::setRemainingStepSetDelegate ( ) [inline]
```

Set the delegate to set the remaining steps for this instance.

Usage:

```
iris::IrisErrorCode setRemainingSteps(uint64_t steps, const std::string &unit);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setRemainingStepSetDelegate<&setRemainingSteps>();
```

Template Parameters

<i>FUNC</i>	Global function to call to set the remaining steps.
-------------	---

7.8.2.5 setRemainingStepSetDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setRemainingStepSetDelegate (
    RemainingStepSetDelegate delegate = RemainingStepSetDelegate() ) [inline]
```

Set the delegate to set the remaining steps for this instance.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↔ not_implemented` for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setRemainingSteps(uint64_t steps, const std::string &unit);
};
MyClass myInstanceOfMyClass;
iris::RemainingStepSetDelegate delegate =
    iris::RemainingStepSetDelegate::make<MyClass, &MyClass::setRemainingSteps>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setRemainingStepSetDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object to call to set the remaining steps.
-----------------	---

7.8.2.6 setRemainingStepSetDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(uint64_t, const std::string &) METHOD>
void iris::IrisInstanceBuilder::setRemainingStepSetDelegate (
    T * instance ) [inline]
```

Set the delegate to set the remaining steps for this instance.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setRemainingSteps(uint64_t steps, const std::string &unit);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setRemainingStepSetDelegate<MyClass, &MyClass::setRemainingSteps>(&myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines a set remaining steps delegate method.
<i>METHOD</i>	A method of class T that is a set remaining steps delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.8.2.7 setStepCountGetDelegate() [1/3]

```
template<IrisErrorCode(*) (uint64_t &, const std::string &) FUNC>
void iris::IrisInstanceBuilder::setStepCountGetDelegate ( ) [inline]
```

Set the delegate to get the step count for this instance.

Usage:

```
iris::IrisErrorCode getStepCount(uint64_t &count, const std::string &unit);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setStepCountGetDelegate(&getStepCount>();
```

Template Parameters

<i>FUNC</i>	Global function to call to get the step count.
-------------	--

7.8.2.8 setStepCountGetDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setStepCountGetDelegate (
    StepCountGetDelegate delegate = StepCountGetDelegate() ) [inline]
```

Set the delegate to get the step count for this instance.

Passing an empty delegate (the default argument) restores the default implementation which always returns E_↔ not_implemented for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getStepCount(uint64_t &count, const std::string &unit);
};
MyClass myInstanceOfMyClass;
iris::StepCountGetDelegate delegate =
    iris::StepCountGetDelegate::make<MyClass, &MyClass::getStepCount>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setStepCountGetDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object to call to get the step count.
-----------------	--

7.8.2.9 setStepCountGetDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>
void iris::IrisInstanceBuilder::setStepCountGetDelegate (
    T * instance ) [inline]
```

Set the delegate to get the step count for this instance.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getStepCount(uint64_t &count, const std::string &unit);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setStepCountGetDelegate<MyClass, &MyClass::getStepCount>(&myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines a get step count delegate method.
<i>METHOD</i>	A method of class T which is a get step count delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

7.9 Disassembler delegate functions

Set disassembler delegates.

Classes

- class [iris::IrisInstanceDisassembler](#)
Disassembler add-on for [IrisInstance](#).

Typedefs

- typedef [IrisDelegate](#)< const std::vector< uint64_t > &, uint64_t, const std::string &, DisassembleContext &, DisassemblyLine & > [iris::DisassembleOpcodeDelegate](#)
Get the disassembly for an individual opcode.
- typedef [IrisDelegate](#)< std::string & > [iris::GetCurrentDisassemblyModeDelegate](#)
Get the current disassembly mode.
- typedef [IrisDelegate](#)< uint64_t, const std::string &, MemoryReadResult &, uint64_t, uint64_t, std::vector< DisassemblyLine > & > [iris::GetDisassemblyDelegate](#)
Get the disassembly of a chunk of memory.

Functions

- void [iris::IrisInstanceDisassembler::addDisassemblyMode](#) (const std::string &name, const std::string &description)
Add a disassembly mode.
- void [iris::IrisInstanceDisassembler::attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
- [iris::IrisInstanceDisassembler::IrisInstanceDisassembler](#) ([IrisInstance](#) *irisInstance=nullptr)
Construct an [IrisInstanceDisassembler](#).
- void [iris::IrisInstanceDisassembler::setDisassembleOpcodeDelegate](#) ([DisassembleOpcodeDelegate](#) delegate)
Set the delegate to get the disassembly of Opcode.
- void [iris::IrisInstanceDisassembler::setCurrentModeDelegate](#) ([GetCurrentDisassemblyModeDelegate](#) delegate)
Set the delegate to get the current disassembly mode.
- void [iris::IrisInstanceDisassembler::setGetDisassemblyDelegate](#) ([GetDisassemblyDelegate](#) delegate)
Set the delegate to get the disassembly of a chunk of memory.

7.9.1 Detailed Description

Set disassembler delegates.

7.9.2 Typedef Documentation

7.9.2.1 DisassembleOpcodeDelegate

```
typedef IrisDelegate<const std::vector<uint64_t>&, uint64_t, const std::string&, DisassembleContext&, DisassemblyLine&> iris::DisassembleOpcodeDelegate
```

Get the disassembly for an individual opcode.

```
IrisErrorCode disassembleOpcode(const std::vector<uint64_t> &opcode, uint64_t address, const std::string &mode, DisassembleContext &context, DisassemblyLine &disassemblyLineOut)
```

Error: Return E_* error code if it failed to disassemble.

7.9.2.2 GetCurrentDisassemblyModeDelegate

```
typedef IrisDelegate<std::string> iris::GetCurrentDisassemblyModeDelegate
```

Get the current disassembly mode.

```
IrisErrorCode getCurrentMode(std::string &currentMode)
```

Error: Return E_* error code if it failed to get the current mode.

7.9.2.3 GetDisassemblyDelegate

```
typedef IrisDelegate<uint64_t, const std::string&, MemoryReadResult&, uint64_t, uint64_t,
std::vector<DisassemblyLine>&> iris::GetDisassemblyDelegate
```

Get the disassembly of a chunk of memory.

```
IrisErrorCode getDisassembly(uint64_t address, const std::string &mode, MemoryReadResult &memoryReadData,
uint64_t count, uint64_t maxAddr, std::vector<DisassemblyLine>
&disassemblyLineOut)
```

Error: Return E_* error code if it failed to disassemble.

7.9.3 Function Documentation

7.9.3.1 addDisassemblyMode()

```
void iris::IrisInstanceDisassembler::addDisassemblyMode (
    const std::string & name,
    const std::string & description )
```

Add a disassembly mode.

This function should only be called during the initial setup of the instance, after which the list of disassembly modes should be static.

Parameters

<i>name</i>	Name of the mode being added.
<i>description</i>	Description of the mode being added.

7.9.3.2 attachTo()

```
void iris::IrisInstanceDisassembler::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

Parameters

<i>irisInstance</i>	The IrisInstance to attach to.
---------------------	--

7.9.3.3 IrisInstanceDisassembler()

```
iris::IrisInstanceDisassembler::IrisInstanceDisassembler (
    IrisInstance * irisInstance = nullptr )
```

Construct an [IrisInstanceDisassembler](#).

Parameters

<i>irisInstance</i>	IrisInstance to attach this add-on to.
---------------------	--

7.9.3.4 setDisassembleOpcodeDelegate()

```
void iris::IrisInstanceDisassembler::setDisassembleOpcodeDelegate (
    DisassembleOpcodeDelegate delegate ) [inline]
```

Set the delegate to get the disassembly of Opcode.

Parameters

<i>delegate</i>	Delegate object that will be called to get the disassembly of an opcode.
-----------------	--

7.9.3.5 setCurrentModeDelegate()

```
void iris::IrisInstanceDisassembler::setCurrentModeDelegate (
    GetCurrentDisassemblyModeDelegate delegate ) [inline]
```

Set the delegate to get the current disassembly mode.

Parameters

<i>delegate</i>	Delegate object that will be called to get the current disassembly mode.
-----------------	--

7.9.3.6 setGetDisassemblyDelegate()

```
void iris::IrisInstanceDisassembler::setGetDisassemblyDelegate (
    GetDisassemblyDelegate delegate ) [inline]
```

Set the delegate to get the disassembly of a chunk of memory.

Parameters

<i>delegate</i>	Delegate object that will be called to get the disassembly of a chunk of memory.
-----------------	--

7.10 Semihosting data request flag constants

Flags used to define the behavior of the readData() method.

7.10.1 Detailed Description

Flags used to define the behavior of the readData() method.

Chapter 8

Class Documentation

8.1 iris::IrisInstanceBuilder::AddressTranslationBuilder Class Reference

Used to set metadata for an address translation.

```
#include <IrisInstanceBuilder.h>
```

Public Member Functions

- **AddressTranslationBuilder** ([IrisInstanceMemory::AddressTranslationInfoAndAccess](#) &info_)
- `template<IrisErrorCode(*) (uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) FUNC>`
[AddressTranslationBuilder](#) & [setTranslateDelegate](#) ()
Set the delegate to perform an address translation.
- [AddressTranslationBuilder](#) & [setTranslateDelegate](#) ([MemoryAddressTranslateDelegate](#) delegate)
Set the delegate to perform an address translation.
- `template<typename T, IrisErrorCode(T::*) (uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) METHOD>`
[AddressTranslationBuilder](#) & [setTranslateDelegate](#) (T *instance)
Set the delegate to perform an address translation.

8.1.1 Detailed Description

Used to set metadata for an address translation.

8.1.2 Member Function Documentation

8.1.2.1 setTranslateDelegate() [1/3]

```
template<IrisErrorCode(*) (uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &)
FUNC>
AddressTranslationBuilder & iris::IrisInstanceBuilder::AddressTranslationBuilder::setTranslate↵
Delegate ( ) [inline]
```

Set the delegate to perform an address translation.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultAddressTranslationDelegate](#)

Template Parameters

<i>FUNC</i>	An address translation delegate function.
-------------	---

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.1.2.2 setTranslateDelegate() [2/3]

```
AddressTranslationBuilder & iris::IrisInstanceBuilder::AddressTranslationBuilder::setTranslate↵
Delegate (
    MemoryAddressTranslateDelegate delegate ) [inline]
```

Set the delegate to perform an address translation.
If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultAddressTranslationDelegate](#)

Parameters

<i>delegate</i>	MemoryAddressTranslateDelegate object.
-----------------	--

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.1.2.3 setTranslateDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslation↵
Result &) METHOD>
AddressTranslationBuilder & iris::IrisInstanceBuilder::AddressTranslationBuilder::setTranslate↵
Delegate (
    T * instance ) [inline]
```

Set the delegate to perform an address translation.
If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultAddressTranslationDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a memory address translation delegate.
<i>METHOD</i>	A memory address translation delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

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

- [IrisInstanceBuilder.h](#)

8.2 iris::IrisInstanceMemory::AddressTranslationInfoAndAccess Struct Reference

Contains static address translation information.

```
#include <IrisInstanceMemory.h>
```

Public Member Functions

- **AddressTranslationInfoAndAccess** (MemorySpaceld inSpaceld, MemorySpaceld outSpaceld, const std::string &description)

Public Attributes

- [MemoryAddressTranslateDelegate](#) **translateDelegate**
- MemorySupportedAddressTranslationResult **translationInfo**

8.2.1 Detailed Description

Contains static address translation information.

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

- [IrisInstanceMemory.h](#)

8.3 iris::BreakpointHitInfo Struct Reference

Public Attributes

- const std::vector< uint64_t > & **accessData**
- const BreakpointInfo & **bptInfo**
- bool **isReadAccess**

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

- [IrisInstanceBreakpoint.h](#)

8.4 iris::IrisInstanceBuilder::EventSourceBuilder Class Reference

Used to set metadata on an EventSource.

```
#include <IrisInstanceBuilder.h>
```

Public Member Functions

- [EventSourceBuilder](#) & [addEnumElement](#) (const std::string &fieldName, uint64_t value, const std::string &symbol, const std::string &description="")
Add an enum element to a specific field.
- [EventSourceBuilder](#) & [addEnumElement](#) (uint64_t value, const std::string &symbol, const std::string &description="")
Add an enum element for the last field added.
- [EventSourceBuilder](#) & [addField](#) (const std::string &name, const std::string &type, uint64_t sizeInBytes, const std::string &description)
Add a field to this event source.
- template<typename T >
[EventSourceBuilder](#) & [addOption](#) (const std::string &name, const std::string &type, const T &defaultValue, bool optional, const std::string &description)
Declare an option for event streams of an event source.
- **EventSourceBuilder** ([IrisInstanceEvent::EventSourceInfoAndDelegate](#) &info_)

- [EventSourceBuilder](#) & [hasSideEffects](#) (bool hasSideEffects__{__}=true)
Set hasSideEffects for this event source.
- [EventSourceBuilder](#) & [removeEnumElement](#) (const std::string &fieldName, uint64_t value)
Remove an enum element by value from a specific field.
- [EventSourceBuilder](#) & [renameEnumElement](#) (const std::string &fieldName, uint64_t value, const std::string &newEnumSymbol)
Rename an enum element by value of a specific field.
- [EventSourceBuilder](#) & [setCounter](#) (bool counter=true)
*Set the *counter* field.*
- [EventSourceBuilder](#) & [setDescription](#) (const std::string &description)
*Set the *description* field.*
- [EventSourceBuilder](#) & [setEventStreamCreateDelegate](#) (EventStreamCreateDelegate delegate)
Set the delegate to create an event stream.
- template<typename T, IrisErrorCode(T::*)(EventStream *&, const EventSourceInfo &, const std::vector< std::string > &) METHOD>
[EventSourceBuilder](#) & [setEventStreamCreateDelegate](#) (T *instance)
Set the delegate to create an event stream.
- [EventSourceBuilder](#) & [setFormat](#) (const std::string &format)
*Set the *format* field.*
- [EventSourceBuilder](#) & [setHidden](#) (bool hidden=true)
Hide/unhide this event source.
- [EventSourceBuilder](#) & [setName](#) (const std::string &name)
*Set the *name* field.*

8.4.1 Detailed Description

Used to set metadata on an EventSource.

8.4.2 Member Function Documentation

8.4.2.1 addEnumElement() [1/2]

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::addEnumElement (
    const std::string & fieldName,
    uint64_t value,
    const std::string & symbol,
    const std::string & description = "" ) [inline]
```

Add an enum element to a specific field.

Parameters

<i>fieldName</i>	Field name.
<i>value</i>	The value of the enum element.
<i>symbol</i>	The symbol string that will be displayed instead of the value.
<i>description</i>	A human readable description of this enum.

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.2 addEnumElement() [2/2]

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::addEnumElement (
```

```
uint64_t value,
const std::string & symbol,
const std::string & description = "" ) [inline]
```

Add an enum element for the last field added.

This must be called after [addField\(\)](#).

Parameters

<i>value</i>	The value of the enum element.
<i>symbol</i>	The symbol string that will be displayed instead of the value.
<i>description</i>	A human readable description of this enum.

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.3 addField()

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::addField (
    const std::string & name,
    const std::string & type,
    uint64_t sizeInBytes,
    const std::string & description ) [inline]
```

Add a field to this event source.

This method constructs an EventSourceFieldInfo object and adds it to the EventSource. It should be called multiple times to add multiple fields.

Parameters

<i>name</i>	The name of the field.
<i>type</i>	The type of the field.
<i>sizeInBytes</i>	The size of the field in bytes.
<i>description</i>	A human readable description of the field.

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.4 addOption()

```
template<typename T >
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::addOption (
    const std::string & name,
    const std::string & type,
    const T & defaultValue,
    bool optional,
    const std::string & description ) [inline]
```

Declare an option for event streams of an event source.

This method fills the 'options' member of EventSourceInfo. It may be called multiple times to add multiple options.

Parameters

<i>name</i>	The name of the option.
<i>type</i>	The type of the option.

Parameters

<i>defaultValue</i>	The default value of the option.
<i>optional</i>	True if the option is optional, False otherwise.
<i>description</i>	A human readable description of the option.

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.5 hasSideEffects()

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::hasSideEffects (
    bool hasSideEffects_ = true ) [inline]
```

Set hasSideEffects for this event source.

Parameters

<i>hasSideEffects_</i>	If true, this event source has side effects. This is exotic. Normal event sources do not have side effects. For example semihosting events have side effects.
------------------------	---

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.6 removeEnumElement()

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::removeEnumElement (
    const std::string & fieldName,
    uint64_t value ) [inline]
```

Remove an enum element by value from a specific field.

Parameters

<i>fieldName</i>	Field name.
<i>value</i>	The value of the enum element.

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.7 renameEnumElement()

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::renameEnumElement (
    const std::string & fieldName,
    uint64_t value,
    const std::string & newEnumSymbol ) [inline]
```

Rename an enum element by value of a specific field.

Parameters

<i>fieldName</i>	Field name.
------------------	-------------

Parameters

<i>value</i>	The value of the enum element.
<i>newEnumSymbol</i>	New enum symbol.

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.8 setCounter()

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::setCounter (
    bool counter = true ) [inline]
```

Set the `counter` field.

Parameters

<i>counter</i>	The counter field of the EventSourceInfo object.
----------------	--

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.9 setDescription()

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::setDescription (
    const std::string & description ) [inline]
```

Set the `description` field.

Parameters

<i>description</i>	The description field of the EventSourceInfo object.
--------------------	--

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.10 setEventStreamCreateDelegate() [1/2]

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::setEventStreamCreate←
Delegate (
    EventStreamCreateDelegate delegate ) [inline]
```

Set the delegate to create an event stream.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultEsCreateDelegate](#)

Parameters

<i>delegate</i>	EventStreamCreateDelegate object.
-----------------	-----------------------------------

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.11 setEventStreamCreateDelegate() [2/2]

```
template<typename T , IrisErrorCode(T::*)(EventStream * &, const EventSourceInfo &, const std::vector< std::string > &) METHOD>
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::setEventStreamCreateDelegate (
    T * instance ) [inline]
```

Set the delegate to create an event stream.
If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultEsCreateDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be an event stream creation method.
<i>METHOD</i>	An event stream creation delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.12 setFormat()

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::setFormat (
    const std::string & format ) [inline]
```

Set the `format` field.

Parameters

<i>format</i>	The format field of the EventSourceInfo object.
---------------	---

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.13 setHidden()

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::setHidden (
    bool hidden = true ) [inline]
```

Hide/unhide this event source.

Parameters

<i>hidden</i>	If true, this event source is not listed in <code>event_getEventSources()</code> calls but can still be accessed by <code>event_getEventSource()</code> for clients that know the event source's name.
---------------	--

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

8.4.2.14 setName()

```
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::setName (
    const std::string & name ) [inline]
```

Set the name field.

Parameters

<i>name</i>	The name field of the EventSourceInfo object.
-------------	---

Returns

A reference to this [EventSourceBuilder](#) object allowing calls to be chained together.

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

- [IrisInstanceBuilder.h](#)

8.5 iris::IrisInstanceEvent::EventSourceInfoAndDelegate Struct Reference

Contains the metadata and delegates for a single EventSource.

```
#include <IrisInstanceEvent.h>
```

Public Attributes

- [EventStreamCreateDelegate](#) **createEventStream**
- EventSourceInfo **info**
- bool **isProxy** {false}
- bool **isValid** {true}
- [ProxyEventInfo](#) **proxyEventInfo**

8.5.1 Detailed Description

Contains the metadata and delegates for a single EventSource.

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

- [IrisInstanceEvent.h](#)

8.6 iris::EventStream Class Reference

Base class for event streams.

```
#include <IrisInstanceEvent.h>
```

Inherited by [iris::IrisEventStream](#).

Public Member Functions

- virtual IrisErrorCode [action](#) (const BreakpointAction &action_)
Execute action on trace stream.
- void [addField](#) (const IrisU64StringConstant &field, bool value)
Add a boolean field value.
- template<class T >
void [addField](#) (const IrisU64StringConstant &field, const T &value)
Add a field value.
- void [addField](#) (const IrisU64StringConstant &field, const uint8_t *data, size_t sizeInBytes)
Add byte array.
- void [addField](#) (const IrisU64StringConstant &field, int64_t value)
Add a sint field value.
- void [addField](#) (const IrisU64StringConstant &field, uint64_t value)
Add a uint field value.
- void [addFieldSlow](#) (const std::string &field, bool value)
Add a boolean field value.
- template<class T >
void [addFieldSlow](#) (const std::string &field, const T &value)
Add a field value.
- void [addFieldSlow](#) (const std::string &field, const uint8_t *data, size_t sizeInBytes)
Add byte array.
- void [addFieldSlow](#) (const std::string &field, int64_t value)
Add a sint field value.
- void [addFieldSlow](#) (const std::string &field, uint64_t value)
Add a uint field value.
- bool [checkRangePc](#) (uint64_t pc) const
Check the range for the PC.
- virtual IrisErrorCode [disable](#) ()=0
Disable this event stream.
- void [emitEventBegin](#) (IrisRequest &req, uint64_t time, uint64_t pc=IRIS_UINT64_MAX)
Start to emit an event callback.
- void [emitEventBegin](#) (uint64_t time, uint64_t pc=IRIS_UINT64_MAX)
Start to emit an event callback.
- void [emitEventEnd](#) (bool send=true)
Emit the callback.
- virtual IrisErrorCode [enable](#) ()=0
Enable this event stream.
- **EventStream** ()
Construct a new event stream.
- virtual IrisErrorCode [flush](#) (RequestId requestId)
Flush event stream.
- uint64_t [getCountVal](#) () const
Get the current value of the counter.
- InstanceId [getEclnstd](#) () const
Get the event callback instance id for this event stream.
- EventStreamId [getEsId](#) () const
Get the Id of this event stream.
- EventSourceId [getEventSourceId](#) () const
Get the event source id of the event source of this event stream (not the event stream id)
- const EventSourceInfo * [getEventSourceInfo](#) () const

- Get the event source info of this event stream.*

 - Instanceld `getProxiedByInstanceld` () const

Get the instance ID of the Iris instance which is a proxy for this event stream.
- virtual IrisErrorCode `getState` (IrisValueMap &fields)

Query the current state of the event.
- virtual IrisErrorCode `insertTrigger` ()
- bool `isCounter` () const

Is this event stream a counter?
- bool `isEnabled` () const

Is this event stream currently enabled?
- bool `IsProxiedByOtherInstance` () const

Is there another Iris instance which is a proxy for this event stream?
- bool `IsProxyForOtherInstance` () const

Is this event stream a proxy for an event stream in another Iris instance?
- void `selfRelease` ()

Trigger the event stream to be released.
- void `setCounter` (uint64_t startVal, const EventCounterMode &counterMode)

Set the counter mode and starting value for this event stream.
- virtual IrisErrorCode `setOptions` (const AttributeValueMap &options, bool eventStreamCreate, std::string &errorMessageOut)

Set options.
- void `setProperties` (IrisInstance *irisInstance, IrisInstanceEvent *irisInstanceEvent, EventSourceId evSrcId, Instanceld ecInstId, const std::string &ecFunc, EventStreamId esId, bool syncEc)

Initialize this event stream.
- void `setProxiedByInstanceld` (Instanceld instId)

Saves the instance ID of the Iris instance that is a proxy for this event stream.
- void `setProxyForOtherInstance` ()

Set that this event stream is a proxy for an event stream in another Iris instance.
- IrisErrorCode `setRanges` (const std::string &aspect, const std::vector< uint64_t > &ranges)

Set the trace ranges for this event stream.

Protected Attributes

- std::string **aspect**
 - members for range —
- bool **aspectFound** {}
 - Found aspect in one of the fields.*
- bool **counter** {}
 - members for a counter —
- EventCounterMode **counterMode** {}
 - Specified counter mode.*
- uint64_t **curAspectValue** {}
 - The current aspect value.*
- uint64_t **curVal** {}
- std::string **ecFunc**
 - The event callback function name specified by eventEnable().*
- Instanceld **ecInstId** {IRIS_UINT64_MAX}
 - Specify target instance that this event is sent to.*
- bool **enabled** {}
 - Event is only generated when the event stream is enabled.*
- EventStreamId **esId** {IRIS_UINT64_MAX}

- The event stream id.*
- EventSourceId **evSrcId** {IRIS_UINT64_MAX}
- The event source of this stream.*
- IrisU64JsonWriter::Object **fieldObj**
- IrisRequest * **internal_req** {}
- [IrisInstance](#) * **irisInstance** {}
- basic members —
- [IrisInstanceEvent](#) * **irisInstanceEvent** {}
- Parent [IrisInstanceEvent](#) owning this stream.*
- bool **isProxyForOtherInstance** {false}
- Is this event stream a proxy for an event stream in another Iris instance?*
- InstanceId [proxiedByInstanceld](#) {IRIS_UINT64_MAX}
- std::vector< uint64_t > **ranges**
- IrisRequest * **req** {}
- Generate callback requests.*
- uint64_t **startVal** {}
- Start value and current value for a counter.*
- bool **syncEc** {}
- Synchronous callback behavior.*

8.6.1 Detailed Description

Base class for event streams.

This class is abstract as it is not known how to enable or disable an event for a simulation.

8.6.2 Member Function Documentation

8.6.2.1 action()

```
virtual IrisErrorCode iris::EventStream::action (
    const BreakpointAction & action_ ) [inline], [virtual]
```

Execute action on trace stream.

This function is usually only ever called by breakpoints which have an action other than eventStream_enable or eventStream_disable.

This function is only implemented by very specific event streams.

Returns

An error code indicating whether the operation was successful.

8.6.2.2 addField() [1/5]

```
void iris::EventStream::addField (
    const IrisU64StringConstant & field,
    bool value ) [inline]
```

Add a boolean field value.

Fast variant for argument names up to 23 chars. Use this if you can. This will also record the aspect value if the aspect of range check is set.

Parameters

<i>field</i>	The name of the field whose value is set.
<i>value</i>	The value of the field.

8.6.2.3 addField() [2/5]

```
template<class T >
void iris::EventStream::addField (
    const IrisU64StringConstant & field,
    const T & value ) [inline]
```

Add a field value.

This is supported for all types supported by IrisU64JsonWriter and IrisObjects.h. Fast variant for argument names up to 23 chars. Use this if you can.

Parameters

<i>field</i>	The name of the field whose value is set.
<i>value</i>	The value of the field.

8.6.2.4 addField() [3/5]

```
void iris::EventStream::addField (
    const IrisU64StringConstant & field,
    const uint8_t * data,
    size_t sizeInBytes ) [inline]
```

Add byte array.

Fast variant for argument names up to 23 chars. Use this if you can.

Parameters

<i>field</i>	The name of the field whose value is set.
<i>data</i>	Pointer to byte data.
<i>sizeInBytes</i>	Size of byte data.

8.6.2.5 addField() [4/5]

```
void iris::EventStream::addField (
    const IrisU64StringConstant & field,
    int64_t value ) [inline]
```

Add a sint field value.

Fast variant for argument names up to 23 chars. Use this if you can. This will also record the aspect value if the aspect of range check is set.

Parameters

<i>field</i>	The name of the field whose value is set.
<i>value</i>	The value of the field.

8.6.2.6 addField() [5/5]

```
void iris::EventStream::addField (
    const IrisU64StringConstant & field,
    uint64_t value ) [inline]
```

Add a uint field value.

Fast variant for argument names up to 23 chars. Use this if you can. This will also record the aspect value if the aspect of range check is set.

Parameters

<i>field</i>	The name of the field whose value is set.
<i>value</i>	The value of the field.

8.6.2.7 addFieldSlow() [1/5]

```
void iris::EventStream::addFieldSlow (
    const std::string & field,
    bool value ) [inline]
```

Add a boolean field value.

Slow variant for argument names with more than 23 chars. Do not use unless you have to. This will also record the aspect value if the aspect of range check is set.

Parameters

<i>field</i>	The name of the field whose value is set.
<i>value</i>	The value of the field.

8.6.2.8 addFieldSlow() [2/5]

```
template<class T >
void iris::EventStream::addFieldSlow (
    const std::string & field,
    const T & value ) [inline]
```

Add a field value.

This is supported for all types supported by IrisU64JsonWriter and IrisObjects.h. Slow variant for argument names with more than 23 chars. Do not use unless you have to.

Parameters

<i>field</i>	The name of the field whose value is set.
<i>value</i>	The value of the field.

8.6.2.9 addFieldSlow() [3/5]

```
void iris::EventStream::addFieldSlow (
    const std::string & field,
    const uint8_t * data,
    size_t sizeInBytes ) [inline]
```

Add byte array.

Slow variant for argument names with more than 23 chars. Do not use unless you have to.

Parameters

<i>field</i>	The name of the field whose value is set.
<i>data</i>	Pointer to byte data.
<i>sizeInBytes</i>	Size of byte data.

8.6.2.10 addFieldSlow() [4/5]

```
void iris::EventStream::addFieldSlow (
    const std::string & field,
    int64_t value ) [inline]
```

Add a sint field value.

Slow variant for argument names with more than 23 chars. Do not use unless you have to. This will also record the aspect value if the aspect of range check is set.

Parameters

<i>field</i>	The name of the field whose value is set.
<i>value</i>	The value of the field.

8.6.2.11 addFieldSlow() [5/5]

```
void iris::EventStream::addFieldSlow (
    const std::string & field,
    uint64_t value ) [inline]
```

Add a uint field value.

Slow variant for argument names with more than 23 chars. Do not use unless you have to. This will also record the aspect value if the aspect of range check is set.

Parameters

<i>field</i>	The name of the field whose value is set.
<i>value</i>	The value of the field.

8.6.2.12 checkRangePc()

```
bool iris::EventStream::checkRangePc (
    uint64_t pc ) const [inline]
```

Check the range for the PC.

This can optionally be called before generating the callback request (before calling [emitEventBegin\(\)](#)).

Parameters

<i>pc</i>	The program counter value to check.
-----------	-------------------------------------

Returns

`true` if the PC value is in range or no range is configured, `false` otherwise.

8.6.2.13 disable()

```
virtual IrisErrorCode iris::EventStream::disable ( ) [pure virtual]
```

Disable this event stream.

This function is only called when [isEnabled\(\)](#)/enabled == true. It is not necessary to verify this inside the [disable\(\)](#) method.

Returns

An error code indicating whether the event stream was successfully disabled. This should be `E_ok` if it was disabled or `E_error_disabling_event_stream` if it could not be disabled.

Implemented in [iris::IrisEventStream](#).

8.6.2.14 emitEventBegin() [1/2]

```
void iris::EventStream::emitEventBegin (
    IrisRequest & req,
    uint64_t time,
    uint64_t pc = IRIS_UINT64_MAX )
```

Start to emit an event callback.

Parameters

<i>req</i>	A request object to use to construct the event callback.
<i>time</i>	The time in simulation ticks at which the event occurred.
<i>pc</i>	The program counter value when the event occurred.

8.6.2.15 emitEventBegin() [2/2]

```
void iris::EventStream::emitEventBegin (
    uint64_t time,
    uint64_t pc = IRIS_UINT64_MAX )
```

Start to emit an event callback.

Parameters

<i>time</i>	The time in simulation ticks at which the event occurred.
<i>pc</i>	The program counter value when the event occurred.

8.6.2.16 emitEventEnd()

```
void iris::EventStream::emitEventEnd (
    bool send = true )
```

Emit the callback.

This will also check the ranges and maintain the counter.

Parameters

<i>send</i>	If <code>true</code> , event callbacks are sent to the callee immediately. If <code>false</code> , the callback are not sent immediately, allowing the caller to delay sending.
-------------	---

8.6.2.17 enable()

```
virtual IrisErrorCode iris::EventStream::enable ( ) [pure virtual]
```

Enable this event stream.

This function is only called when [isEnabled\(\)](#)/enabled == false. It is not necessary to verify this inside the [enable\(\)](#) method.

Returns

An error code indicating whether the event stream was successfully enabled. This should be E_ok if it was enabled or E_error_enabling_event_stream if it could not be enabled.

Implemented in [iris::IrisEventStream](#).

8.6.2.18 flush()

```
virtual IrisErrorCode iris::EventStream::flush (
    RequestId requestId ) [inline], [virtual]
```

Flush event stream.

Supported in the derived classes for specific event sources.

Parameters

<i>requestId</i>	Request id of the eventStream_flush() call. This is returned to the caller in an extra FLUSH_REQUEST_ID field in the response to the flush call.
------------------	--

Returns

An error code indicating whether the operation was successful.

8.6.2.19 getCountVal()

```
uint64_t iris::EventStream::getCountVal ( ) const [inline]
```

Get the current value of the counter.

Returns

The current value of the event counter.

8.6.2.20 getEcInstId()

```
InstanceId iris::EventStream::getEcInstId ( ) const [inline]
```

Get the event callback instance id for this event stream.

Returns

The instId for the instance that this event stream calls when an event fires.

8.6.2.21 getEsId()

```
EventStreamId iris::EventStream::getEsId ( ) const [inline]
```

Get the Id of this event stream.

Returns

The esId for this event stream.

8.6.2.22 getEventSourceId()

```
EventSourceId iris::EventStream::getEventSourceId ( ) const [inline]
```

Get the event source id of the event source of this event stream (not the event stream id)

Returns

The event source id of this event stream.

8.6.2.23 getEventSourceInfo()

```
const EventSourceInfo * iris::EventStream::getEventSourceInfo ( ) const [inline]
```

Get the event source info of this event stream.

Returns

The event source info that was used to create this event stream.

8.6.2.24 getProxiedByInstanceId()

```
InstanceId iris::EventStream::getProxiedByInstanceId ( ) const [inline]
```

Get the instance ID of the Iris instance which is a proxy for this event stream.

Returns

The instance ID of the Iris instance which is a proxy

8.6.2.25 getState()

```
virtual IrisErrorCode iris::EventStream::getState (
    IrisValueMap & fields ) [inline], [virtual]
```

Query the current state of the event.

Supported in the derived classes for specific event sources.

Parameters

<i>fields</i>	A map which will be populated with the current values for this event's fields.
---------------	--

Returns

An error code indicating whether the operation was successful.

8.6.2.26 isCounter()

```
bool iris::EventStream::isCounter ( ) const [inline]
```

Is this event stream a counter?

Returns

`true` if this event stream is a counter, otherwise `false`.

8.6.2.27 isEnabled()

```
bool iris::EventStream::isEnabled ( ) const [inline]
```

Is this event stream currently enabled?

Returns

`true` if this event stream is enabled or `false` if it disabled.

8.6.2.28 IsProxiedByOtherInstance()

```
bool iris::EventStream::IsProxiedByOtherInstance ( ) const [inline]
```

Is there another Iris instance which is a proxy for this event stream?

Returns

`true` if this event stream is being proxied by another Iris instance, otherwise `false`.

8.6.2.29 IsProxyForOtherInstance()

```
bool iris::EventStream::IsProxyForOtherInstance ( ) const [inline]
```

Is this event stream a proxy for an event stream in another Iris instance?

Returns

`true` if this event stream is a proxy, otherwise `false`.

8.6.2.30 selfRelease()

```
void iris::EventStream::selfRelease ( ) [inline]
```

Trigger the event stream to be released.

If this event stream is not waiting for any response, release it immediately. Otherwise, release it when it has finished waiting. The event stream is disabled beforehand if it is still enabled.

Note

Do not touch anything related to this object after calling this function.

Do not call this function if this object was not created by 'new'.

8.6.2.31 setCounter()

```
void iris::EventStream::setCounter (
    uint64_t startVal,
    const EventCounterMode & counterMode )
```

Set the counter mode and starting value for this event stream.

Parameters

<i>startVal</i>	The starting value of the counter.
<i>counterMode</i>	The mode in which this counter operates.

8.6.2.32 setOptions()

```
virtual IrisErrorCode iris::EventStream::setOptions (
    const AttributeValueMap & options,
    bool eventStreamCreate,
    std::string & errorMessageOut ) [inline], [virtual]
```

Set options.

Supported in the derived classes for specific event sources. This is called by [setProperty\(\)](#) which in turn is called when the event stream is created. Creating the event stream will fail when this function returns an error and when an options argument is present in `eventStream_create()`.

Parameters

<i>options</i>	Map of options (key/value pairs).
----------------	-----------------------------------

Parameters

<i>eventStreamCreate</i>	True: These are the options set by <code>eventStream_create()</code> . False: These are options set by <code>eventStream_setOptions()</code> .
<i>errorMessageOut</i>	When this function returns an error it should set <code>errorMessageOut</code> to a meaningful error message.

Returns

An error code indicating whether the operation was successful.

8.6.2.33 setProperties()

```
void iris::EventStream::setProperties (
    IrisInstance * irisInstance,
    IrisInstanceEvent * irisInstanceEvent,
    EventSourceId evSrcId,
    InstanceId ecInstId,
    const std::string & ecFunc,
    EventStreamId esId,
    bool syncEc )
```

Initialize this event stream.

Parameters

<i>irisInstance</i>	The IrisInstance that is producing this stream. This will be used to send event callback requests.
<i>irisInstanceEvent</i>	Parent IrisInstanceEvent owning this event stream.
<i>evSrcId</i>	The metadata for the event source generating this stream.
<i>ecInstId</i>	The event callback instId: the instance that this stream calls when an event fires.
<i>ecFunc</i>	The event callback function: the function that is called when an event fires.
<i>esId</i>	The event stream id for this event stream.
<i>syncEc</i>	True if this event stream is synchronous and should send event callbacks as requests. If false event callbacks are sent as notifications and do not wait for a response.

8.6.2.34 setProxiedByInstanceId()

```
void iris::EventStream::setProxiedByInstanceId (
    InstanceId instId ) [inline]
```

Saves the instance ID of the Iris instance that is a proxy for this event stream.

Parameters

<i>instId</i>	The instance ID of the proxy Iris instance
---------------	--

8.6.2.35 setRanges()

```
IrisErrorCode iris::EventStream::setRanges (
    const std::string & aspect,
    const std::vector< uint64_t > & ranges )
```

Set the trace ranges for this event stream.

Parameters

<i>aspect</i>	The field whose range to check.
<i>ranges</i>	A list where each 3 elements form a 3-tuple of (mask, start, end) values to configure ranges.

Returns

An error code indicating whether the ranges could be set successfully.

8.6.3 Member Data Documentation

8.6.3.1 counter

```
bool iris::EventStream::counter {} [protected]
```

— members for a counter —
Is a counter?

8.6.3.2 irisInstance

```
IrisInstance* iris::EventStream::irisInstance {} [protected]
```

— basic members —
The Iris instance that created this event.

8.6.3.3 proxiedByInstanceld

```
InstanceId iris::EventStream::proxiedById {IRIS_UINT64_MAX} [protected]
```

An event stream in another Iris instance is a proxy for this event stream proxiedById - the instance ID of the other Iris instance
The documentation for this class was generated from the following file:

- [IrisInstanceEvent.h](#)

8.7 iris::IrisInstanceBuilder::FieldBuilder Class Reference

Used to set metadata on a register field resource.

```
#include <IrisInstanceBuilder.h>
```

Public Member Functions

- [FieldBuilder](#) & [addEnum](#) (const std::string &symbol, const IrisValue &value, const std::string &description=std::string())
Add a symbol to the enums field for numeric resources.
- [FieldBuilder](#) [addField](#) (const std::string &name, uint64_t lsbOffset, uint64_t bitWidth, const std::string &description)
Add another subregister field to the parent register.
- [FieldBuilder](#) [addLogicalField](#) (const std::string &name, uint64_t bitWidth, const std::string &description)
Add another logical subregister field to the parent register.
- [FieldBuilder](#) & [addStringEnum](#) (const std::string &stringValue, const std::string &description=std::string())
Add a symbol to the enums field for string resources.
- [FieldBuilder](#) (IrisInstanceResource::ResourceInfoAndAccess &info_, [RegisterBuilder](#) *parent_reg_, [IrisInstanceBuilder](#) *instance_builder_)
- ResourceId [getRscId](#) () const

- Return the rscId that was allocated for this resource.*

 - [FieldBuilder](#) & [getRscId](#) (ResourceId &rscIdOut)

Get the rscId that was allocated for this resource.
- [RegisterBuilder](#) & [parent](#) ()

Get the [RegisterBuilder](#) for the parent register.
- [FieldBuilder](#) & [setAddressOffset](#) (uint64_t addressOffset)

Set the `addressOffset` field.
- [FieldBuilder](#) & [setBitWidth](#) (uint64_t bitWidth)

Set the `bitWidth` field.
- [FieldBuilder](#) & [setBreakpointSupportInfo](#) (const std::string &supported)

Set the `breakpointSupport` field.
- [FieldBuilder](#) & [setCanonicalRn](#) (uint64_t canonicalRn_)

Set the `canonicalRn` field.
- [FieldBuilder](#) & [setCanonicalRnElfDwarf](#) (uint16_t architecture, uint16_t dwarfRegNum)

Set the `canonicalRn` field for "ElfDwarf" scheme.
- [FieldBuilder](#) & [setName](#) (const std::string &cname)

Set the `cname` field.
- [FieldBuilder](#) & [setDescr](#) (const std::string &description)

Obsolete alias for [setDescription\(\)](#). Do not use.
- [FieldBuilder](#) & [setDescription](#) (const std::string &description)

Set the `description` field.
- [FieldBuilder](#) & [setFormat](#) (const std::string &format)

Set the `format` field.
- [FieldBuilder](#) & [setLsbOffset](#) (uint64_t lsbOffset)

Set the `lsbOffset` field.
- [FieldBuilder](#) & [setName](#) (const std::string &name)

Set the `name` field.
- [FieldBuilder](#) & [setParentRscId](#) (ResourceId parentRscId)

Set the `parentRscId` field.
- `template<IrisErrorCode>(const ResourceInfo &, ResourceReadResult &) FUNC>`
[FieldBuilder](#) & [setReadDelegate](#) ()

Set the delegate to read the resource.
- [FieldBuilder](#) & [setReadDelegate](#) ([ResourceReadDelegate](#) readDelegate)

Set the delegate to read the resource.
- `template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>`
[FieldBuilder](#) & [setReadDelegate](#) (T *instance)

Set the delegate to read the resource.
- `template<typename T >`
[FieldBuilder](#) & [setResetData](#) (std::initializer_list< T > &&t)

Set the `resetData` field for wide registers.
- [FieldBuilder](#) & [setResetData](#) (uint64_t value)

Set the `resetData` field to a value <= 64 bit.
- `template<typename Container >`
[FieldBuilder](#) & [setResetDataFromContainer](#) (const Container &container)

Set the `resetData` field for wide registers.
- [FieldBuilder](#) & [setResetString](#) (const std::string &resetString)

Set the `resetString` field.
- [FieldBuilder](#) & [setRwMode](#) (const std::string &rwMode)

Set the `rwMode` field.
- [FieldBuilder](#) & [setSubRscId](#) (uint64_t subRscId)

Set the `subRscId` field.

- [FieldBuilder](#) & [setTag](#) (const std::string &tag)
Set the named boolean tag to true (e.g. isPc)
- [FieldBuilder](#) & [setTag](#) (const std::string &tag, const IrisValue &value)
Set a tag to the specified value.
- [FieldBuilder](#) & [setType](#) (const std::string &type)
Set the type field.
- template<IrisErrorCode(*)>(const ResourceInfo &, const [ResourceWriteValue](#) &) FUNC>
[FieldBuilder](#) & [setWriteDelegate](#) ()
Set the delegate to write the resource.
- [FieldBuilder](#) & [setWriteDelegate](#) ([ResourceWriteDelegate](#) writeDelegate)
Set the delegate to write the resource.
- template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const [ResourceWriteValue](#) &) METHOD>
[FieldBuilder](#) & [setWriteDelegate](#) (T *instance)
Set the delegate to write the resource.
- template<typename T >
[FieldBuilder](#) & [setWriteMask](#) (std::initializer_list< T > &&t)
Set the writeMask field for wide registers.
- [FieldBuilder](#) & [setWriteMask](#) (uint64_t value)
Set the writeMask field to a value <= 64 bit.
- template<typename Container >
[FieldBuilder](#) & [setWriteMaskFromContainer](#) (const Container &container)
Set the writeMask field for wide registers.

Protected Attributes

- [IrisInstanceResource::ResourceInfoAndAccess](#) * **info** {}
- [IrisInstanceBuilder](#) * **instance_builder** {}
- [RegisterBuilder](#) * **parent_reg** {}

8.7.1 Detailed Description

Used to set metadata on a register field resource.

8.7.2 Member Function Documentation

8.7.2.1 addEnum()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::addEnum (
    const std::string & symbol,
    const IrisValue & value,
    const std::string & description = std::string() ) [inline]
```

Add a symbol to the enums field for numeric resources.

This should be called multiple times to add multiple symbols.

Parameters

<i>symbol</i>	The symbol string to be associated with the specified value.
<i>value</i>	The value of this symbol.
<i>description</i>	A description of this symbol.

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.2 addField()

```
FieldBuilder iris::IrisInstanceBuilder::FieldBuilder::addField (
    const std::string & name,
    uint64_t lsbOffset,
    uint64_t bitWidth,
    const std::string & description ) [inline]
```

Add another subregister field to the parent register.

See also

[RegisterBuilder::addField](#)

8.7.2.3 addLogicalField()

```
FieldBuilder iris::IrisInstanceBuilder::FieldBuilder::addLogicalField (
    const std::string & name,
    uint64_t bitWidth,
    const std::string & description ) [inline]
```

Add another logical subregister field to the parent register.

See also

[RegisterBuilder::addField](#)

8.7.2.4 addStringEnum()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::addStringEnum (
    const std::string & stringValue,
    const std::string & description = std::string() ) [inline]
```

Add a symbol to the enums field for string resources.

This should be called multiple times to add multiple symbols.

Parameters

<i>value</i>	The string value of this symbol. This is also used as the symbols string.
<i>description</i>	A description of this symbol.

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.5 getRscId() [1/2]

```
ResourceId iris::IrisInstanceBuilder::FieldBuilder::getRscId ( ) const [inline]
```

Return the rscl that was allocated for this resource.

Returns

The rscl that was allocated for this resource.

8.7.2.6 getRscId() [2/2]

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::getRscId (
    ResourceId & rscIdOut ) [inline]
```

Get the rscId that was allocated for this resource.

This variant is useful to get the ResourceId of fields added in a chained call where return values are not practical.

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.7 parent()

```
RegisterBuilder & iris::IrisInstanceBuilder::FieldBuilder::parent ( ) [inline]
```

Get the [RegisterBuilder](#) for the parent register.

Returns

The [RegisterBuilder](#) object for the parent register.

8.7.2.8 setAddressOffset()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setAddressOffset (
    uint64_t addressOffset ) [inline]
```

Set the addressOffset field.

Parameters

<i>addressOffset</i>	The addressOffset field of the RegisterInfo object.
----------------------	---

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.9 setBitWidth()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setBitWidth (
    uint64_t bitWidth ) [inline]
```

Set the bitWidth field.

Parameters

<i>bitWidth</i>	The bitWidth field of the ResourceInfo object.
-----------------	--

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.10 setBreakpointSupportInfo()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setBreakpointSupportInfo (
    const std::string & supported ) [inline]
```

Set the breakpointSupport field.

Parameters

<i>supported</i>	The breakpointSupport field of the RegisterInfo object.
------------------	---

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.11 setCanonicalRn()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setCanonicalRn (
    uint64_t canonicalRn_ ) [inline]
```

Set the canonicalRn field.

Note: Use [setCanonicalRnElfDwarf\(\)](#) when using the "ElfDwarf" scheme.

Parameters

<i>canonicalRn</i>	The canonicalRn field of the RegisterInfo object.
--------------------	---

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.12 setCanonicalRnElfDwarf()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setCanonicalRnElfDwarf (
    uint16_t architecture,
    uint16_t dwarfRegNum ) [inline]
```

Set the canonicalRn field for "ElfDwarf" scheme.

Parameters

<i>architecture</i>	ELF EM_* constant for architecture.
<i>dwarfRegNum</i>	DWARF register number for architecture.

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.13 setCname()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setCname (
    const std::string & cname ) [inline]
```

Set the cname field.

Parameters

<i>cname</i>	The cname field of the ResourceInfo object.
--------------	---

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.14 setDescription()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setDescription (
    const std::string & description ) [inline]
```

Set the description field.

Parameters

<i>description</i>	The description field of the ResourceInfo object.
--------------------	---

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.15 setFormat()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setFormat (
    const std::string & format ) [inline]
```

Set the format field.

Parameters

<i>format</i>	The format field of the ResourceInfo object.
---------------	--

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.16 setLsbOffset()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setLsbOffset (
    uint64_t lsbOffset ) [inline]
```

Set the lsbOffset field.

Parameters

<i>lsbOffset</i>	The lsbOffset field of the RegisterInfo object.
------------------	---

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.17 setName()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setName (
    const std::string & name ) [inline]
```

Set the name field.

Parameters

<i>name</i>	The name field of the ResourceInfo object.
-------------	--

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.18 setParentRscId()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setParentRscId (
    ResourceId parentRscId ) [inline]
```

Set the `parentRscId` field.

This function makes this register a child of the specified parent. It is not necessary to call this function when adding child registers using the [addField\(\)](#) function.

Parameters

<i>parent</i> ↔ <i>RscId</i>	The rscId of the parent register.
---------------------------------	-----------------------------------

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.19 setReadDelegate() [1/3]

```
template<IrisErrorCode(*) (const ResourceInfo &, ResourceReadResult &) FUNC>
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setReadDelegate ( ) [inline]
```

Set the delegate to read the resource.

Set a delegate which calls function `FUNC()`.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceReadDelegate](#)

Template Parameters

<i>FUNC</i>	A resource read delegate function.
-------------	------------------------------------

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.20 setReadDelegate() [2/3]

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setReadDelegate (
    ResourceReadDelegate readDelegate ) [inline]
```

Set the delegate to read the resource.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceReadDelegate](#)

Parameters

<i>readDelegate</i>	ResourceReadDelegate object.
---------------------	------------------------------

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.21 setReadDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setReadDelegate (
    T * instance ) [inline]
```

Set the delegate to read the resource.

Set a delegate which calls METHOD() on an instance of class T.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceReadDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a resource read delegate.
<i>METHOD</i>	A resource read delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.22 setResetData() [1/2]

```
template<typename T >
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setResetData (
    std::initializer_list< T > && t ) [inline]
```

Set the `resetData` field for wide registers.

This function accepts a braced initializer-list and is otherwise identical to [setResetDataFromContainer\(\)](#).

Each element will be promoted/narrowed to `uint64_t`.

Parameters

<i>t</i>	Braced initializer-list.
----------	--------------------------

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.23 setResetData() [2/2]

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setResetData (
    uint64_t value ) [inline]
```

Set the `resetData` field to a value ≤ 64 bit.

If the register is wider than the passed value the value is zero extended.

If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>value</i>	resetData value of the register.
--------------	----------------------------------

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.24 setResetDataFromContainer()

```
template<typename Container >
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setResetDataFromContainer (
    const Container & container ) [inline]
```

Set the `resetData` field for wide registers.

Container must be a type which allows to iterate over `uint64_t` bit chunks of the value, least significant bits first, for example `std::array<uint64_t>` or `std::vector<uint64_t>`.

Each element of the container will be promoted/narrowed to `uint64_t`.

If the register is wider than the passed value the value is zero extended.

If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>container</i>	Container containing the value in 64-bit chunks.
------------------	--

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.25 setResetString()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setResetString (
    const std::string & resetString ) [inline]
```

Set the `resetString` field.

Set the reset value for string registers.

Parameters

<i>resetString</i>	The <code>resetString</code> field of the <code>RegisterInfo</code> object.
--------------------	---

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.26 setRwMode()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setRwMode (
    const std::string & rwMode ) [inline]
```

Set the `rwMode` field.

Parameters

<i>rwMode</i>	The <code>rwMode</code> field of the ResourceInfo object.
---------------	---

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.27 setSubRscId()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setSubRscId (
    uint64_t subRscId ) [inline]
```

Set the `subRscId` field.

Parameters

<i>sub↔ RscId</i>	The <code>subRscId</code> field of the ResourceInfo object.
-----------------------	---

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.28 setTag() [1/2]

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setTag (
    const std::string & tag ) [inline]
```

Set the named boolean tag to true (e.g. `isPc`)

Parameters

<i>tag</i>	The name of the tag to set.
------------	-----------------------------

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.29 setTag() [2/2]

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setTag (
    const std::string & tag,
    const IrisValue & value ) [inline]
```


Set a tag to the specified value.

Parameters

<i>tag</i>	The name of the tag to set.
<i>value</i>	The value to set the tag to.

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.30 setType()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setType (
    const std::string & type ) [inline]
```

Set the `type` field.

Parameters

<i>type</i>	The type field of the ResourceInfo object.
-------------	--

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.31 setWriteDelegate() [1/3]

```
template<IrisErrorCode(*)>(const ResourceInfo &, const ResourceWriteValue &) FUNC>
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setWriteDelegate ( ) [inline]
```

Set the delegate to write the resource.

Set a delegate which calls function `FUNC()`.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceWriteDelegate](#)

Template Parameters

<i>FUNC</i>	A resource write delegate function.
-------------	-------------------------------------

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.32 setWriteDelegate() [2/3]

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setWriteDelegate (
    ResourceWriteDelegate writeDelegate ) [inline]
```

Set the delegate to write the resource.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceWriteDelegate](#)

Parameters

<i>writeDelegate</i>	ResourceWriteDelegate object.
----------------------	-------------------------------

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.33 setWriteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &)
METHOD>
```

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setWriteDelegate (
    T * instance ) [inline]
```

Set the delegate to write the resource.

Set a delegate which calls METHOD() on an instance of class T.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceWriteDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a resource write delegate.
<i>METHOD</i>	A resource write delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.34 setWriteMask() [1/2]

```
template<typename T >
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setWriteMask (
    std::initializer_list< T > && t ) [inline]
```

Set the writeMask field for wide registers.

This function accepts a braced initializer-list and is otherwise identical to [setWriteMaskFromContainer\(\)](#).

Each element will be promoted/narrowed to uint64_t.

Parameters

<i>t</i>	Braced initializer-list.
----------	--------------------------

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.35 setWriteMask() [2/2]

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setWriteMask (
    uint64_t value ) [inline]
```

Set the writeMask field to a value <= 64 bit.

If the register is wider than the passed value the value is zero extended.

If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>value</i>	writeMask value of the register.
--------------	----------------------------------

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

8.7.2.36 setWriteMaskFromContainer()

```
template<typename Container >
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setWriteMaskFromContainer (
    const Container & container ) [inline]
```

Set the writeMask field for wide registers.

Container must be a type which allows to iterate over uint64_t bit chunks of the value, least significant bits first, for example std::array<uint64_t> or std::vector<uint64_t>.

Each element of the container will be promoted/narrowed to uint64_t.

If the register is wider than the passed value the value is zero extended.

If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>container</i>	Container containing the value in 64-bit chunks.
------------------	--

Returns

A reference to this [FieldBuilder](#) object allowing calls to be chained together.

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

- [IrisInstanceBuilder.h](#)

8.8 iris::IrisCConnection Class Reference

Provide an IrisConnectionInterface which loads an IrisC library.

```
#include <IrisCConnection.h>
```

Inherits IrisConnectionInterface.

Public Member Functions

- virtual IrisInterface * **getIrisInterface** () IRIS_OVERRIDE
Get the IrisInterface for this connection. See also IrisConnectionInterface::getIrisInterface().
- **IrisCConnection** (IrisC_Funcions *functions)
- virtual IrisErrorCode **processAsyncMessages** (bool waitForAMessage) IRIS_OVERRIDE
Process asynchronous messages for the calling thread. See also IrisConnectionInterface::processAsyncMessages().
- virtual uint64_t **registerIrisInterfaceChannel** (IrisInterface *iris_interface, const std::string &connectionInfo) IRIS_OVERRIDE
Register a communication channel. See also IrisConnectionInterface::registerIrisInterfaceChannel().

- virtual void **unregisterIrisInterfaceChannel** (uint64_t channelId) IRIS_OVERRIDE
Unregister a communication channel. See also `IrisConnectionInterface::unregisterIrisInterfaceChannel()`.

Protected Member Functions

- int64_t **IrisC_handleMessage** (const uint64_t *message)
Wrapper functions to call the underlying IrisC functions.
- int64_t **IrisC_processAsyncMessages** (bool waitForAMessage)
- int64_t **IrisC_registerChannel** (IrisC_CommunicationChannel *channel, uint64_t *channel_id_out)
- int64_t **IrisC_unregisterChannel** (uint64_t channel_id)
- **IrisCConnection** ()
Construct an empty object. Used by subclasses that need to load a DSO and call `init()`.

Protected Attributes

- void * **iris_c_context**
Context pointer to use when calling `IrisC_` functions. This is also needed by subclasses.*

8.8.1 Detailed Description

Provide an `IrisConnectionInterface` which loads an IrisC library.

See also

[IrisClient](#)

[IrisGlobalInstance](#)

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

- [IrisCConnection.h](#)

8.9 iris::IrisClient Class Reference

Inherits `IrisInterface`, `impl::IrisProcessEventsInterface`, and `IrisConnectionInterface`.

Public Member Functions

- void [connect](#) (const std::string &connectionSpec)
- IrisErrorCode [connect](#) (const std::string &hostname, uint16_t port, unsigned timeoutInMs, std::string &error↵ ResponseOut)
- void [connectCommandLine](#) (const std::vector< std::string > &commandLine_, const std::string &program↵ Name)
- void **connectCommandLineKeepOtherArgs** (std::vector< std::string > &commandLine, const std::string &programName)
Same as [connectCommandLine\(\)](#) but remove all known arguments from `commandLine` and keep all other arguments in `commandLine`.
- void [connectSocketFd](#) (SocketFd socketfd, unsigned timeoutInMs=1000)
- IrisErrorCode [disconnect](#) ()
- bool [disconnectAndWaitForChildToExit](#) (double timeoutInMs=5000, double timeoutInMsAfterSigInt=5000, double timeoutInMsAfterSigKill=5000)
- pid_t **getChildPid** () const
Get child process id of previously spawned process or 0 if no process was spawned yet using [spawnAndConnect\(\)](#).
- std::string **getConnectionStr** () const
Get connection string, describing the Iris server we are connected to.
- impl::IrisRpcAdapterTcp::Format **getEffectiveSendingFormat** () const
Get effective sending format that Rpc adapter uses.

- [IrisInstance](#) & [getIrisInstance](#) ()
- virtual IrisInterface * **getIrisInterface** () override
- int **getLastExitStatus** () const
Get last exit status of child process, or -1 if the child process did not yet exit.
- IrisInterface * **getSendingInterface** ()
Get interface for sending messages to the server.
- void [initServiceServer](#) (impl::IrisTcpSocket *socket_)
- **IrisClient** (const service::IrisServiceTcpServer *, const std::string &instName=std::string())
Service constructor to initialize IrisService Server on IrisService side.
- [IrisClient](#) (const std::string &hostname, uint16_t port, const std::string &instName=std::string())
Construct a connection to an Iris server.
- **IrisClient** (const std::string &instName=std::string(), const std::string &connectionSpec=std::string())
Client constructor.
- bool **isConnected** () const
Return true iff connected to a server.
- void [loadPlugin](#) (const std::string &plugin_name)
- virtual IrisErrorCode **processAsyncMessages** (bool waitForAMessage) override
- virtual void [processEvents](#) () override
- uint64_t **registerChannel** (IrisC_CommunicationChannel *channel, const std::string &connectionInfo)
- virtual uint64_t **registerIrisInterfaceChannel** (IrisInterface *iris_interface, const std::string &connectionInfo) override
- void [setInstanceName](#) (const std::string &instName)
- void **setIrisMessageLogLevel** (unsigned level, bool increaseOnly=false)
Enable message logging.
- void **setPreferredSendingFormat** (impl::IrisRpcAdapterTcp::Format p)
Set preferred sending format that Rpc adapter uses.
- void [setSleepOnDestructionMs](#) (uint64_t sleepOnDestructionMs_)
- void **setVerbose** (unsigned level, bool increaseOnly=false)
Set verbose level.
- void [spawnAndConnect](#) (const std::vector< std::string > &modelCommandLine, const std::string &additionalServerArgs=std::string(), const std::string &additionalClientArgs=std::string())
- virtual void [stopWaitForEvent](#) () override
- void **unloadPlugin** ()
- void **unregisterChannel** (uint64_t channelId)
- virtual void **unregisterIrisInterfaceChannel** (uint64_t channelId) override
- virtual void [waitForEvent](#) () override
- bool [waitpidWithTimeout](#) (pid_t pid, int *status, int options, double timeoutInMs)
- virtual ~**IrisClient** ()
Destructor.

Static Public Member Functions

- static std::string [getConnectCommandLineHelp](#) ()

Public Attributes

- const std::string [connectionHelpStr](#)
Connection help string.

8.9.1 Constructor & Destructor Documentation

8.9.1.1 IrisClient()

```
iris::IrisClient::IrisClient (
    const std::string & hostname,
    uint16_t port,
    const std::string & instName = std::string() ) [inline]
```

Construct a connection to an Iris server.

Parameters

<i>hostname</i>	<p>Hostname of the Iris server. This can be an IP address. For example:</p> <ul style="list-style-type: none"> "192.168.0.5" IP address of a different host. "127.0.0.1" Loopback IP address to connect to a server on the same machine. "localhost" Hostname of the loopback interface. Port == 0 means to scan ports 7100 to 7109. "foo.bar.com" Hostname of a remote machine.
<i>port</i>	Server port number to connect to on the host.

8.9.2 Member Function Documentation

8.9.2.1 connect() [1/2]

```
void iris::IrisClient::connect (
    const std::string & connectionSpec ) [inline]
```

Connect to an Iris server.

The connection details are specified as a string. See "connectionHelpStr" for syntax. This function is self documenting: Passing "help" will return a list of all supported connection types and their syntax, as an E_help_↵ message error.

This throws E_not_connected when connectionSpec was erroneous, and E_socket_error or E_connection_refused when the connection could not be established. In case of an error the socket is closed.

8.9.2.2 connect() [2/2]

```
IrisErrorCode iris::IrisClient::connect (
    const std::string & hostname,
    uint16_t port,
    unsigned timeoutInMs,
    std::string & errorResponseOut ) [inline]
```

Connect to TCP server on hostname:port.

If hostname == "localhost" and port == 0 then a port scan on ports 7100 to 7109 is done. In case of an error the socket is closed.

8.9.2.3 connectCommandLine()

```
void iris::IrisClient::connectCommandLine (
    const std::vector< std::string > & commandLine_,
    const std::string & programName ) [inline]
```

Connect via command-line-like interface.

This high-level function is convenient for tools which transparently want to allow all kinds of connections.

This either spawns a model child process ("spawn") or connects to a running model process ("tcp", default). See [getConnectCommandLineHelp\(\)](#) for command line syntax and supported arguments. All errors are reported via exceptions.

This is a frontend to [spawnAndConnect\(\)](#) and to [connect\(\)](#).

commandLine: See [getConnectCommandLineHelp\(\)](#) (or pass the command line ["help"]) for supported arguments.
 programName: This is just used in the help message.

8.9.2.4 connectSocketFd()

```
void iris::IrisClient::connectSocketFd (
    SocketFd socketfd,
    unsigned timeoutInMs = 1000 ) [inline]
```

Connect using an existing socketFd. All errors are reported by exceptions. In case of an error the socket is closed.

8.9.2.5 disconnect()

```
IrisErrorCode iris::IrisClient::disconnect ( ) [inline]
```

Disconnect from server. Close socket. (Only for mode IRIS_TCP_CLIENT.)

8.9.2.6 disconnectAndWaitForChildToExit()

```
bool iris::IrisClient::disconnectAndWaitForChildToExit (
    double timeoutInMs = 5000,
    double timeoutInMsAfterSigInt = 5000,
    double timeoutInMsAfterSigKill = 5000 ) [inline]
```

Disconnect and wait for child process (previously spawned with [spawnAndConnect\(\)](#)) to exit. If no model was spawned this is silently ignored.

Wait at most timeoutInMs until the child exits. If the child did not exit by then, send a SIGINT and wait for timeoutInMsAfterSigInt until the child exits. If the child did not exit by then, send a SIGKILL and wait for timeoutInMsAfterSigKill until the child exits. If the child did not exit by then, an E_not_connected exception is thrown. If timeoutInMs is 0, do not wait and continue with SIGINT. If timeoutAfterSigInt is 0, do not issue a SIGINT and continue with SIGKILL. If timeoutAfterSigKill is 0, do not issue a SIGKILL and throw an E_not_connected exception. If any of the timeouts is < 0, wait indefinitely.

Return true if the child exited, else false.

8.9.2.7 getConnectCommandLineHelp()

```
static std::string iris::IrisClient::getConnectCommandLineHelp ( ) [inline], [static]
```

Get help string for [connectCommandLine\(\)](#). This can be used by tools using [connectCommandLine\(\)](#) as part of their -help message.

8.9.2.8 getIrisInstance()

```
IrisInstance & iris::IrisClient::getIrisInstance ( ) [inline]
```

Get contained [IrisInstance](#). This can be used as a generic client instance to call Iris functions.

8.9.2.9 initServiceServer()

```
void iris::IrisClient::initServiceServer (
    impl::IrisTcpSocket * socket_ ) [inline]
```

Initialize as an IrisService server, only used in IRIS_SERVICE_SERVER mode. This function will store pointer to IrisTcpSocket created by IrisService and initialize adapter as a server. -socket_ pointer to IrisTcpSocket created by IrisService when receiving new connection. (TODO safer memory management of this object) -return Nothing.

8.9.2.10 loadPlugin()

```
void iris::IrisClient::loadPlugin (
    const std::string & plugin_name ) [inline]
```

Load Plugin function, only used in IRIS_SERVICE_SERVER mode Only one plugin can be loaded at a a time

8.9.2.11 processEvents()

```
virtual void iris::IrisClient::processEvents ( ) [inline], [override], [virtual]
```

Client main processing function.

- Check for incoming requests/responses and process them .
- Check for pending outgoing requests/responses and process them. This function is ideal for integrating the client into other processing environments in one of the following ways: (1) Thread-less: Requests are only executed from within [processEvents\(\)](#).
- pro: Iris request and responses are always synchronized with the rest of the code of the client. No explicit synchronization (mutexes etc.) necessary.
- con: No blocking Iris requests can be called from within received synchronous callbacks. (2) Asynchronous (`handleRequestAsynchronously = true`): Requests are executed in another thread
- pro: Blocking Iris requests can be called from within received synchronous callbacks transparently.
- con: Received Iris requests are called on another thread and they require explicit synchronization to be synchronized with the rest of the code of the client. It is harmless to call this function when there is nothing to do.

8.9.2.12 [setInstanceName\(\)](#)

```
void iris::IrisClient::setInstanceName (
    const std::string & instName ) [inline]
```

Set instance name of the contained Iris instance returned by [getIrisInstance](#). This must be called before [connect\(\)](#).

8.9.2.13 [setSleepOnDestructionMs\(\)](#)

```
void iris::IrisClient::setSleepOnDestructionMs (
    uint64_t sleepOnDestructionMs_ ) [inline]
```

Sleep a short time on destruction to de-interleave output by different processes. This has not functional impact or purpose. It just beautifies the output on stdout.

8.9.2.14 [spawnAndConnect\(\)](#)

```
void iris::IrisClient::spawnAndConnect (
    const std::vector< std::string > & modelCommandLine,
    const std::string & additionalServerArgs = std::string(),
    const std::string & additionalClientArgs = std::string() ) [inline]
```

Spawn model and connect to it. All errors are reported via exceptions. `additionalServerArgs` are added to the models `-iris-connect` argument and ultimately passed to `IrisTcpServer::startServer()`, for example `"verbose=1"` to enable verbose messages. `additionalClientArgs` are added to the argument passed to [IrisClient::connect\(\)](#), for example `"verbose=1,timeout=2000"` to enable verbose messages and a 2 second timeout.

8.9.2.15 [stopWaitForEvent\(\)](#)

```
virtual void iris::IrisClient::stopWaitForEvent ( ) [inline], [override], [virtual]
```

Stop waiting in [waitForEvent\(\)](#). Return from [waitForEvent\(\)](#) as soon as possible even without a socket event.

8.9.2.16 [waitForEvent\(\)](#)

```
virtual void iris::IrisClient::waitForEvent ( ) [inline], [override], [virtual]
```

Wait for any event which would cause [processEvents\(\)](#) to do some work. This function intentionally blocks until there is something useful to do. This function can be interrupted by calling [stopWaitForEvent\(\)](#).

8.9.2.17 [waitpidWithTimeout\(\)](#)

```
bool iris::IrisClient::waitpidWithTimeout (
    pid_t pid,
    int * status,
```

```
int options,
double timeoutInMs ) [inline]
```

waitpid() with timeout. Throw exceptions on errors. Return true if the child exited within the timeout, else false.

8.9.3 Member Data Documentation

8.9.3.1 connectionHelpStr

```
const std::string iris::IrisClient::connectionHelpStr
```

Initial value:

```
=
    "Supported connection types:\n"
    "tcp[=HOST][,port=PORT][,timeout=T]\n"
    "    Connect to an Iris TCP server on HOST:PORT.\n"
    "    The default for HOST is 'localhost' and the default for PORT is 0 if HOST is 'localhost' and 7100
otherwise. If PORT is 0 then a port scan on ports 7100 to 7109 is done.\n"
    "    T is the connection timeout in ms (defaults to 100 if PORT=0, else 1000).\n"
    "\n"
    "socketfd=FD[,timeout=T]\n"
    "    Use socket file descriptor FD as an established UNIX domain socket connection.\n"
    "    T is the timeout for the Iris handshake in ms.\n"
    "\n"
    "General parameters:\n"
    "    verbose[=N]: Increase verbose level of IrisClient to level N (0..3).\n"
    "    iris-log[=N]: Log Iris functions calls (1=pretty, 2=JSON, 3=JSON-multiline, +8=U64JSON, +16=time,
+32=reltime).\n"
```

Connection help string.

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

- [IrisClient.h](#)

8.10 iris::IrisCommandLineParser Class Reference

```
#include <IrisCommandLineParser.h>
```

Classes

- [struct Option](#)
Option container.

Public Member Functions

- [Option](#) & [addOption](#) (char shortOption, const std::string &longOption, const std::string &help, const std::string &formalArgumentName, int64_t defaultValue)
- [Option](#) & [addOption](#) (char shortOption, const std::string &longOption, const std::string &help, const std::string &formalArgumentName=std::string(), const std::string &defaultValue=std::string())
- void [clear](#) ()
- double [getDb](#) (const std::string &longOption) const
- std::string [getHelpMessage](#) () const
- int64_t [getInt](#) (const std::string &longOption) const
- std::vector< std::string > [getList](#) (const std::string &longOption) const
Get list of elements of a list option.
- std::map< std::string, std::string > [getMap](#) (const std::string &longOption) const
- std::vector< std::string > & [getNonOptionArguments](#) ()
- const std::vector< std::string > & [getNonOptionArguments](#) () const
Get non-option arguments.
- std::string [getProgramName](#) () const
Get program name.
- std::string [getStr](#) (const std::string &longOption) const

- Get string value.*
- uint64_t **getSwitch** (const std::string &longOption) const
 - Check how many times an option switch (an option without an argument) was specified.*
- uint64_t **getUInt** (const std::string &longOption) const
- **IrisCommandLineParser** (const std::string &programName, const std::string &usageHeader, const std::string &versionStr, bool keepDashDash=false)
- bool **isSpecified** (const std::string &longOption) const
- void **noNonOptionArguments** ()
- bool **operator()** (const std::string &longOption) const
 - Check whether an option was specified.*
- int **parseCommandLine** (int argc, char **argv)
- int **parseCommandLine** (int argc, const char **argv)
- void **pleaseSpecifyOneOf** (const std::vector< std::string > &options, const std::vector< std::string > &formalNonOptionArguments=std::vector< std::string >())
- int **printError** (const std::string &message) const
 - Print error message (and do not exit).*
- int **printErrorAndExit** (const std::exception &e) const
- int **printErrorAndExit** (const std::string &message) const
- int **printMessage** (const std::string &message, int error=0, bool exit=false) const
- void **setMessageFunc** (const std::function< int(const std::string &message, int error, bool exit)> &messageFunc)
- void **setProgramName** (const std::string &programName_, bool append=false)
 - Set/override program name.*
- void **setValue** (const std::string &longOption, const std::string &value, bool append=false)
- void **unsetValue** (const std::string &longOption)

Static Public Member Functions

- static int **defaultMessageFunc** (const std::string &message, int error, bool exit)

Static Public Attributes

- static const bool **KeepDashDash** = true
 - Keep "--" in the non-option arguments because it has semantics for the application beyond stopping option parsing.*

8.10.1 Detailed Description

Generic command line parser.

This covers roughly all features supported by GNU getopt_long() and provides -h/--help and --version.

Usage:

1. Declare options by calling **addOption()** for each option.
2. Parse command line by calling **parseCommandLine()**.
3. Retrieve command line option values by calling the get...() functions.

Example:

8.10.2 Constructor & Destructor Documentation

8.10.2.1 IrisCommandLineParser()

```
iris::IrisCommandLineParser::IrisCommandLineParser (
    const std::string & programName,
    const std::string & usageHeader,
    const std::string & versionStr,
    bool keepDashDash = false )
```

Constructor. programName, usageHeader and versionStr: Appears in the `–help` and `–version` messages. keepDashDash: Keep `--` in the non-option arguments because it has semantics for the application beyond stopping option parsing.

8.10.3 Member Function Documentation

8.10.3.1 addOption() [1/2]

```
Option & iris::IrisCommandLineParser::addOption (
    char shortOption,
    const std::string & longOption,
    const std::string & help,
    const std::string & formalArgumentName,
    int64_t defaultValue ) [inline]
```

Same as above for integer defaults. (Without this overload, specifying an integer default of 0 will automatically get converted to a NULL const char* and then to a std::string which segfaults.)

8.10.3.2 addOption() [2/2]

```
Option & iris::IrisCommandLineParser::addOption (
    char shortOption,
    const std::string & longOption,
    const std::string & help,
    const std::string & formalArgumentName = std::string(),
    const std::string & defaultValue = std::string() )
```

Add command line option. shortOption: Single character or 0 if no short option. longOption: Long option (mandatory, must be unique and non-empty). help: Description for `–help`. formalArgumentName: Empty means: This option has no argument (switch). Nonempty means: This option has an argument and this is named 'formalArgumentName' in the `–help` message. defaultValue: Default value of this option when not specified on the command line. When defaultValue is not specified: By default `getSwitch()`, `getInt()` and `getUInt()` return 0 and `getStr()` returns an empty string.

8.10.3.3 clear()

```
void iris::IrisCommandLineParser::clear ( )
```

Clear all values parsed by a previous `parseCommandLine` call. All options will be reset to their default values. All option definitions (`addOption()`) will be preserved.

8.10.3.4 defaultMessageFunc()

```
static int iris::IrisCommandLineParser::defaultMessageFunc (
    const std::string & message,
    int error,
    bool exit ) [static]
```

Default message function. The default message function prints message on stdout and exits with "error" status if `exit==true`, else it returns error status.

8.10.3.5 getDbI()

```
double iris::IrisCommandLineParser::getDbI (
    const std::string & longOption ) const
```

Get double value. (This will print an error and exit when there is a parse error.)

8.10.3.6 getHelpMessage()

```
std::string iris::IrisCommandLineParser::getHelpMessage ( ) const
```

Get help message. (`parserCommandLine()` automatically prints this on `-help` so there is usually no need to call this function.)

8.10.3.7 getInt()

```
int64_t iris::IrisCommandLineParser::getInt (
    const std::string & longOption ) const
```

Get integer value. (This will print an error and exit when there is a parse error.)

8.10.3.8 getMap()

```
std::map< std::string, std::string > iris::IrisCommandLineParser::getMap (
    const std::string & longOption ) const
```

Get NAME->VALUE map of elements of a list option. The elements are assumed to have the format "NAME=↔VALUE" or "NAME". If "=VALUE" is missing then VALUE is the empty string.

8.10.3.9 getNonOptionArguments()

```
std::vector< std::string > & iris::IrisCommandLineParser::getNonOptionArguments ( ) [inline]
```

Get read/write access to non-option arguments. This is useful when chaining different non-option argument parsers.

8.10.3.10 getUInt()

```
uint64_t iris::IrisCommandLineParser::getUInt (
    const std::string & longOption ) const
```

Get unsigned integer value. (This will print an error and exit when there is a parse error.)

8.10.3.11 isSpecified()

```
bool iris::IrisCommandLineParser::isSpecified (
    const std::string & longOption ) const
```

Return true iff option is specified explicitly on the command line. (This can be used to detect whether an option was present on the command line even if it was just set to its default value.)

8.10.3.12 noNonOptionArguments()

```
void iris::IrisCommandLineParser::noNonOptionArguments ( )
```

Print an error for each non-option argument and exit if any non-option arguments are present. Call this after [parseCommandLine\(\)](#) for programs which do not support any non-option arguments as these are otherwise silently ignored.

8.10.3.13 parseCommandLine()

```
int iris::IrisCommandLineParser::parseCommandLine (
    int argc,
    const char ** argv )
```

Parse command line. After calling this function the named argument values can be retrieved by the `get...()` functions. All arguments after the first occurrence of a "--" argument are treated as non-option arguments. Also handles `-help` and `-version` and `exit()`s when these are specified.

`argv[0]` is ignored. The program name is passed in the constructor argument.

Calling [parseCommandLine\(\)](#) again will add and/or override options as if they were in a single command line.

Return value: By default [parseCommandLine\(\)](#) exits (and so does not return) when it detects an error or when `-help` or `-version` was specified, so the return value can safely (and should) be ignored.

When the exit behavior is overridden by calling [setMessageFunc\(\)](#) with a non-exiting function, then [parseCommandLine\(\)](#) returns the return value of the message function or 0 when the message function was not called (no error and no `-help/-version`).

Note that parse errors in integers or doubles are only identified by the respective `get*()` functions.

8.10.3.14 pleaseSpecifyOneOf()

```
void iris::IrisCommandLineParser::pleaseSpecifyOneOf (
    const std::vector< std::string > & options,
    const std::vector< std::string > & formalNonOptionArguments = std::vector< std::string >() )
```

Check whether at least one of the options or non-option-arguments are specified and exit with an error message if not. Call this for programs which require at least one of these options or arguments to be set. If `formalNonOptionArguments` is empty only options are checked.

8.10.3.15 printErrorAndExit() [1/2]

```
int iris::IrisCommandLineParser::printErrorAndExit (
    const std::exception & e ) const
```

Print error message and exit. Note that custom message functions may decide not to exit even on errors. In this case [parseCommandLine\(\)](#) returns the return value of the message function.

8.10.3.16 printErrorAndExit() [2/2]

```
int iris::IrisCommandLineParser::printErrorAndExit (
    const std::string & message ) const
```

Print error message and exit. Note that custom message functions may decide not to exit even on errors. In this case [parseCommandLine\(\)](#) returns the return value of the message function.

8.10.3.17 printMessage()

```
int iris::IrisCommandLineParser::printMessage (
    const std::string & message,
    int error = 0,
    bool exit = false ) const
```

Print message. This can be used by additional checks on the arguments to print warnings. This calls the message function set by [setMessageFunc\(\)](#) or the [defaultMessageFunc\(\)](#).

8.10.3.18 setMessageFunc()

```
void iris::IrisCommandLineParser::setMessageFunc (
    const std::function< int(const std::string &message, int error, bool exit)> &
    messageFunc )
```

Set custom message function which prints errors (`error!=0`), `-help` and `-version` messages (`error==0`) and which potentially also exit(s) (`exit==true`).

The default message function prints message on stdout and exits with "error" status if `exit==true`, else it returns error status.

Custom message functions may either exit, or they may return a value which is then returned by `parseCommandLine()` for errors raised by [parseCommandLine\(\)](#). For errors in the `get*()` functions the return value is ignored.

8.10.3.19 setValue()

```
void iris::IrisCommandLineParser::setValue (
    const std::string & longOption,
    const std::string & value,
    bool append = false )
```

Set/override command line option. By default overwrite the entire list for list options. Set `append=true` for list options to append to list.

8.10.3.20 unsetValue()

```
void iris::IrisCommandLineParser::unsetValue (
    const std::string & longOption )
```

Unset command line option. Set value to default value and mark as not specified.

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

- [IrisCommandLineParser.h](#)

8.11 iris::IrisEventEmitter< ARGS > Class Template Reference

A helper class for generating Iris events.

```
#include <IrisEventEmitter.h>
```

Inherits [IrisEventEmitterBase](#).

Public Member Functions

- [IrisEventEmitter](#) ()
Construct an event emitter.
- void [operator\(\)](#) (ARGS... args)
Emit an event.

8.11.1 Detailed Description

```
template<typename... ARGS>
class iris::IrisEventEmitter< ARGS >
```

A helper class for generating Iris events.

Template Parameters

<i>ARGS</i>	Argument types corresponding to the fields in this event.
-------------	---

Use [IrisEventEmitter](#) with [IrisInstanceBuilder](#) to add events to your Iris instance:

```
// Declare an event emitter
iris::IrisEventEmitter<uint64_t, bool> my_event;
// Add it to an Iris instance
iris::IrisInstance my_instance(...);
my_instance->getBuilder()->addEventSource("MY_EVENT", my_event)
    .addField("FOO", "uint", 8, "A value")
    .addField("FLAG", "bool", 1, "A flag");
// Emit an event
my_event(0x1234, true);
```

8.11.2 Member Function Documentation

8.11.2.1 operator()

```
template<typename... ARGS>
void iris::IrisEventEmitter< ARGS >::operator() (
    ARGS... args ) [inline]
```

Emit an event.

The arguments to this function are the fields of the event source, in the same order that they appear in the template arguments to the [IrisEventEmitter](#) class.

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

- [IrisEventEmitter.h](#)

8.12 iris::IrisEventRegistry Class Reference

Class to register Iris event streams for an event.

```
#include <IrisInstanceEvent.h>
```

Public Types

- typedef std::set< [EventStream](#) * >::const_iterator **iterator**

Public Member Functions

- template<class T >
void [addField](#) (const IrisU64StringConstant &field, const T &value) const
Add a field value.
- template<class T >
void [addFieldSlow](#) (const std::string &field, const T &value) const
Add a field value.
- iterator [begin](#) () const
Get an iterator to the beginning of the event stream set.
- void **emitEventBegin** (uint64_t time, uint64_t pc=IRIS_UINT64_MAX) const
- void [emitEventEnd](#) () const
Emit the callback.
- bool [empty](#) () const
Return true if no event streams are registered.
- iterator [end](#) () const
Get an iterator to the end of the event stream set.
- template<class T, typename F >
void [forEach](#) (F &&func) const
Call a function for each event stream.
- bool [registerEventStream](#) ([EventStream](#) *evStream)
Register an event stream.
- bool [unregisterEventStream](#) ([EventStream](#) *evStream)
Unregister an event stream.

8.12.1 Detailed Description

Class to register Iris event streams for an event.

8.12.2 Member Function Documentation

8.12.2.1 addField()

```
template<class T >
void iris::IrisEventRegistry::addField (
    const IrisU64StringConstant & field,
    const T & value ) const [inline]
```

Add a field value.

This is supported for all types supported by IrisU64JsonWriter and IrisObjects.h. Fast variant for argument names up to 23 chars. Use this if you can.

Template Parameters

<i>T</i>	The type of value.
----------	--------------------

Parameters

<i>field</i>	The name of the field whose value is set.
<i>value</i>	The value of the field.

8.12.2.2 addFieldSlow()

```
template<class T >
void iris::IrisEventRegistry::addFieldSlow (
    const std::string & field,
    const T & value ) const [inline]
```

Add a field value.

This is supported for all types supported by IrisU64JsonWriter and IrisObjects.h. Slow variant for argument names with more than 23 chars. Do not use unless you have to.

Template Parameters

<i>T</i>	The type of <i>value</i> .
----------	----------------------------

Parameters

<i>field</i>	The name of the field whose value is set.
<i>value</i>	The value of the field.

8.12.2.3 begin()

```
iterator iris::IrisEventRegistry::begin ( ) const [inline]
```

Get an iterator to the beginning of the event stream set.

See also

[end](#)

Returns

An iterator to the beginning of the event stream set.

8.12.2.4 emitEventEnd()

```
void iris::IrisEventRegistry::emitEventEnd ( ) const
```

Emit the callback.

This also checks the ranges and maintains the counter.

8.12.2.5 empty()

```
bool iris::IrisEventRegistry::empty ( ) const [inline]
```

Return true if no event streams are registered.

Returns

true if no event streams are registered.

8.12.2.6 end()

```
iterator iris::IrisEventRegistry::end ( ) const [inline]
```

Get an iterator to the end of the event stream set.

See also

[begin](#)

Returns

An iterator to the end of the event stream set.

8.12.2.7 forEach()

```
template<class T , typename F >
void iris::IrisEventRegistry::forEach (
    F && func ) const [inline]
```

Call a function for each event stream.

This function can be used as an alternative to [addField\(\)](#)/[addFieldSlow\(\)](#), when each event stream needs to be handled individually, for example because the event stream has options or because only selected fields should be emitted.

The main use-case of this function is to emit the fields of all event streams.

Example of an event source which optionally allows inverting its data: `class MyEventStream : public iris::IrisEventStream {...} IrisEventRegistry evreg;` In the callback set with (`IrisInstanceBuilder.addSource().`) `set← EventStreamCreateDelegate()` create a new event stream with `new MyEventStream(evreg);`

`// Emit event. evreg.emitEventBegin(time, pc); // Start building the callback data. evreg.forEach<MyEvent← Stream>([&](MyEventStream& es) { es.addField(ISTR("DATA"), es.invert ? ~data : data); }); evreg.emitEventEnd();`
`// Emit the callback.`

Template Parameters

<i>T</i>	Class derived from IrisEventStream .
<i>F</i>	Function to be called for each event stream (usually a lambda function).

8.12.2.8 registerEventStream()

```
bool iris::IrisEventRegistry::registerEventStream (
    EventStream * evStream )
```

Register an event stream.

Parameters

<i>evStream</i>	The stream to be registered.
-----------------	------------------------------

Returns

`true` if the stream was registered successfully.

8.12.2.9 unregisterEventStream()

```
bool iris::IrisEventRegistry::unregisterEventStream (
    EventStream * evStream )
```

Unregister an event stream.

Parameters

<i>evStream</i>	The stream to be unregistered.
-----------------	--------------------------------

Returns

`true` if the stream was unregistered successfully.

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

- [IrisInstanceEvent.h](#)

8.13 iris::IrisEventStream Class Reference

Event stream class for Iris-specific events.

`#include <IrisInstanceEvent.h>`

Inherits [iris::EventStream](#).

Public Member Functions

- virtual `IrisErrorCode` [disable](#) () IRIS_OVERRIDE
Disable this event stream.
- virtual `IrisErrorCode` [enable](#) () IRIS_OVERRIDE
Enable this event stream.
- **IrisEventStream** ([IrisEventRegistry](#) *registry_)

Additional Inherited Members

8.13.1 Detailed Description

Event stream class for Iris-specific events.

8.13.2 Member Function Documentation

8.13.2.1 disable()

```
virtual IrisErrorCode iris::IrisEventStream::disable ( ) [virtual]
```

Disable this event stream.

This function is only called when [isEnabled\(\)](#)/enabled == true. It is not necessary to verify this inside the [disable\(\)](#) method.

Returns

An error code indicating whether the event stream was successfully disabled. This should be `E_ok` if it was disabled or `E_error_disabling_event_stream` if it could not be disabled.

Implements [iris::EventStream](#).

8.13.2.2 enable()

```
virtual IrisErrorCode iris::IrisEventStream::enable ( ) [virtual]
```

Enable this event stream.

This function is only called when [isEnabled\(\)](#)/enabled == false. It is not necessary to verify this inside the [enable\(\)](#) method.

Returns

An error code indicating whether the event stream was successfully enabled. This should be E_ok if it was enabled or E_error_enabling_event_stream if it could not be enabled.

Implements [iris::EventStream](#).

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

- [IrisInstanceEvent.h](#)

8.14 iris::IrisGlobalInstance Class Reference

Inherits [IrisInterface](#), and [IrisConnectionInterface](#).

Public Member Functions

- void **emitLogMessage** (const std::string &message, const std::string &severityLevel)
- [IrisInstance](#) & [getIrisInstance](#) ()
- virtual [IrisInterface](#) * **getIrisInterface** () override
Get the IrisInterface for this connection.
- **IrisGlobalInstance** ()
Constructor.
- virtual void **irisHandleMessage** (const uint64_t *message) override
Handle incoming Iris messages.
- virtual [IrisErrorCode](#) **processAsyncMessages** (bool waitForAMessage) override
- uint64_t [registerChannel](#) ([IrisC_CommunicationChannel](#) *channel, const std::string &connectionInfo)
- virtual uint64_t [registerIrisInterfaceChannel](#) ([IrisInterface](#) *iris_interface, const std::string &connectionInfo) override
- virtual void **setIrisProxyInterface** ([IrisProxyInterface](#) *irisProxyInterface_) override
Set proxy interface.
- void **setLogLevel** (unsigned level)
- void [setLogMessageFunction](#) (std::function< [IrisErrorCode](#)(const std::string &, const std::string &)> func)
Set the function which will be called to log message for logger_logMessage Iris API.
- void **unregisterChannel** (uint64_t channelId)
Unregister a channel.
- virtual void [unregisterIrisInterfaceChannel](#) (uint64_t channelId) override
- **~IrisGlobalInstance** ()
Destructor.

8.14.1 Member Function Documentation

8.14.1.1 getIrisInstance()

```
IrisInstance & iris::IrisGlobalInstance::getIrisInstance ( ) [inline]
```

Get contained [IrisInstance](#). This can be used as a generic client instance to call Iris functions.

8.14.1.2 registerChannel()

```
uint64_t iris::IrisGlobalInstance::registerChannel (
    IrisC\_CommunicationChannel * channel,
    const std::string & connectionInfo )
```

Register a channel. Returns an associated channel id.

8.14.1.3 registerIrisInterfaceChannel()

```
virtual uint64_t iris::IrisGlobalInstance::registerIrisInterfaceChannel (
    IrisInterface * iris_interface,
    const std::string & connectionInfo ) [override], [virtual]
```

Register a local IrisInterface with the system. This allows it to receive messages (requests and responses). Returns the unique channelId used to identify this channel when registering instances.

8.14.1.4 setLogMessageFunction()

```
void iris::IrisGlobalInstance::setLogMessageFunction (
    std::function< IrisErrorCode(const std::string &, const std::string &)> func )
[inline]
```

Set the function which will be called to log message for logger_logMessage Iris API.

Parameters

<i>func</i>	A function object that will be called to log the message.
-------------	---

8.14.1.5 unregisterIrisInterfaceChannel()

```
virtual void iris::IrisGlobalInstance::unregisterIrisInterfaceChannel (
    uint64_t channelId ) [inline], [override], [virtual]
```

Unregister a previously registered channel. This will automatically unregister all instances associated with that channel.

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

- [IrisGlobalInstance.h](#)

8.15 iris::IrisInstance Class Reference

Public Types

- using [EventCallbackFunction](#) = std::function< IrisErrorCode(EventStreamId, const IrisValueMap &, uint64_t, InstanceId, bool, std::string &)>

Public Member Functions

- void [addCallback_IRIS_INSTANCE_REGISTRY_CHANGED](#) ([EventCallbackFunction](#) f)
- void [clearCachedMetaInfo](#) ()

Clear cached meta-information including the list of InstanceInfos for all instances in the system.
- void [destroyAllEventStreams](#) ()

Destroy all event streams.
- void [disableEvent](#) (const std::string &eventSpec)

Disable all matching event callback(s).
- void [enableEvent](#) (const std::string &eventSpec, std::function< void()> callback, bool syncEc=false)

Enable event callback(s).
- void [enableEvent](#) (const std::string &eventSpec, std::function< void(const EventStreamInfo &eventStreamInfo, IrisReceivedRequest &request)> callback, bool syncEc=false)

Enable event callback(s).
- void [eventBufferDestroyed](#) (EventBufferId evBufId)

Notify instance that a specific event buffer was just destroyed.
- std::vector< EventSourceInfo > [findEventSources](#) (const std::string &instancePathFilter="all")

Find all event sources in the system.

- `std::vector< EventStreamInfo > findEventSourcesAndFields` (const std::string &spec, InstanceId defaultInstId=IRIS_UINT64_MAX)
Find specific event sources in the system.
- `void findEventSourcesAndFields` (const std::string &spec, std::vector< EventStreamInfo > &eventStreamInfosOut, InstanceId defaultInstId=IRIS_UINT64_MAX)
- `std::vector< InstanceInfo > findInstanceInfos` (const std::string &instancePathFilter="all")
Find instance infos of all instances in the system.
- `IrisInstanceBuilder * getBuilder` ()
Get the [IrisInstanceBuilder](#) object for this instance. This can be used to set up metadata and callbacks for standard Iris functions.
- `const std::vector< EventSourceInfo > & getEventSourceInfosOfAllInstances` ()
Find all event sources of all instances in the system.
- `InstanceId getInstId` (const std::string &instName)
Get instance id for a specific instance name.
- `InstanceInfo getInstInfo` (const std::string &instancePathFilter)
Get instance info of a specific instance in the system.
- `const InstanceInfo & getInstInfo` (InstanceId instId)
Get InstanceInfo including properties for a specific instId.
- `const std::vector< InstanceInfo > & getInstanceList` ()
Get list of InstanceInfos of all instances in the system, including properties.
- `const std::string & getInstanceName` () const
Get the instance name of this instance. This is valid after [registerInstance\(\)](#) returns.
- `std::string getInstanceName` (InstanceId instId)
Get instance name for a specific instId.
- `InstanceId getInstId` () const
Get the instance id of this instance. This is valid after [registerInstance\(\)](#) returns.
- `IrisInterface * getLocalIrisInterface` ()
Get the local IrisInterface of this instance. This is the interface that other instances use to send their requests and responses to this instance.
- `MemorySpaceId getMemorySpaceId` (InstanceId instId, const std::string &name)
Get memory space id of memory space by name.
- `MemorySpaceId getMemorySpaceId` (InstanceId instId, uint64_t canonicalMsn)
Get memory space id of memory space identified by its canonical memory space number (e.g. CanonicalMsnArm_ constant).*
- `const MemorySpaceInfo & getMemorySpaceInfo` (InstanceId instId, const std::string &name)
Get MemorySpaceInfo of memory space by name.
- `const MemorySpaceInfo & getMemorySpaceInfo` (InstanceId instId, uint64_t canonicalMsn)
Get MemorySpaceInfo of memory space identified by its canonical memory space number (e.g. CanonicalMsnArm_ constant).*
- `const std::vector< MemorySpaceInfo > & getMemorySpaceInfos` (InstanceId instId)
Get list of MemorySpaceInfos.
- `const PropertyMap & getPropertyMap` () const
Get property map.
- `IrisInterface * getRemoteIrisInterface` ()
Get the remote Iris interface.
- `const std::vector< ResourceGroupInfo > & getResourceGroups` (InstanceId instId)
Get list of resource groups.
- `ResourceId getResourceId` (InstanceId instId, const std::string &resourceSpec)
Get resource id for a specific resource.
- `const ResourceInfo & getResourceInfo` (InstanceId instId, const std::string &resourceSpec)
Get ResourceInfo for a specific resource.
- `const ResourceInfo & getResourceInfo` (InstanceId instId, ResourceId resourceId)

- Get ResourceInfo for a specific resource.*

 - `const std::vector< ResourceInfo > & getResourceInfos (InstanceId instId)`

Get list of resource infos.
- `IrisCppAdapter & irisCall ()`

Get an IrisCppAdapter to call an Iris function of any other instance.
- `IrisCppAdapter & irisCallNoThrow ()`

Get an IrisCppAdapter to call an Iris function of any other instance.
- `IrisCppAdapter & irisCallThrow ()`

Get an IrisCppAdapter to call an Iris function of any other instance. When an Iris function returns an error response, this adapter always throws an exception. Usage:
- `IrisInstance (IrisConnectionInterface *connection_interface=nullptr, const std::string &instName=std::string(), uint64_t flags=DEFAULT_FLAGS)`

Construct a new Iris instance.
- `IrisInstance (IrisInstantiationContext *context)`

Construct a new Iris instance using an [IrisInstantiationContext](#).
- `bool isAdapterInitialized () const`
- `bool isRegistered () const`
- `bool isValidEvBufId (EventBufferId evBufId) const`
- Check whether event buffer id is valid.*
- `void notifyStateChanged ()`
- Notify client instances that the state of any resource/memory/table/disassembly etc changed.*
- `void processAsyncRequests ()`
- Process async requests. Use this to keep the Iris system running while a thread is blocked waiting for something.*
- `template<class T >`
`void publishCppInterface (const std::string &interfaceName, T *pointer, const std::string &jsonDescription)`
Publish a C++ interface XYZ through a new instance `_getCppInterfaceXYZ()` function.
- `void registerEventBufferCallback (EventBufferCallbackDelegate delegate, const std::string &name, const std::string &description, const std::string &dlgInstanceTypeStr)`

Register an event buffer callback using an [EventBufferCallbackDelegate](#).
- `template<typename T, IrisErrorCode(T::*)(const EventBufferCallbackData &data) METHOD>`
`void registerEventBufferCallback (T *instance, const std::string &name, const std::string &description, const std::string &dlgInstanceTypeStr)`
Register an event buffer callback using an [EventBufferCallbackDelegate](#).
- `template<class T >`
`void registerEventBufferCallback (T *instance, const std::string &name, const std::string &description, void(T::*memberFunctionPtr)(IrisReceivedRequest &), const std::string &instanceTypeStr)`
Register an event buffer callback function.
- `void registerEventCallback (EventCallbackDelegate delegate, const std::string &name, const std::string &description, const std::string &dlgInstanceTypeStr)`

Register a general event callback using an [EventCallbackDelegate](#).
- `template<typename T, IrisErrorCode(T::*)(uint64_t, const AttributeValueMap &, uint64_t, uint64_t, bool, std::string &) METHOD>`
`void registerEventCallback (T *instance, const std::string &name, const std::string &description, const std::string &dlgInstanceTypeStr)`
Register a general event callback using an [EventCallbackDelegate](#).
- `template<class T >`
`void registerEventCallback (T *instance, const std::string &name, const std::string &description, void(T::*memberFunctionPtr)(IrisReceivedRequest &), const std::string &instanceTypeStr)`
Register a general event callback.
- `template<class T >`
`void registerFunction (T *instance, const std::string &name, void(T::*memberFunctionPtr)(IrisReceivedRequest &), const std::string &functionInfoJson, const std::string &instanceTypeStr)`
Register an Iris function implementation.
- `IrisErrorCode registerInstance (const std::string &instName, uint64_t flags=DEFAULT_FLAGS)`

- Register this instance if it was not registered when constructed.*

 - uint64_t [resourceRead](#) (InstanceId instId, const std::string &resourceSpec)
Read numeric resource and return its value.
 - uint64_t [resourceReadCrn](#) (InstanceId instId, uint64_t canonicalRegisterNumber)
Read numeric resource and return its value (using the canonical register number aka DWARF register id).
 - std::string [resourceReadStr](#) (InstanceId instId, const std::string &resourceSpec)
Read string resource, or read other resources as string.
 - void [resourceWrite](#) (InstanceId instId, const std::string &resourceSpec, uint64_t value)
Write numeric resource.
 - void [resourceWriteCrn](#) (InstanceId instId, uint64_t canonicalRegisterNumber, uint64_t value)
Write numeric resource by canonical register number (aka DWARF register id).
 - void [resourceWriteStr](#) (InstanceId instId, const std::string &resourceSpec, const std::string &value)
Write string resource, or write numeric resource from string.
 - bool [sendRequest](#) (IrisRequest &req)
Send an Iris request or notification and potentially wait for a response.
 - void [sendResponse](#) (const uint64_t *response)
Send a response to the remote Iris interface.
 - void [setAdapterInitialized](#) ()
 - void [setCallback_IRIS_SHUTDOWN_LEAVE](#) (EventCallbackFunction f)
 - void [setCallback_IRIS_SIMULATION_TIME_EVENT](#) (EventCallbackFunction f)
 - void [setConnectionInterface](#) (IrisConnectionInterface *connection_interface)
Set the remote connection interface.
 - void [setEventHandler](#) (IrisInstanceEvent *handler)
Set the event handler.
 - void [setInstId](#) (InstanceId instId)
Internal function. Do not call. Set the instance id of this instance. The instId is automatically set after calling instanceRegistry.registerInstance().
 - void [setPendingSyncStepResponse](#) (RequestId requestId)
Set pending response to a step_syncStep() call.
 - template<class T >
void [setProperty](#) (const std::string &propertyName, const T &propertyValue)
Set/add instance property.
 - bool [setSyncStepEventBufferId](#) (EventBufferId evBufId)
Set event buffer to use with step_syncStep() call.
 - void [setThrowOnError](#) (bool throw_on_error)
Set default error behavior for irisCall().
 - void [simulationTimeDisableEvents](#) ()
Disable the internal reception of IRIS_SIMULATION_TIME_EVENT events for performance reasons (e.g. during synchronous stepping).
 - bool [simulationTimeIsRunning](#) ()
Return true iff simulation is currently running.
 - void [simulationTimeRun](#) ()
Run simulation time and wait until simulation time started running.
 - void [simulationTimeRunUntilStop](#) (double timeoutInSeconds=0.0)
Run simulation time and wait until simulation time stopped again or until timeout expired.
 - void [simulationTimeStop](#) ()
Stop simulation time and wait until simulation time stopped.
 - bool [simulationTimeWaitForStop](#) (double timeoutInSeconds=0.0)
Wait for simulation time to stop or timeout.
 - void [unpublishCppInterface](#) (const std::string &interfaceName)
Unpublish a previously published C++ interface.

- void **unregisterEventBufferCallback** (const std::string &name)
Unregister the named event buffer callback function.
- void **unregisterEventCallback** (const std::string &name)
Unregister the named event callback function.
- void **unregisterFunction** (const std::string &name)
Unregister a function that was previously registered with [registerFunction\(\)](#) or [irisRegisterFunction\(\)](#).
- IrisErrorCode **unregisterInstance** ()
Unregister this instance.
- **~IrisInstance** ()
Destructor.

Static Public Attributes

- static const uint64_t **DEFAULT_FLAGS** = [THROW_ON_ERROR](#)
Default flags used if not otherwise specified.
- static const bool **SYNCHRONOUS** = true
Cause [enableEvent\(\)](#) callback to be called back synchronously (i.e. the caller is blocked until the callback function returns).
- static const uint64_t **THROW_ON_ERROR** = (1 << 1)
Throw an exception when an Iris call returns an error response.
- static const uint64_t **UNIQUEIFY** = (1 << 0)
Uniquify instance name when registering.

Protected Attributes

- InstanceInfo **thisInstanceInfo** {}
InstanceInfo of this instance.

8.15.1 Member Typedef Documentation

8.15.1.1 EventCallbackFunction

```
using iris::IrisInstance::EventCallbackFunction = std::function<IrisErrorCode(EventStreamId,
const IrisValueMap&, uint64_t, InstanceId, bool, std::string&)>
```

Event callback function type.

(Each [IrisInstance](#) can implicitly register two events which are used internally (IRIS_SIMULATION_TIME_EVENT and IRIS_SHUTDOWN_LEAVE). Using the functions below clients can make use of these events without going through the effort of calling [irisRegisterEventCallback\(\)](#)/registerEventCallback(), event_getEventSource() and eventStream_create(), and it also reduces the number of callbacks being called at runtime.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 IrisInstance() [1/2]

```
iris::IrisInstance::IrisInstance (
    IrisConnectionInterface * connection_interface = nullptr,
    const std::string & instName = std::string(),
    uint64_t flags = DEFAULT\_FLAGS )
```

Construct a new Iris instance.

Parameters

<i>connection_interface</i>	The IrisConnectionInterface that this instance should use to connect to the simulation.
<i>instName</i>	Name of the instance. This should be prefixed with one of the following, as appropriate: <ul style="list-style-type: none"> • "client." • "component." • "framework."
<i>flags</i>	A bitwise OR of Instance Flags . Client instances should usually set the flag iris::IrisInstance::UNQUIFY .

8.15.2.2 IrisInstance() [2/2]

```
iris::IrisInstance::IrisInstance (
    IrisInstantiationContext * context )
```

Construct a new Iris instance using an [IrisInstantiationContext](#).

Parameters

<i>context</i>	A context object that provides the necessary information to instantiate an instance.
----------------	--

8.15.3 Member Function Documentation

8.15.3.1 addCallback_IRIS_INSTANCE_REGISTRY_CHANGED()

```
void iris::IrisInstance::addCallback_IRIS_INSTANCE_REGISTRY_CHANGED (
    EventCallbackFunction f )
```

Add callback function for IRIS_INSTANCE_REGISTRY_CHANGED.

8.15.3.2 destroyAllEventStreams()

```
void iris::IrisInstance::destroyAllEventStreams ( )
```

Destroy all event streams.

All event streams are always automatically destroyed when [IrisInstance](#) (and so [IrisInstanceEvent](#)) is destroyed. This function allows to destroy all event streams to be destroyed before [IrisInstance](#).

8.15.3.3 disableEvent()

```
void iris::IrisInstance::disableEvent (
    const std::string & eventSpec )
```

Disable all matching event callback(s).

This disables all event callbacks which were previously enabled using [enableEvent\(\)](#) which match eventSpec. The eventSpec argument for [enableEvent\(\)](#) and [disableEvent\(\)](#) do not have to be the same string. In particular it is not necessary to specify event fields and it is not possible to selectively disable one specific event stream out of multiple created for the same event source.

[disableEvent\(\)](#) always iterates over all currently active event streams and disables all event streams which originate from the event sources specified in eventSpec.

Example: // Handle INST of cpu0 and cpu1 in different ways. `irisInstance.enableEvent("*.cpu0.INST", [&] (const EventStreamInfo& eventStreamInfo, IrisReceivedRequest& request) { ... }); irisInstance.enableEvent("*.cpu1.INST", [&] (const EventStreamInfo& eventStreamInfo, IrisReceivedRequest& request) { ... }); // Disable just the cpu1 events. irisInstance.disableEvent("*.cpu1.INST");`

8.15.3.4 enableEvent() [1/2]

```
void iris::IrisInstance::enableEvent (
    const std::string & eventSpec,
    std::function< void()> callback,
    bool syncEc = false )
```

Enable event callback(s).

This is equivalent to [enableEvent\(\)](#) specified above except that the callback does not take any arguments which is useful for the global simulation phase events.

Example:

Initialize a plugin or client in the SystemC end_of_elaboration() phase. This is the phase when all other instances are initialized and can be inspected. `irisInstance.enableEvent("IRIS_SIM_PHASE_END_OF_ELABORATION", [&] { ... enable trace (using enableTrace()), inspect other instances, etc ... }, iris::IrisInstance::SYNCHRONOUS);`

8.15.3.5 enableEvent() [2/2]

```
void iris::IrisInstance::enableEvent (
    const std::string & eventSpec,
    std::function< void(const EventStreamInfo &eventStreamInfo, IrisReceivedRequest
&request)> callback,
    bool syncEc = false )
```

Enable event callback(s).

Create one or more event streams and set up the callback function to be called for all events on the event streams. If no event stream is created because no event source matching spec is found, or if an error occurred when create an events stream, an error is thrown.

Calling this function multiple times matching the same event source is valid, but it results in multiple event streams being created which should usually be avoided for performance reasons.

A new unique callback function with the name `ec_i<instanceId>_<eventSourceName>[N]` is registered, where N is used to make the function name different from all other functions. This is name usually not of interest for the usage of this function.

Parameters

<i>eventSpec</i>	This specifies one or more event source names of one or more instances. See findEventSourcesAndFields() for the syntax specification. When the instance part of an event source is omitted the global instance is assumed. Passing "help" will throw an <code>E_help_message</code> error with a help messages describing the syntax and listing all available event sources in the system.
<i>callback</i>	Callback function called for every event. Usually a lambda function.
<i>syncEc</i>	If true, call callback function synchronously (i.e. caller waits for return of the callback function). Useful for simulation phases.

Examples:

Initialize a plugin or client in the SystemC end_of_elaboration() phase. This is the phase when all other instances are initialized and can be inspected. Every plugin usually does this in its constructor to enable other traces in the end_of_elaboration() phase. `irisInstance.enableEvent("IRIS_SIM_PHASE_END_OF_ELABORATION", [&] { // Enable traces, inspect other instances. irisInstance.enableEvent("*.INST", [&] (const EventStreamInfo& eventStreamInfo, IrisReceivedRequest& request) { ... handle INST trace ... }); }, iris::IrisInstance::SYNCHRONOUS);`

Print all simulation phases as they happen: `irisInstance.enableEvent("IRIS_SIM_PHASE_*:IRIS_SHUTDOWN_*", [&](const iris::EventStreamInfo& eventStreamInfo, iris::IrisReceivedRequest&) { std::cout << eventStreamInfo.↵eventSourceInfo.name << "\n"; }, iris::IrisInstance::SYNCHRONOUS);`

Receive INST callbacks from all cores: `irisInstance.enableEvent("*.INST", [&] (const EventStreamInfo& event↵StreamInfo, IrisReceivedRequest& request) { ... });`

See also `Examples/Plugin/SimpleTrace/main.cpp` and `Examples/Plugin/GenericTrace/main.cpp`.

This may throw:

- `E_syntax_error`: Syntax error in spec (like missing closing parenthesis).
- `E_unknown_event_source`: A pattern in `EVENT_SOURCE` in `eventSpec` did not match any instance and/or

event source name.

- `E_unknown_event_field`: A pattern in `FIELD_OR_OPTION` in `eventSpec` did not match any field or option of its event source.

8.15.3.6 eventBufferDestroyed()

```
void iris::IrisInstance::eventBufferDestroyed (
    EventBufferId evBufId )
```

Notify instance that a specific event buffer was just destroyed.

This function is called when a client disconnects because then all event buffers and event streams associated with that client are destroyed. It is also called by `eventBuffer_destroy()`.

This function clears a pending `SyncStepResponse` if this uses the destroyed event buffer. It also clears the `evBufId` cached in `IrisInstanceStep` if it uses the destroyed event buffer.

8.15.3.7 findEventSources()

```
std::vector< EventSourceInfo > iris::IrisInstance::findEventSources (
    const std::string & instancePathFilter = "all" )
```

Find all event sources in the system.

See `filterInstanceInfos()` (`IrisObjects.h`) for `instancePathFilter` semantics.

8.15.3.8 findEventSourcesAndFields()

```
std::vector< EventStreamInfo > iris::IrisInstance::findEventSourcesAndFields (
    const std::string & spec,
    InstanceId defaultInstId = IRIS_UINT64_MAX )
```

Find specific event sources in the system.

Find all event sources in the system and/or in the instance defined by `defaultInstId` matching wildcard patterns.

All matching event sources are added to `eventStreamInfosOut` which is not cleared beforehand.

The following fields in each `EventStreamInfo` element are set to the meta-info of the events source: `sInstId`, `evSrcId`, `evSrcName`, `fields`, `hasFields` and `eventSourceInfo`.

No event streams are created. The output is suitable as the `eventStreamInfos` argument for `eventBuffer_create()`. Alternatively, individual event streams can be created using `eventStream_create()` by looping over `eventStreamInfosOut`.

The set of returned event sources is defined by the filters specified in "spec" which has the following format:

- `[~]EVENT_SOURCE ["(" [FIELD_OR_OPTION ["+" FIELD_OR_OPTION] ...] ")"] [":" ...]`
- `EVENT_SOURCE` is a wildcard pattern matching on strings of the form `<instance_path>.<event_source_name>` (for all instances in the system) and on strings `<event_source_name>` for event sources of default `InstId`.
- `FIELD_OR_OPTION` is either a wildcard pattern matching on field names of the selected event sources, or it is of the format `OPT=VAL` setting option `OPT` to value `VAL`. Use `(+OPT=VAL)` to set option and still emit all fields.
- Use `~EVENT_SOURCE` to remove any previously matched event sources. The adding and removing event sources is executed in the specified order, so usually removes should come at the end. This makes it easy to enable events using wildcards and then exclude certain events. Example: `*:~*UTLB`: Enable all events in the system except all UTLB related events.
- Likewise, use `~FIELD` to remove any previously selected fields. When the first `FIELD` is a negative field matching starts with all fields.

Examples:

- `INST` (Trace `INST` on the selected core.)
" - *.INST:*.CORE_STORES (Trace `INST` and `CORE_STORES` on all cores.)\n"

- *.INST(PC+DISASS) (Only trace PC and disassembly of INST.)
" - *.INST(~DISASS) (Trace all fields except disassembly of INST.)\n"
- *:~*SEMIHOSTING*:~*UTLB* (Enable all trace sources in the whole system except semihosting and UTLB related traces.)
" - *.TRACE_DATA_FMT_V1_1(+bufferSize=1048576) (Enable trace stream in FMT V1.1 format with buffer size 1MB and all fields.)\n\n";

This may throw:

- E_syntax_error: Syntax error in spec (like missing closing parenthesis).
- E_unknown_event_source: A pattern in EVENT_SOURCE in spec did not match any instance and/or event source name.
- E_unknown_event_field: A pattern in FIELD_OR_OPTION in spec did not match any field or option of its event source.

8.15.3.9 findInstanceInfos()

```
std::vector< InstanceInfo > iris::IrisInstance::findInstanceInfos (
    const std::string & instancePathFilter = "all" )
```

Find instance infos of all instances in the system.

This function uses instance info data cached in this instance. The cache can be cleared with [clearCachedMetaInfo\(\)](#). See [filterInstanceInfos\(\)](#) (IrisObjects.h) for instancePathFilter semantics.

8.15.3.10 getBuilder()

```
IrisInstanceBuilder * iris::IrisInstance::getBuilder ( )
```

Get the [IrisInstanceBuilder](#) object for this instance. This can be used to set up metadata and callbacks for standard Iris functions.

Returns

The [IrisInstanceBuilder](#) object for this instance.

8.15.3.11 getInstanceId()

```
InstanceId iris::IrisInstance::getInstanceId (
    const std::string & instName )
```

Get instance id for a specifid instance name.

If no such instance is known [IrisErrorException\(E_unknown_instance_name\)](#) is thrown.

This information is cached in this instance. The cache can be cleared with [clearCachedMetaInfo\(\)](#).

Returns

Instance id.

8.15.3.12 getInstanceInfo() [1/2]

```
InstanceInfo iris::IrisInstance::getInstanceInfo (
    const std::string & instancePathFilter )
```

Get instance info of a specific instance in the system.

This function expects either a correct instance path or a pattern which just matches a single instance, for example "core" which always returns the first core, regardless of the number of cores in the system. If no instance is found or if more than one instances are found, [IrisErrorException\(E_unknown_instance_name\)](#) is thrown.

This function should only be used when the instance name is known upfront, or to get access to the first core only. Use [findInstanceInfos\(\)](#) to discover arbitrary instances.

This function uses instance info data cached in this instance. The cache can be cleared with [clearCachedMetaInfo\(\)](#). See [filterInstanceInfos\(\)](#) (IrisObjects.h) for instancePathFilter semantics.

8.15.3.13 getInstanceInfo() [2/2]

```
const InstanceInfo & iris::IrisInstance::getInstanceInfo (
    InstanceId instId )
```

Get InstanceInfo including properties for a specific instId.

This information is cached in this instance. The cache can be cleared with [clearCachedMetaInfo\(\)](#).

Returns

InstanceInfo (including properties) for instId. Throws IrisErrorException(E_unknown_instance_id) if instId is unknown.

8.15.3.14 getInstanceList()

```
const std::vector< InstanceInfo > & iris::IrisInstance::getInstanceList ( )
```

Get list of InstanceInfos of all instances in the system, including properties.

Note that the index into the returned list is generally not the InstanceId. Use getInstanceInfo(instId) to get the InstanceInfo for a specific instance id.

This information is cached in this instance. The cache can be cleared with [clearCachedMetaInfo\(\)](#).

Returns

InstanceInfos (including properties) for all instances in the system.

8.15.3.15 getInstanceName() [1/2]

```
const std::string & iris::IrisInstance::getInstanceName ( ) const [inline]
```

Get the instance name of this instance. This is valid after [registerInstance\(\)](#) returns.

Returns

The instance name of this instance. This is the same as the name parameter passed to the constructor or [registerInstance\(\)](#) unless this instance was registered with the UNQUIFY flag set and the name was modified to make it unique.

8.15.3.16 getInstanceName() [2/2]

```
std::string iris::IrisInstance::getInstanceName (
    InstanceId instId )
```

Get instance name for a specifid instId.

This function does not throw. It returns "instance.<instId>" for unknown instIds.

This information is cached in this instance. The cache can be cleared with [clearCachedMetaInfo\(\)](#).

Returns

instance name or "instance.<instId>" instId is unknown.

8.15.3.17 getInstId()

```
InstanceId iris::IrisInstance::getInstId ( ) const [inline]
```

Get the instance id of this instance. This is valid after [registerInstance\(\)](#) returns.

Returns

The instId for this instance.

8.15.3.18 getLocalIrisInterface()

```
IrisInterface * iris::IrisInstance::getLocalIrisInterface ( ) [inline]
```

Get the local IrisInterface of this instance. This is the interface that other instances use to send their requests and responses to this instance.

Returns

IrisInterface to send messages to this instance.

8.15.3.19 getMemorySpaceId()

```
MemorySpaceId iris::IrisInstance::getMemorySpaceId (
    InstanceId instId,
    const std::string & name )
```

Get memory space id of memory space by name.

Note: Memory space names change over time and are not a stable method to identify memory spaces. If possible the canonical memory space number should be used instead to identify memory spaces.

8.15.3.20 getMemorySpaceInfo()

```
const MemorySpaceInfo & iris::IrisInstance::getMemorySpaceInfo (
    InstanceId instId,
    const std::string & name )
```

Get MemorySpaceInfo of memory space by name.

Note: Memory space names change over time and are not a stable method to identify memory spaces. If possible the canonical memory space number should be used instead to identify memory spaces.

8.15.3.21 getPropertyMap()

```
const PropertyMap & iris::IrisInstance::getPropertyMap ( ) const [inline]
```

Get property map.

This can be used to lookup properties: `getWithDefault(my_instance->getPropertyMap(), "myStringProperty", "").getAsString();`

8.15.3.22 getRemoteIrisInterface()

```
IrisInterface * iris::IrisInstance::getRemoteIrisInterface ( ) [inline]
```

Get the remote Iris interface.

Returns

Returns the IrisInterface that this instance sends requests and responses to.

8.15.3.23 getResourceId()

```
ResourceId iris::IrisInstance::getResourceId (
    InstanceId instId,
    const std::string & resourceSpec )
```

Get resource id for a specific resource.

See [resourceRead\(\)](#) for semantics of resourceSpec.

Throws an error when resource is not found.

Returns

Resource id.

8.15.3.24 irisCall()

```
IrisCppAdapter & iris::IrisInstance::irisCall ( ) [inline]
```

Get an IrisCppAdapter to call an Iris function of any other instance.

Usage:

```
irisCall().resource_read(...);
```

for the Iris function `resource_read()`.

8.15.3.25 irisCallNoThrow()

```
IrisCppAdapter & iris::IrisInstance::irisCallNoThrow ( ) [inline]
```

Get an IrisCppAdapter to call an Iris function of any other instance.

When an Iris function returns an error response, this adapter returns the error code and does not throw an exception.

Usage:

```
iris::IrisErrorCode code = irisCallNoThrow().resource_read(...);
```

8.15.3.26 irisCallThrow()

```
IrisCppAdapter & iris::IrisInstance::irisCallThrow ( ) [inline]
```

Get an IrisCppAdapter to call an Iris function of any other instance. When an Iris function returns an error response, this adapter always throws an exception. Usage:

```
try
{
    irisCall().resource_read(...);
}
catch (iris::IrisErrorException &e)
{
    ...
}
```

8.15.3.27 isRegistered()

```
bool iris::IrisInstance::isRegistered ( ) const [inline]
```

Return true iff we are registered as an instance (= we have a valid instance id).

8.15.3.28 isValidEvBufId()

```
bool iris::IrisInstance::isValidEvBufId (
    EventBufferId evBufId ) const
```

Check whether event buffer id is valid.

This function is use to validate event buffer ids.

Returns

Returns true iff evBufId is a valid event buffer id.

8.15.3.29 notifyStateChanged()

```
void iris::IrisInstance::notifyStateChanged ( )
```

Notify client instances that the state of any resource/memory/table/disassembly etc changed.

This should only ever be called when the value of anything changes spontaneously, e.g. through a private GUI of an instance. This must not be called when the state changes because of normal simulation operations.

Calling this function is very exotic. Normal component instances and client instances will never want to call this.

8.15.3.30 publishCppInterface()

```
template<class T >
void iris::IrisInstance::publishCppInterface (
    const std::string & interfaceName,
    T * pointer,
    const std::string & jsonDescription ) [inline]
```


Publish a C++ interface XYZ through a new instance `_getCplusplusInterfaceXYZ()` function.
Null pointers are silently ignored. An interface previously registered under the same name is silently overwritten.

Parameters

<i>interfaceName</i>	Class name or interface name of the interface to be published. This must be a C identifier without namespaces etc. The interface can be retrieved with <code>"instance_getCplusplusInterface<interfaceName>()"</code> .
<i>pointer</i>	Pointer to the C++ class instance implementing this interface.
<i>jsonDescription</i>	Text for <code>FunctionInfo.description</code> . This must be a valid JSON string without enclosing quotes. This text is amended by generic notes about the compatibility of C++ pointers which are valid for every C++ interface.

8.15.3.31 registerEventBufferCallback() [1/3]

```
void iris::IrisInstance::registerEventBufferCallback (
    EventBufferCallbackDelegate delegate,
    const std::string & name,
    const std::string & description,
    const std::string & dlgInstanceTypeStr ) [inline]
```

Register an event buffer callback using an `EventBufferCallbackDelegate`.

Parameters

<i>delegate</i>	<code>EventBufferCallbackDelegate</code> to call to handle the function.
<i>name</i>	Name of the function as it will be published.
<i>description</i>	Description of this event callback function.
<i>dlgInstanceTypeStr</i>	The name of the delegate type. This is only used for logging purposes.

8.15.3.32 registerEventBufferCallback() [2/3]

```
template<typename T , IrisErrorCode(T::*)(const EventBufferCallbackData &data) METHOD>
void iris::IrisInstance::registerEventBufferCallback (
    T * instance,
    const std::string & name,
    const std::string & description,
    const std::string & dlgInstanceTypeStr ) [inline]
```

Register an event buffer callback using an `EventBufferCallbackDelegate`.

Parameters

<i>instance</i>	An instance of class T on which to call the delegate <code>T::METHOD()</code> .
<i>name</i>	Name of the function as it will be published.
<i>description</i>	Description of this event callback function.
<i>dlgInstanceTypeStr</i>	The name of the delegate type. This is only used for logging purposes.

8.15.3.33 registerEventBufferCallback() [3/3]

```
template<class T >
void iris::IrisInstance::registerEventBufferCallback (
```

```

T * instance,
const std::string & name,
const std::string & description,
void(T::*)(IrisReceivedRequest &) memberFunctionPtr,
const std::string & instanceTypeStr ) [inline]

```

Register an event buffer callback function.

Event buffer callbacks have the same signature, only the description is different.

Parameters

<i>instance</i>	An instance of class T on which to call the member function.
<i>name</i>	Name of the function as it will be published.
<i>description</i>	Description of this event callback function.
<i>memberFunctionPtr</i>	Pointer to the C++ implementation of the function.
<i>instanceTypeStr</i>	The name of class T. This is only used for logging purposes.

8.15.3.34 registerEventCallback() [1/3]

```

void iris::IrisInstance::registerEventCallback (
    EventCallbackDelegate delegate,
    const std::string & name,
    const std::string & description,
    const std::string & dlgInstanceTypeStr ) [inline]

```

Register a general event callback using an EventCallbackDelegate.

Parameters

<i>delegate</i>	EventCallbackDelegate to call to handle the function.
<i>name</i>	Name of the function as it will be published.
<i>description</i>	Description of this event callback function.
<i>dlgInstanceTypeStr</i>	The name of the delegate type. This is only used for logging purposes.

8.15.3.35 registerEventCallback() [2/3]

```

template<typename T , IrisErrorCode(T::*)(uint64_t, const AttributeValueMap &, uint64_t, uint64_t,
_t, bool, std::string &) METHOD>
void iris::IrisInstance::registerEventCallback (
    T * instance,
    const std::string & name,
    const std::string & description,
    const std::string & dlgInstanceTypeStr ) [inline]

```

Register a general event callback using an EventCallbackDelegate.

Parameters

<i>instance</i>	An instance of class T on which to call the delegate T::METHOD().
<i>name</i>	Name of the function as it will be published.
<i>description</i>	Description of this event callback function.
<i>dlgInstanceTypeStr</i>	The name of the delegate type. This is only used for logging purposes.

8.15.3.36 registerEventCallback() [3/3]

```
template<class T >
void iris::IrisInstance::registerEventCallback (
    T * instance,
    const std::string & name,
    const std::string & description,
    void(T::*)(IrisReceivedRequest &) memberFunctionPtr,
    const std::string & instanceTypeStr ) [inline]
```

Register a general event callback.

Event callbacks have the same signature, only the description is different.

Parameters

<i>instance</i>	An instance of class T on which to call the member function.
<i>name</i>	Name of the function as it will be published.
<i>description</i>	Description of this event callback function.
<i>memberFunctionPtr</i>	Pointer to the C++ implementation of the function.
<i>instanceTypeStr</i>	The name of class T. This is only used for logging purposes.

8.15.3.37 registerFunction()

```
template<class T >
void iris::IrisInstance::registerFunction (
    T * instance,
    const std::string & name,
    void(T::*)(IrisReceivedRequest &) memberFunctionPtr,
    const std::string & functionInfoJson,
    const std::string & instanceTypeStr ) [inline]
```

Register an Iris function implementation.

The following macro can be used instead of calling this function to avoid specifying the function name twice↵
: [irisRegisterFunction\(instancePtr, instanceType, functionName, functionInfoJson\)](#)

Parameters

<i>instance</i>	An instance of class T on which to call the member function.
<i>name</i>	Name of the function as it will be published.
<i>memberFunctionPtr</i>	Pointer to the C++ implementation of the function.
<i>functionInfoJson</i>	A string containing the JSON-encoded FunctionInfo object for this function.
<i>instanceTypeStr</i>	The name of class T. This is only used for logging purposes.

8.15.3.38 registerInstance()

```
IrisErrorCode iris::IrisInstance::registerInstance (
    const std::string & instName,
    uint64_t flags = DEFAULT\_FLAGS )
```

Register this instance if it was not registered when constructed.

Parameters

<i>instName</i>	Name of the instance. This should be prefixed with one of the following, as appropriate: <ul style="list-style-type: none"> • "client." • "component." • "framework."
<i>flags</i>	A bitwise OR of Instance Flags . Client instances should usually set the flag iris::IrisInstance::UNQUIFY .

8.15.3.39 resourceRead()

```
uint64_t iris::IrisInstance::resourceRead (
    InstanceId instId,
    const std::string & resourceSpec )
```

Read numeric resource and return its value.

Resource spec may be:

- <resource_name>[.<child_name>...]
- <resource_group>.<resource_name>[.<child_name>...]
- tag:<tag> (e.g. "tag:isInstructionCounter" or "tag:isPc")
- crn:<canonical_register_number_in_decimal> (usage: resourceRead(instId, "crn:" + std::to_string(iris::ElfDwarf::ARM_R0)), see [iris/IrisElfDwarfArm.h](#), consider using [resourceReadCrn\(\)](#) instead)
- rscl:<resourceId> (fallback in case resourceId is already known, consider using [irisCallThrow\(\)->resource_read\(\)](#) instead)

If the resource is not found or could not be read the appropriate error is thrown. If the resource is not a numeric resource `E_type_mismatch` is thrown.

This is a convenience function, intended to make reading well-known registers easy (e.g. PC, instruction counter). This intentionally does not handle the generic case (string registers, wide registers) to keep the usage simple. Use `resource_read()` to read any register which does not fit this function.

The resource meta-information is cached in this instance, but the value is not. The cache can be cleared with [clearCachedMetaInfo\(\)](#).

Returns

Resource value.

8.15.3.40 resourceReadCrn()

```
uint64_t iris::IrisInstance::resourceReadCrn (
    InstanceId instId,
    uint64_t canonicalRegisterNumber ) [inline]
```

Read numeric resource and return its value (using the canonical register number aka DWARF register id).

See [resourceRead\(\)](#) and the "crn:" case within.

Returns

Resource value.

8.15.3.41 resourceReadStr()

```
std::string iris::IrisInstance::resourceReadStr (
    InstanceId instId,
    const std::string & resourceSpec )
```

Read string resource, or read other resources as string.

Numeric resource values get converted to a string according to the type and bitWidth. Errors in the result.error fields are returned as string. noValue resources return an empty string.

See [resourceRead\(\)](#) for semantics of resourceSpec, errors and limitations.

8.15.3.42 resourceWrite()

```
void iris::IrisInstance::resourceWrite (
    InstanceId instId,
    const std::string & resourceSpec,
    uint64_t value )
```

Write numeric resource.

If the resource is not a numeric resource E_type_mismatch is thrown.

See [resourceRead\(\)](#) for semantics of resourceSpec, errors and limitations.

8.15.3.43 resourceWriteCrn()

```
void iris::IrisInstance::resourceWriteCrn (
    InstanceId instId,
    uint64_t canonicalRegisterNumber,
    uint64_t value ) [inline]
```

Write numeric resource by canonical register number (aka DWARF register id).

See [resourceWrite\(\)](#) for semantics.

8.15.3.44 resourceWriteStr()

```
void iris::IrisInstance::resourceWriteStr (
    InstanceId instId,
    const std::string & resourceSpec,
    const std::string & value )
```

Write string resource, or write numeric resource from string.

If the resource is not a string the value is converted to a numeric value according to the resource type.

See [resourceRead\(\)](#) for semantics of resourceSpec, errors and limitations.

8.15.3.45 sendRequest()

```
bool iris::IrisInstance::sendRequest (
    IrisRequest & req ) [inline]
```

Send an Iris request or notification and potentially wait for a response.

Parameters

<i>req</i>	Iris request to send.
------------	-----------------------

Returns

Returns true iff a non-error response was received, and therefore the result values must be decoded.

Use this to manually call functions implemented in the called target but not implemented in IrisCppAdapter.

8.15.3.46 sendResponse()

```
void iris::IrisInstance::sendResponse (
    const uint64_t * response ) [inline]
```

Send a response to the remote Iris interface.

Call this from the function implementations registered with [registerFunction\(\)](#) or [irisRegisterFunction\(\)](#).

Parameters

<i>response</i>	The Iris response message to send.
-----------------	------------------------------------

8.15.3.47 setCallback_IRIS_SHUTDOWN_LEAVE()

```
void iris::IrisInstance::setCallback_IRIS_SHUTDOWN_LEAVE (
    EventCallbackFunction f )
```

Set callback function for IRIS_SHUTDOWN_LEAVE.

8.15.3.48 setCallback_IRIS_SIMULATION_TIME_EVENT()

```
void iris::IrisInstance::setCallback_IRIS_SIMULATION_TIME_EVENT (
    EventCallbackFunction f )
```

Set callback function for IRIS_SIMULATION_TIME_EVENT.

8.15.3.49 setConnectionInterface()

```
void iris::IrisInstance::setConnectionInterface (
    IrisConnectionInterface * connection_interface )
```

Set the remote connection interface.

Used to set the IrisConnectionInterface if it was not set in the constructor.

Parameters

<i>connection_interface</i>	The interface used to connect to an Iris simulation.
-----------------------------	--

8.15.3.50 setPendingSyncStepResponse()

```
void iris::IrisInstance::setPendingSyncStepResponse (
    RequestId requestId )
```

Set pending response to a step_syncStep() call.

This function is called when the step_syncStep() function is called and the response is delivered when the simulation time stopped.

8.15.3.51 setProperty()

```
template<class T >
void iris::IrisInstance::setProperty (
    const std::string & propertyName,
    const T & propertyValue ) [inline]
```

Set/add instance property.

This creates a new property or overwrites an existing one.

Properties (name and value) are defined by the instance that has them. Properties are not to be confused with parameters, whose values are defined by clients or by parent components and some parameters might change at runtime.

Properties are exposed by the function instance_getProperties(). This should only ever be called upon initialization, before other components have a chance to call instance_getProperties(). Properties are constant and should not be changed at runtime. T can be bool, uint64_t, int64_t, or std::string.

Parameters

<i>propertyName</i>	Name of the property.
---------------------	-----------------------

Parameters

<i>propertyValue</i>	Value of the property.
----------------------	------------------------

8.15.3.52 setSyncStepEventBufferId()

```
bool iris::IrisInstance::setSyncStepEventBufferId (
    EventBufferId evBufId )
```

Set event buffer to use with `step_syncStep()` call.

Specifying `IRIS_UINT64_MAX` is valid and means that `step_syncStep()` should not return any events".

8.15.3.53 setThrowOnError()

```
void iris::IrisInstance::setThrowOnError (
    bool throw_on_error ) [inline]
```

Set default error behavior for [irisCall\(\)](#).

Parameters

<i>throw_on_error</i>	If true, calls made using irisCall() that respond with an error response will throw an exception. This is the same behavior as irisCallThrow() . If false, calls made using irisCall() that respond with an error response will return the error code and not throw an exception. This is the same behavior as irisCallNoThrow() .
-----------------------	--

8.15.3.54 simulationTimeDisableEvents()

```
void iris::IrisInstance::simulationTimeDisableEvents ( )
```

Disable the internal reception of `IRIS_SIMULATION_TIME_EVENT` events for performance reasons (e.g. during synchronous stepping).

The callback set with [setCallback_IRIS_SIMULATION_TIME_EVENT\(\)](#) will no longer be called.

Internal `IRIS_SIMULATION_TIME_EVENTS` will automatically be re-enabled as soon as one of the other `simulationTime*()` functions is called.

This function throws Iris errors.

8.15.3.55 simulationTimeIsRunning()

```
bool iris::IrisInstance::simulationTimeIsRunning ( )
```

Return true iff simulation is currently running.

Note that this information is always out of date if there is another simulation controller.

This function throws Iris errors.

8.15.3.56 simulationTimeRun()

```
void iris::IrisInstance::simulationTimeRun ( )
```

Run simulation time and wait until simulation time started running.

Does not wait until model stopped again. See [simulationTimeRunUntilStop\(\)](#).

This function throws Iris errors.

8.15.3.57 simulationTimeRunUntilStop()

```
void iris::IrisInstance::simulationTimeRunUntilStop (
    double timeoutInSeconds = 0.0 )
```

Run simulation time and wait until simulation time stopped again or until timeout expired.

This function throws Iris errors.

8.15.3.58 simulationTimeStop()

```
void iris::IrisInstance::simulationTimeStop ( )
```

Stop simulation time and wait until simulation time stopped.
This function throws Iris errors.

8.15.3.59 simulationTimeWaitForStop()

```
bool iris::IrisInstance::simulationTimeWaitForStop (
    double timeoutInSeconds = 0.0 )
```

Wait for simulation time to stop or timeout.

This function only works after [simulationTimeRun\(\)](#) has been called. When the simulation time already stopped after [simulationTimeRun\(\)](#) then this function exits immediately.

This function throws Iris errors.

Parameters

<i>timeoutInSeconds</i>	Stop waiting after the specified timeout and return false on timeout. 0.0 means to wait indefinitely.
-------------------------	---

Returns

true if simulation time stopped, false on timeout. When timeoutInSeconds is 0.0 (= no timeout) this always returns true.

8.15.3.60 unublishCppInterface()

```
void iris::IrisInstance::unublishCppInterface (
    const std::string & interfaceName ) [inline]
```

Unpublish a previously published C++ interface.

After calling this function the corresponding instance_getCppInterface...() function is no longer available. This is silently ignored if the interface was not previously published.

Parameters

<i>interfaceName</i>	Class name or interface name of the interface to be unpublished.
----------------------	--

8.15.3.61 unregisterInstance()

```
IrisErrorCode iris::IrisInstance::unregisterInstance ( )
```

Unregister this instance.

Iris calls must not be made after the instance has been unregistered.

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

- [IrisInstance.h](#)

8.16 iris::IrisInstanceBreakpoint Class Reference

Breakpoint add-on for [IrisInstance](#).

```
#include <IrisInstanceBreakpoint.h>
```

Public Member Functions

- void [addCondition](#) (const std::string &name, const std::string &type, const std::string &description, const std::vector< std::string > bpt_types=std::vector< std::string >())

- Add an optional component-specific condition that can be configured by clients.

 - void `attachTo` (`IrisInstance` *irisInstance)

Attach this `IrisInstance` add-on to a specific `IrisInstance`.
- const `BreakpointInfo` * `getBreakpointInfo` (`BreakpointId` bptId) const

Get `BreakpointInfo` for a breakpoint id.
- void `handleBreakpointHit` (const `BreakpointHitInfo` &bptHitInfo)

Handle breakpoint hit.
- `IrisInstanceBreakpoint` (`IrisInstance` *irisInstance=nullptr)
- void `notifyBreakpointHit` (`BreakpointId` bptId, `uint64_t` time, `uint64_t` pc, `MemorySpaceId` pcSpaceId)

Notify clients that a code breakpoint was hit.
- void `notifyBreakpointHitData` (`BreakpointId` bptId, `uint64_t` time, `uint64_t` pc, `MemorySpaceId` pcSpaceId, `uint64_t` accessAddr, `uint64_t` accessSize, const `std::string` &accessRw, const `std::vector`< `uint64_t` > &data)

Notify clients that a data breakpoint was hit.
- void `notifyBreakpointHitRegister` (`BreakpointId` bptId, `uint64_t` time, `uint64_t` pc, `MemorySpaceId` pcSpaceId, const `std::string` &accessRw, const `std::vector`< `uint64_t` > &data)

Notify clients that a register breakpoint was hit.
- void `setBreakpointDeleteDelegate` (`BreakpointDeleteDelegate` delegate)

Set breakpoint delete delegate for all breakpoints deleted by this instance.
- void `setBreakpointSetDelegate` (`BreakpointSetDelegate` delegate)

Set breakpoint set delegate for all breakpoints set by this instance.
- void `setEventHandler` (`IrisInstanceEvent` *handler)

Set the event handler used to notify the clients that enable the `IRIS_BREAKPOINT_HIT` event.
- void `setHandleBreakpointHitDelegate` (`HandleBreakpointHitDelegate` delegate)

Set a delegate for handling breakpoint hit in this instance.

8.16.1 Detailed Description

Breakpoint add-on for `IrisInstance`.

Instances use this class to support breakpoint functionality.

It implements all `Iris breakpoint*()` functions and maintains the breakpoint information that is set by `breakpoint_set()` and is exposed by `breakpoint_getList()`.

Example usage:

```
irisInstanceBpt = new iris::IrisInstanceBreakpoint(irisInstance);
irisInstanceBpt->setBreakpointSetDelegate(bptSetDel);           // Use this delegate for breakpoint set.
irisInstanceBpt->setBreakpointDeleteDelegate(bptDeleteDel);     // Use this delegate for breakpoint delete.
// When a breakpoint is hit, notify the instances that enable the IRIS_BREAKPOINT_HIT event.
irisInstanceBpt->setEventHandler(irisInstanceEvent);
```

See `DummyComponent.h` for a working example.

8.16.2 Member Function Documentation

8.16.2.1 addCondition()

```
void iris::IrisInstanceBreakpoint::addCondition (
    const std::string & name,
    const std::string & type,
    const std::string & description,
    const std::vector< std::string > bpt_types = std::vector< std::string >() )
```

Add an optional component-specific condition that can be configured by clients.

Parameters

<i>name</i>	The name of the condition.
<i>type</i>	The type of the value that clients set to configure the condition.

Parameters

<i>description</i>	A description of the condition.
<i>bpt_types</i>	A list of breakpoint types that this condition can be applied to. An empty list indicates all types.

8.16.2.2 attachTo()

```
void iris::IrisInstanceBreakpoint::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
Only use this method if nullptr was passed to the constructor.

Parameters

<i>irisInstance</i>	The IrisInstance to attach to.
---------------------	--

8.16.2.3 getBreakpointInfo()

```
const BreakpointInfo * iris::IrisInstanceBreakpoint::getBreakpointInfo (
    BreakpointId bptId ) const
```

Get BreakpointInfo for a breakpoint id.

Parameters

<i>bptId</i>	The breakpoint id for which the BreakpointInfo is requested.
--------------	--

Returns

A pointer to the BreakpointInfo for the requested breakpoint or nullptr if *bptId* is not a valid breakpoint id.

8.16.2.4 handleBreakpointHit()

```
void iris::IrisInstanceBreakpoint::handleBreakpointHit (
    const BreakpointHitInfo & bptHitInfo )
```

Handle breakpoint hit.

Parameters

<i>bptHitInfo</i>	The information of the breakpoint that is hit. Calls a delegate method in the model.
-------------------	--

8.16.2.5 notifyBreakpointHit()

```
void iris::IrisInstanceBreakpoint::notifyBreakpointHit (
    BreakpointId bptId,
    uint64_t time,
    uint64_t pc,
    MemorySpaceId pcSpaceId )
```

Notify clients that a code breakpoint was hit.
It notifies clients by emitting an `IRIS_BREAKPOINT_HIT` event.

Parameters

<i>bptId</i>	Breakpoint id for the breakpoint that was hit.
<i>time</i>	Simulation time at which the breakpoint hit.
<i>pc</i>	Value of the relevant program counter when the event hit.
<i>pc</i> ↔ <i>SpaceId</i>	Memory space Id for the memory space that the PC address corresponds to.

8.16.2.6 notifyBreakpointHitData()

```
void iris::IrisInstanceBreakpoint::notifyBreakpointHitData (
    BreakpointId bptId,
    uint64_t time,
    uint64_t pc,
    MemorySpaceId pcSpaceId,
    uint64_t accessAddr,
    uint64_t accessSize,
    const std::string & accessRw,
    const std::vector< uint64_t > & data )
```

Notify clients that a data breakpoint was hit.

It notifies clients by emitting an IRIS_BREAKPOINT_HIT event.

Parameters

<i>bptId</i>	Breakpoint id for the breakpoint that was hit.
<i>time</i>	Simulation time at which the breakpoint hit.
<i>pc</i>	Value of the relevant program counter when the event hit.
<i>pcSpaceId</i>	Memory space Id for the memory space that the PC address corresponds to.
<i>accessAddr</i>	The address of the data access that triggered the breakpoint.
<i>accessSize</i>	The size of the data access that triggered the breakpoint.
<i>accessRw</i>	Indicates the direction of the access. "r" = read access or "w" = write access.
<i>data</i>	The data that was written or read during the access that triggered the breakpoint.

8.16.2.7 notifyBreakpointHitRegister()

```
void iris::IrisInstanceBreakpoint::notifyBreakpointHitRegister (
    BreakpointId bptId,
    uint64_t time,
    uint64_t pc,
    MemorySpaceId pcSpaceId,
    const std::string & accessRw,
    const std::vector< uint64_t > & data )
```

Notify clients that a register breakpoint was hit.

It notifies clients by emitting an IRIS_BREAKPOINT_HIT event.

Parameters

<i>bptId</i>	Breakpoint id for the breakpoint that was hit.
<i>time</i>	Simulation time at which the breakpoint hit.
<i>pc</i>	Value of the relevant program counter when the event hit.
<i>pc</i> ↔ <i>SpaceId</i>	Memory space Id for the memory space that the PC address corresponds to.

Parameters

<i>accessRw</i>	Indicates the direction of the access. "r" = read access or "w" = write access.
<i>data</i>	The data that was written or read during the access that triggered the breakpoint.

8.16.2.8 setBreakpointDeleteDelegate()

```
void iris::IrisInstanceBreakpoint::setBreakpointDeleteDelegate (
    BreakpointDeleteDelegate delegate )
```

Set breakpoint delete delegate for all breakpoints deleted by this instance.

Parameters

<i>delegate</i>	A BreakpointDeleteDelegate to call when a breakpoint is deleted.
-----------------	--

8.16.2.9 setBreakpointSetDelegate()

```
void iris::IrisInstanceBreakpoint::setBreakpointSetDelegate (
    BreakpointSetDelegate delegate )
```

Set breakpoint set delegate for all breakpoints set by this instance.

Parameters

<i>delegate</i>	A BreakpointSetDelegate to call when a breakpoint is set.
-----------------	---

8.16.2.10 setEventHandler()

```
void iris::IrisInstanceBreakpoint::setEventHandler (
    IrisInstanceEvent * handler )
```

Set the event handler used to notify the clients that enable the IRIS_BREAKPOINT_HIT event.

All breakpoint events are normal events and are handled through the same mechanism as other events.

8.16.2.11 setHandleBreakpointHitDelegate()

```
void iris::IrisInstanceBreakpoint::setHandleBreakpointHitDelegate (
    HandleBreakpointHitDelegate delegate )
```

Set a delegate for handling breakpoint hit in this instance.

Parameters

<i>delegate</i>	A HandleBreakpointHitDelegate to call when a breakpoint is hit.
-----------------	---

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

- [IrisInstanceBreakpoint.h](#)

8.17 iris::IrisInstanceBuilder Class Reference

Builder interface to populate an [IrisInstance](#) with registers, memory etc.

```
#include <IrisInstanceBuilder.h>
```

Classes

- class [AddressTranslationBuilder](#)
Used to set metadata for an address translation.
- class [EventSourceBuilder](#)
Used to set metadata on an EventSource.
- class [FieldBuilder](#)
Used to set metadata on a register field resource.
- class [MemorySpaceBuilder](#)
Used to set metadata for a memory space.
- class [ParameterBuilder](#)
Used to set metadata on a parameter.
- class [RegisterBuilder](#)
Used to set metadata on a register resource.
- class [SemihostingManager](#)
semihosting_apis [IrisInstanceBuilder](#) semihosting APIs
- class [TableBuilder](#)
Used to set metadata for a table.
- class [TableColumnBuilder](#)
Used to set metadata for a table column.

Public Member Functions

- [AddressTranslationBuilder](#) **addAddressTranslation** (MemorySpaceId inSpaceId, MemorySpaceId outSpaceId, const std::string &description)
Add an address translation.
- void **addBreakpointCondition** (const std::string &name, const std::string &type, const std::string &description, const std::vector< std::string > bpt_types=std::vector< std::string >())
Add an optional component-specific condition.
- [EventSourceBuilder](#) **addEventSource** (const std::string &name, bool isHidden=false)
Add metadata for an event source.
- [EventSourceBuilder](#) **addEventSource** (const std::string &name, [IrisEventEmitterBase](#) &event_emitter, bool isHidden=false)
Add metadata for an event source that uses an [IrisEventEmitter](#).
- [MemorySpaceBuilder](#) **addMemorySpace** (const std::string &name)
Add metadata for one memory space.
- [RegisterBuilder](#) **addNoValueRegister** (const std::string &name, const std::string &description, const std::string &format)
Add metadata for one noValue resource.
- [ParameterBuilder](#) **addParameter** (const std::string &name, uint64_t bitWidth, const std::string &description)
Add numeric parameter.
- [RegisterBuilder](#) **addRegister** (const std::string &name, uint64_t bitWidth, const std::string &description, uint64_t addressOffset=IRIS_UINT64_MAX, uint64_t canonicalRn=IRIS_UINT64_MAX)
Add metadata for one numeric register resource.
- [ParameterBuilder](#) **addStringParameter** (const std::string &name, const std::string &description)
Add string parameter.
- [RegisterBuilder](#) **addStringRegister** (const std::string &name, const std::string &description)
Add metadata for one string register resource.
- [TableBuilder](#) **addTable** (const std::string &name)
Add metadata for one table.
- void **beginResourceGroup** (const std::string &name, const std::string &description, uint64_t subRscId←Start=IRIS_UINT64_MAX, const std::string &cname=std::string())

- Begin a new resource group.*

 - void [deleteEventSource](#) (const std::string &name)

Delete event source.
- [EventSourceBuilder enhanceEventSource](#) (const std::string &name)

Enhance existing event source.
- [ParameterBuilder enhanceParameter](#) (ResourceId rsclId)

Get [ParameterBuilder](#) to enhance a parameter.
- [RegisterBuilder enhanceRegister](#) (ResourceId rsclId)

Get [RegisterBuilder](#) to enhance register.
- void [finalizeRegisterReadEvent](#) ()
- void [finalizeRegisterUpdateEvent](#) ()
- Finalize set up of an [IrisEventEmitter](#).*
- const BreakpointInfo * [getBreakpointInfo](#) (BreakpointId bptId)

Get the breakpoint information for a given breakpoint.
- [IrisInstanceEvent](#) * [getIrisInstanceEvent](#) ()
- const ResourceInfo & [getResourceInfo](#) (ResourceId rsclId)

Get ResourceInfo of a previously added register.
- bool [hasEventSource](#) (const std::string &name)
- Check whether event source already exists.*
- [IrisInstanceBuilder](#) ([IrisInstance](#) *iris_instance)

Construct an [IrisInstanceBuilder](#) for an [Iris](#) instance.
- void [notifyBreakpointHit](#) (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId)

Notify clients that a code breakpoint was hit.
- void [notifyBreakpointHitData](#) (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId, uint64_t accessAddr, uint64_t accessSize, const std::string &accessRw, const std::vector< uint64_t > &data)

Notify clients that a data breakpoint was hit ([IRIS_BREAKPOINT_HIT](#)).
- void [notifyBreakpointHitRegister](#) (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId, const std::string &accessRw, const std::vector< uint64_t > &data)

Notify clients that a register breakpoint was hit ([IRIS_BREAKPOINT_HIT](#)).
- uint64_t [openImage](#) (const std::string &filename)

Open an image to be read using [image_loadDataPull\(\)](#) or [image_loadDataRead\(\)](#).
- void [renameEventSource](#) (const std::string &name, const std::string &newName)
- Rename existing event source.*
- void [resetRegisterReadEvent](#) ()
- Reset the active register read event.*
- void [resetRegisterUpdateEvent](#) ()
- Reset the active register update event.*
- template<IrisErrorCode(*)>(const BreakpointInfo &) FUNC>
void [setBreakpointDeleteDelegate](#) ()

Set the delegate that is called when a breakpoint is deleted.
- void [setBreakpointDeleteDelegate](#) ([BreakpointDeleteDelegate](#) delegate)

Set the delegate that is called when a breakpoint is deleted.
- template<typename T , IrisErrorCode(T::*)(const BreakpointInfo &) METHOD>
void [setBreakpointDeleteDelegate](#) (T *instance)

Set the delegate that is called when a breakpoint is deleted.
- template<IrisErrorCode(*)>(BreakpointInfo &) FUNC>
void [setBreakpointSetDelegate](#) ()

Set the delegate that is called when a breakpoint is set.
- void [setBreakpointSetDelegate](#) ([BreakpointSetDelegate](#) delegate)

Set the delegate that is called when a breakpoint is set.

- `template<typename T , IrisErrorCode(T::*)(BreakpointInfo &) METHOD>`
`void setBreakpointSetDelegate (T *instance)`
Set the delegate that is called when a breakpoint is set.
- `template<IrisErrorCode(*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) FUNC>`
`void setDefaultAddressTranslateDelegate ()`
Set the default address translation function for all subsequently added memory spaces.
- `void setDefaultAddressTranslateDelegate (MemoryAddressTranslateDelegate delegate=MemoryAddressTranslateDelegate())`
Set the default address translation function for all subsequently added memory spaces.
- `template<typename T , IrisErrorCode(T::*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) METHOD>`
`void setDefaultAddressTranslateDelegate (T *instance)`
Set the default address translation function for all subsequently added memory spaces.
- `template<IrisErrorCode(*)(EventStream *&, const EventSourceInfo &, const std::vector< std::string > &) FUNC>`
`void setDefaultEsCreateDelegate ()`
Set the delegate that helps to create a new event stream for the simulation-specific event.
- `void setDefaultEsCreateDelegate (EventStreamCreateDelegate delegate)`
Set the delegate that helps to create a new event stream for the simulation-specific event.
- `template<typename T , IrisErrorCode(T::*)(EventStream *&, const EventSourceInfo &, const std::vector< std::string > &) METHOD>`
`void setDefaultEsCreateDelegate (T *instance)`
Set the delegate that helps to create a new event stream for the simulation-specific event.
- `template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) FUNC>`
`void setDefaultGetMemorySidebandInfoDelegate ()`
Set the default sideband info function for all subsequently added memory spaces.
- `void setDefaultGetMemorySidebandInfoDelegate (MemoryGetSidebandInfoDelegate delegate)`
Set the default sideband info function for all subsequently added memory spaces.
- `template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) METHOD>`
`void setDefaultGetMemorySidebandInfoDelegate (T *instance)`
Set the default sideband info function for all subsequently added memory spaces.
- `template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) FUNC>`
`void setDefaultMemoryReadDelegate ()`
Set the default read function for all subsequently added memory spaces.
- `void setDefaultMemoryReadDelegate (MemoryReadDelegate delegate=MemoryReadDelegate())`
Set the default read function for all subsequently added memory spaces.
- `template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) METHOD>`
`void setDefaultMemoryReadDelegate (T *instance)`
Set the default read function for all subsequently added memory spaces.
- `template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) FUNC>`
`void setDefaultMemoryWriteDelegate ()`
Set default write function for all subsequently added memory spaces.
- `void setDefaultMemoryWriteDelegate (MemoryWriteDelegate delegate=MemoryWriteDelegate())`
Set the default write function for all subsequently added memory spaces.
- `template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>`
`void setDefaultMemoryWriteDelegate (T *instance)`
Set the default write function for all subsequently added memory spaces.
- `template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) READER, IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &) WRITER>`
`void setDefaultResourceDelegates (T *instance)`
Set both read and write resource delegates if they are defined in the same class.

- `template<IrisErrorCode(*)>(const ResourceInfo &, ResourceReadResult &) FUNC>`
`void setDefaultResourceReadDelegate ()`
Set default read access function for all subsequently added resources.
- `void setDefaultResourceReadDelegate (ResourceReadDelegate delegate=ResourceReadDelegate())`
Set default read access function for all subsequently added resources.
- `template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>`
`void setDefaultResourceReadDelegate (T *instance)`
Set default read access function for all subsequently added resources.
- `template<IrisErrorCode(*)>(const ResourceInfo &, const ResourceWriteValue &) FUNC>`
`void setDefaultResourceWriteDelegate ()`
Set default write access function for all subsequently added resources.
- `void setDefaultResourceWriteDelegate (ResourceWriteDelegate delegate=ResourceWriteDelegate())`
Set default write access function for all subsequently added resources.
- `template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &) METHOD>`
`void setDefaultResourceWriteDelegate (T *instance)`
Set default write access function for all subsequently added resources.
- `template<IrisErrorCode(*)>(const TableInfo &, uint64_t, uint64_t, TableReadResult &) FUNC>`
`void setDefaultTableReadDelegate ()`
Set the default table read function for all subsequently added tables.
- `template<typename T , IrisErrorCode(T::*)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) METHOD>`
`void setDefaultTableReadDelegate (T *instance)`
Set the default table read function for all subsequently added tables.
- `void setDefaultTableReadDelegate (TableReadDelegate delegate=TableReadDelegate())`
Set the default table read function for all subsequently added tables.
- `template<IrisErrorCode(*)>(const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>`
`void setDefaultTableWriteDelegate ()`
Set the default table write function for all subsequently added tables.
- `template<typename T , IrisErrorCode(T::*)(const TableInfo &, const TableRecords &, TableWriteResult &) METHOD>`
`void setDefaultTableWriteDelegate (T *instance)`
Set the default table write function for all subsequently added tables.
- `void setDefaultTableWriteDelegate (TableWriteDelegate delegate=TableWriteDelegate())`
Set the default table write function for all subsequently added tables.
- `template<IrisErrorCode(*)>(bool &) FUNC>`
`void setExecutionStateGetDelegate ()`
Set the delegate to get the execution state for this instance.
- `void setExecutionStateGetDelegate (PerInstanceExecutionStateGetDelegate delegate)`
Set the delegate to get the execution state for this instance.
- `template<typename T , IrisErrorCode(T::*)(bool &) METHOD>`
`void setExecutionStateGetDelegate (T *instance)`
Set the delegate to get the execution state for this instance.
- `template<IrisErrorCode(*)>(bool) FUNC>`
`void setExecutionStateSetDelegate ()`
Set the delegate to set the execution state for this instance.
- `void setExecutionStateSetDelegate (PerInstanceExecutionStateSetDelegate delegate=PerInstanceExecutionStateSetDelegate)`
Set the delegate to set the execution state for this instance.
- `template<typename T , IrisErrorCode(T::*)(bool) METHOD>`
`void setExecutionStateSetDelegate (T *instance)`
Set the delegate to set the execution state for this instance.
- `template<IrisErrorCode(*)>(const BreakpointHitInfo &) FUNC>`
`void setHandleBreakpointHitDelegate ()`
Set the delegate that is called when a breakpoint is hit.
- `void setHandleBreakpointHitDelegate (HandleBreakpointHitDelegate delegate)`

- Set the delegate that is called when a breakpoint is hit.*

 - `template<typename T , IrisErrorCode(T::*)(const BreakpointHitInfo &) METHOD>`
`void setHandleBreakpointHitDelegate (T *instance)`
- Set the delegate that is called when a breakpoint is hit.*

 - `template<IrisErrorCode(*) (const std::vector< uint8_t > &) FUNC>`
`void setLoadImageDataDelegate ()`
- Set the delegate to load an image from the data provided.*

 - `void setLoadImageDataDelegate (ImageLoadDataDelegate delegate=ImageLoadDataDelegate())`
- Set the delegate to load an image from the data provided.*

 - `template<typename T , IrisErrorCode(T::*)(const std::vector< uint8_t > &) METHOD>`
`void setLoadImageDataDelegate (T *instance)`
- Set the delegate to load an image from the data provided.*

 - `template<IrisErrorCode(*) (const std::string &) FUNC>`
`void setLoadImageFileDelegate ()`
- Set the delegate to load an image from a file.*

 - `void setLoadImageFileDelegate (ImageLoadFileDelegate delegate=ImageLoadFileDelegate())`
- Set the delegate to load an image from a file.*

 - `template<typename T , IrisErrorCode(T::*)(const std::string &) METHOD>`
`void setLoadImageFileDelegate (T *instance)`
- Set the delegate to load an image from a file.*

 - `void setNextSubRsclId (uint64_t nextSubRsclId)`
- Set the rsclId that will be used for the next resource to be added.*

 - `void setPropertyCanonicalMsnScheme (const std::string &canonicalMsnScheme)`
- Set the memory.canonicalMsnScheme instance property.*

 - `void setPropertyCanonicalRnScheme (const std::string &canonicalRnScheme)`
- Set the register.canonicalRnScheme instance property.*

 - `EventSourceBuilder setRegisterReadEvent (const std::string &name, const std::string &description=std::string())`
- Add a new register read event source.*

 - `EventSourceBuilder setRegisterReadEvent (const std::string &name, IrisRegisterEventEmitterBase &event_emitter)`
- Add a new register read event source.*

 - `EventSourceBuilder setRegisterUpdateEvent (const std::string &name, const std::string &description=std::string())`
- Add a new register update event source.*

 - `EventSourceBuilder setRegisterUpdateEvent (const std::string &name, IrisRegisterEventEmitterBase &event_emitter)`
- Add a new register update event source.*

 - `template<IrisErrorCode(*) (uint64_t &, const std::string &) FUNC>`
`void setRemainingStepGetDelegate ()`
- Set the delegate to get the remaining steps for this instance.*

 - `void setRemainingStepGetDelegate (RemainingStepGetDelegate delegate)`
- Set the delegate to get the remaining steps for this instance.*

 - `template<typename T , IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>`
`void setRemainingStepGetDelegate (T *instance)`
- Set the delegate to get the remaining steps for this instance.*

 - `template<IrisErrorCode(*) (uint64_t, const std::string &) FUNC>`
`void setRemainingStepSetDelegate ()`
- Set the delegate to set the remaining steps for this instance.*

 - `void setRemainingStepSetDelegate (RemainingStepSetDelegate delegate=RemainingStepSetDelegate())`
- Set the delegate to set the remaining steps for this instance.*

 - `template<typename T , IrisErrorCode(T::*)(uint64_t, const std::string &) METHOD>`
`void setRemainingStepSetDelegate (T *instance)`

- Set the delegate to set the remaining steps for this instance.*

 - `template<IrisErrorCode(*)>(uint64_t &, const std::string &) FUNC>`
`void setStepCountGetDelegate ()`
- Set the delegate to get the step count for this instance.*

 - `void setStepCountGetDelegate (StepCountGetDelegate delegate=StepCountGetDelegate())`
- Set the delegate to get the step count for this instance.*

 - `template<typename T , IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>`
`void setStepCountGetDelegate (T *instance)`
- Set the delegate to get the step count for this instance.*

 - `void setTag (ResourceId rsId, const std::string &tag)`

Set a tag for a specific resource.
- `void setGetCurrentDisassemblyModeDelegate (GetCurrentDisassemblyModeDelegate delegate)`

disass_apis [IrisInstanceBuilder](#) disassembler APIs

 - `template<typename T , IrisErrorCode(T::*)(std::string &) METHOD>`
`void setGetCurrentDisassemblyModeDelegate (T *instance)`
 - `void setGetDisassemblyDelegate (GetDisassemblyDelegate delegate)`

Set the delegate to get the disassembly of a chunk of memory.

 - `template<typename T , IrisErrorCode(T::*)(uint64_t, const std::string &, MemoryReadResult &, uint64_t, uint64_t, std::vector< DisassemblyLine > &) METHOD>`
`void setGetDisassemblyDelegate (T *instance)`
 - `template<IrisErrorCode(*)>(uint64_t, const std::string &, MemoryReadResult &, uint64_t, uint64_t, std::vector< DisassemblyLine > &) FUNC>`
`void setGetDisassemblyDelegate ()`
 - `void setDisassembleOpcodeDelegate (DisassembleOpcodeDelegate delegate)`

Set the delegate to get the disassembly of Opcode.

 - `template<typename T , IrisErrorCode(T::*)(const std::vector< uint64_t > &, uint64_t, const std::string &, DisassembleContext &, DisassemblyLine &) METHOD>`
`void setDisassembleOpcodeDelegate (T *instance)`
 - `template<IrisErrorCode(*)>(const std::vector< uint64_t > &, uint64_t, const std::string &, DisassembleContext &, DisassemblyLine &) FUNC>`
`void setDisassembleOpcodeDelegate ()`
 - `void addDisassemblyMode (const std::string &name, const std::string &description)`

Add a disassembly mode.
- `void setDbgStateSetRequestDelegate (DebuggableStateSetRequestDelegate delegate=DebuggableStateSetRequestDelegate)`

debuggable_state_apis [IrisInstanceBuilder](#) debuggable state APIs

 - `template<typename T , IrisErrorCode(T::*)(bool) METHOD>`
`void setDbgStateSetRequestDelegate (T *instance)`

Set the delegate to set the debuggable state request flag for this instance.

 - `template<IrisErrorCode(*)>(bool) FUNC>`
`void setDbgStateSetRequestDelegate ()`

Set the delegate to set the debuggable state request flag for this instance.

 - `void setDbgStateGetAcknowledgeDelegate (DebuggableStateGetAcknowledgeDelegate delegate=DebuggableStateGetAcknowledgeDelegate)`

Set the delegate to get the debuggable state acknowledge flag for this instance.

 - `template<typename T , IrisErrorCode(T::*)(bool &) METHOD>`
`void setDbgStateGetAcknowledgeDelegate (T *instance)`

Set the delegate to get the debuggable state acknowledge flag for this instance.

 - `template<IrisErrorCode(*)>(bool &) FUNC>`
`void setDbgStateGetAcknowledgeDelegate ()`

Set the delegate to get the debuggable state acknowledge flag for this instance.

- `template<typename T, IrisErrorCode(T::*)(bool) SET_REQUEST, IrisErrorCode(T::*)(bool &) GET_ACKNOWLEDGE>`
`void setDbgStateDelegates (T *instance)`
Set both the debuggable state delegates.
- `void setCheckpointSaveDelegate (CheckpointSaveDelegate delegate=CheckpointSaveDelegate())`
Delegates for checkpointing.
- `template<typename T, IrisErrorCode(T::*)(const std::string &) METHOD>`
`void setCheckpointSaveDelegate (T *instance)`
- `void setCheckpointRestoreDelegate (CheckpointRestoreDelegate delegate=CheckpointRestoreDelegate())`
- `template<typename T, IrisErrorCode(T::*)(const std::string &) METHOD>`
`void setCheckpointRestoreDelegate (T *instance)`
- `SemihostingManager enableSemihostingAndGetManager ()`
Enable semihosting functionality for this instance and get a manager object to make use of it.

8.17.1 Detailed Description

Builder interface to populate an `IrisInstance` with registers, memory etc.
 See DummyComponent.h for a working example.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 IrisInstanceBuilder()

```
iris::IrisInstanceBuilder::IrisInstanceBuilder (
    IrisInstance * iris_instance )
```

Construct an `IrisInstanceBuilder` for an Iris instance.

Parameters

<code>iris_instance</code>	The instance to build.
----------------------------	------------------------

8.17.3 Member Function Documentation

8.17.3.1 addTable()

```
TableBuilder iris::IrisInstanceBuilder::addTable (
    const std::string & name ) [inline]
```

Add metadata for one table.

Typical use pattern:

```
addTableInfo("name")
    .setDescription("description")
    .setMinIndex(...)
    .setMaxIndex(...)
    .setIndexFormatHint(...)
    .setFormatShort(...)
    .setFormatLong(...)
    .setReadDelegate(...)
    .setWriteDelegate(...)
    .addColumnInfo(...)
    .addColumnInfo(...)
    ...
```

Parameters

<code>name</code>	Name of the new table.
-------------------	------------------------

Returns

A [TableBuilder](#) object than can be used to set metadata for the new table.

8.17.3.2 enableSemihostingAndGetManager()

[SemihostingManager](#) `iris::IrisInstanceBuilder::enableSemihostingAndGetManager () [inline]`
 Enable semihosting functionality for this instance and get a manager object to make use of it.

Returns

A [SemihostingManager](#) object to manage semihosting functionality for this instance.

8.17.3.3 setDbgStateDelegates()

```
template<typename T , IrisErrorCode(T::*)(bool) SET_REQUEST, IrisErrorCode(T::*)(bool &) GET←
_ACKNOWLEDGE>
void iris::IrisInstanceBuilder::setDbgStateDelegates (
    T * instance ) [inline]
```

Set both the debuggable state delegates.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setRequestFlag(bool request_debuggable_state);
    iris::IrisErrorCode getAcknowledgeFlag(bool &debuggable_state_acknowledge);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateDelegates<MyClass,
    &MyClass::setRequest,
    &MyClass::getAcknowledgeFlag>(&myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines both a debuggable state request set and a get acknowledge delegate method.
<i>SET_REQUEST</i>	A method of class T which is a debuggable state request set delegate.
<i>GET_ACKNOWLEDGE</i>	A method of class T which is a debuggable state get acknowledge delegate.

Parameters

<i>instance</i>	An instance of class T on which SET_REQUEST and GET_ACKNOWLEDGE should be called.
-----------------	---

8.17.3.4 setDbgStateGetAcknowledgeDelegate() [1/3]

```
template<IrisErrorCode(*) (bool &) FUNC>
void iris::IrisInstanceBuilder::setDbgStateGetAcknowledgeDelegate ( ) [inline]
```

Set the delegate to get the debuggable state acknowledge flag for this instance.

Usage:

```
iris::IrisErrorCode getAcknowledgeFlag(bool &debuggable_state_acknowledge);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateGetAcknowledgeDelegate<&getAcknowledgeFlag>();
```

Template Parameters

<i>FUNC</i>	Global function to call to get the debuggable state acknowledge flag.
-------------	---

8.17.3.5 setDbgStateGetAcknowledgeDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setDbgStateGetAcknowledgeDelegate (
    DebuggableStateGetAcknowledgeDelegate delegate = DebuggableStateGetAcknowledgeDelegate()
) [inline]
```

Set the delegate to get the debuggable state acknowledge flag for this instance.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↔` `not_implemented` for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getAcknowledgeFlag(bool &debuggable_state_acknowledge);
};
MyClass myInstanceOfMyClass;
iris::DebuggableStateGetAcknowledgeDelegate delegate =
    iris::DebuggableStateGetAcknowledgeDelegate::make<MyClass,
        &MyClass::getAcknowledgeFlag>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateGetAcknowledgeDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object to call to get the debuggable state acknowledge flag.
-----------------	---

8.17.3.6 setDbgStateGetAcknowledgeDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(bool &) METHOD>
void iris::IrisInstanceBuilder::setDbgStateGetAcknowledgeDelegate (
    T * instance ) [inline]
```

Set the delegate to get the debuggable state acknowledge flag for this instance.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getAcknowledgeFlag(bool &debuggable_state_acknowledge);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateGetAcknowledgeDelegate<MyClass, &MyClass::getAcknowledgeFlag>(&myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines a debuggable state get acknowledge delegate method.
<i>METHOD</i>	A method of class T which is a debuggable state get acknowledge delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

8.17.3.7 setDbgStateSetRequestDelegate() [1/3]

```
template<IrisErrorCode(*) (bool) FUNC>
void iris::IrisInstanceBuilder::setDbgStateSetRequestDelegate ( ) [inline]
```

Set the delegate to set the debuggable state request flag for this instance.

Usage:

```
iris::IrisErrorCode setRequestFlag(bool request_debuggable_state);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
```

```
builder->setDbgStateSetRequestDelegate<setRequestFlag>();
```

Template Parameters

<i>FUNC</i>	Global function to call to set the debuggable state request flag.
-------------	---

8.17.3.8 setDbgStateSetRequestDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setDbgStateSetRequestDelegate (
    DebuggableStateSetRequestDelegate delegate = DebuggableStateSetRequestDelegate()
) [inline]
```

debuggable_state_apis [IrisInstanceBuilder](#) debuggable state APIs

Set the delegate to set the debuggable state request flag for this instance.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↔` not_implemented for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setRequestFlag(bool request_debuggable_state);
};
MyClass myInstanceOfMyClass;
iris::DebuggableStateSetRequestDelegate delegate =
    iris::DebuggableStateSetRequestDelegate::make<MyClass, &MyClass::setRequestFlag>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateSetRequestDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object to call to set the debuggable state request flag.
-----------------	---

8.17.3.9 setDbgStateSetRequestDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(bool) METHOD>
void iris::IrisInstanceBuilder::setDbgStateSetRequestDelegate (
    T * instance ) [inline]
```

Set the delegate to set the debuggable state request flag for this instance.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setRequestFlag(bool request_debuggable_state);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateSetRequestDelegate<MyClass, &MyClass::setRequestFlag>(&myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines a debuggable state request set delegate method.
<i>METHOD</i>	A method of class T which is a debuggable state request set delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

8.17.3.10 setDefaultTableReadDelegate() [1/3]

```
template<IrisErrorCode(*) (const TableInfo &, uint64_t, uint64_t, TableReadResult &) FUNC>
void iris::IrisInstanceBuilder::setDefaultTableReadDelegate ( ) [inline]
```

Set the default table read function for all subsequently added tables.

Tables that do not explicitly override the access function using

```
addTable(...).setReadDelegate(...)
```

will use this delegate.

Usage:

```
iris::IrisErrorCode readTable(const iris::TableInfo &tableInfo, uint64_t index,
                             uint64_t count, iris::TableReadResult &result);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableReadDelegate(&readTable);
builder->addTable(...); // Uses readTable
```

Template Parameters

<i>FUNC</i>	Global function to call to read a table.
-------------	--

8.17.3.11 setDefaultTableReadDelegate() [2/3]

```
template<typename T , IrisErrorCode(T::*)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) METHOD>
void iris::IrisInstanceBuilder::setDefaultTableReadDelegate (
```

```
    T * instance ) [inline]
```

Set the default table read function for all subsequently added tables.

Tables that do not explicitly override the access function using

```
addTable(...).setReadDelegate(...)
```

will use this delegate.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode readTable(const iris::TableInfo &tableInfo, uint64_t index,
                                  uint64_t count, iris::TableReadResult &result);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableReadDelegate<MyClass, &MyClass::readTable>(&myInstanceOfMyClass);
builder->addTable(...); // Uses readTable
```

Template Parameters

<i>T</i>	Class that defines a table read delegate method.
<i>METHOD</i>	A method of class T which is a table read delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

8.17.3.12 setDefaultTableReadDelegate() [3/3]

```
void iris::IrisInstanceBuilder::setDefaultTableReadDelegate (
    TableReadDelegate delegate = TableReadDelegate() ) [inline]
```

Set the default table read function for all subsequently added tables.

Tables that do not explicitly override the access function using

```
addTable(...).setReadDelegate(...)
```

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns E_↔

not_implemented for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode readTable(const iris::TableInfo &tableInfo, uint64_t index,
                                uint64_t count, iris::TableReadResult &result);
};
MyClass myInstanceOfMyClass;
iris::TableReadDelegate delegate =
    iris::TableReadDelegate::make<MyClass, &MyClass::readTable>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableReadDelegate(delegate);
builder->addTable(...); // Uses readTable
```

Parameters

<i>delegate</i>	Delegate object to call to read a table.
-----------------	--

8.17.3.13 setDefaultTableWriteDelegate() [1/3]

```
template<IrisErrorCode(*)>(const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>
void iris::IrisInstanceBuilder::setDefaultTableWriteDelegate ( ) [inline]
```

Set the default table write function for all subsequently added tables.

Tables that do not explicitly override the access function using

`addTable(...).setWriteDelegate(...)`

will use this delegate.

Usage:

```
iris::IrisErrorCode writeTable(const iris::TableInfo &tableInfo,
                              const iris::TableRecords &records,
                              iris::TableWriteResult &result);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableWriteDelegate<&writeTable>();
builder->addTable(...); // Uses writeTable
```

Template Parameters

<i>FUNC</i>	Global function to call to write a table.
-------------	---

8.17.3.14 setDefaultTableWriteDelegate() [2/3]

```
template<typename T , IrisErrorCode(T::*)(const TableInfo &, const TableRecords &, TableWriteResult &) METHOD>
void iris::IrisInstanceBuilder::setDefaultTableWriteDelegate (
    T * instance ) [inline]
```

Set the default table write function for all subsequently added tables.

Tables that do not explicitly override the access function using

`addTable(...).setWriteDelegate(...)`

will use this delegate.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode writeTable(const iris::TableInfo &tableInfo,
                                  const iris::TableRecords &records,
                                  iris::TableWriteResult &result);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableWriteDelegate<MyClass, &MyClass::writeTable>(&myInstanceOfMyClass);
builder->addTable(...); // Uses writeTable
```

Template Parameters

<i>T</i>	Class that defines a table write delegate method.
----------	---

Template Parameters

<i>METHOD</i>	A method of class T which is a table write delegate.
---------------	--

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

8.17.3.15 setDefaultTableWriteDelegate() [3/3]

```
void iris::IrisInstanceBuilder::setDefaultTableWriteDelegate (
    TableWriteDelegate delegate = TableWriteDelegate() ) [inline]
```

Set the default table write function for all subsequently added tables.

Tables that do not explicitly override the access function using

```
addTable(...).setWriteDelegate(...)
```

will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↵` not_implemented for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode writeTable(const iris::TableInfo &tableInfo,
                                  const iris::TableRecords &records,
                                  iris::TableWriteResult &result);
};
MyClass myInstanceOfMyClass;
iris::TableWriteDelegate delegate =
    iris::TableWriteDelegate::make<MyClass, &MyClass::writeTable>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableWriteDelegate(delegate);
builder->addTable(...); // Uses writeTable
```

Parameters

<i>delegate</i>	Delegate object to call to write a table.
-----------------	---

8.17.3.16 setExecutionStateGetDelegate() [1/3]

```
template<IrisErrorCode(*) (bool &) FUNC>
void iris::IrisInstanceBuilder::setExecutionStateGetDelegate ( ) [inline]
```

Set the delegate to get the execution state for this instance.

Usage:

```
iris::IrisErrorCode getState(bool &execution_enabled);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateGetDelegate<getState>();
```

Template Parameters

<i>FUNC</i>	Global function to call to get the execution state.
-------------	---

8.17.3.17 setExecutionStateGetDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setExecutionStateGetDelegate (
    PerInstanceExecutionStateGetDelegate delegate ) [inline]
```

Set the delegate to get the execution state for this instance.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↵` `not_implemented` for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getState(bool &execution_enabled);
};
MyClass myInstanceOfMyClass;
iris::PerInstanceExecutionStateGetDelegate delegate =
    iris::PerInstanceExecutionStateGetDelegate::make<MyClass, &MyClass::getState>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateGetDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object to call to get the execution state.
-----------------	---

8.17.3.18 setExecutionStateGetDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(bool &) METHOD>
void iris::IrisInstanceBuilder::setExecutionStateGetDelegate (
    T * instance ) [inline]
```

Set the delegate to get the execution state for this instance.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getState(bool &execution_enabled);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateGetDelegate<MyClass, &MyClass::getState>(&myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines a get execution state delegate method.
<i>METHOD</i>	A method of class T which is a get execution state delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

8.17.3.19 setExecutionStateSetDelegate() [1/3]

```
template<IrisErrorCode(*) (bool) FUNC>
void iris::IrisInstanceBuilder::setExecutionStateSetDelegate ( ) [inline]
```

Set the delegate to set the execution state for this instance.

Usage:

```
iris::IrisErrorCode setState(bool enable_execution);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateSetDelegate<&setState>();
```

Template Parameters

<i>FUNC</i>	Global function to call to set the execution state.
-------------	---

8.17.3.20 setExecutionStateSetDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setExecutionStateSetDelegate (
    PerInstanceExecutionStateSetDelegate delegate = PerInstanceExecutionStateSetDelegate()
) [inline]
```

Set the delegate to set the execution state for this instance.

Passing an empty delegate (the default argument) restores the default implementation which always returns `E_↔` not_implemented for all requests.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setState(bool enable_execution);
};
MyClass myInstanceOfMyClass;
iris::PerInstanceExecutionStateSetDelegate delegate =
    iris::PerInstanceExecutionStateSetDelegate::make<MyClass, &MyClass::setState>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateSetDelegate(delegate);
```

Parameters

<i>delegate</i>	Delegate object to call to set the execution state.
-----------------	---

8.17.3.21 setExecutionStateSetDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(bool) METHOD>
void iris::IrisInstanceBuilder::setExecutionStateSetDelegate (
    T * instance ) [inline]
```

Set the delegate to set the execution state for this instance.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setState(bool enable_execution);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateSetDelegate<MyClass, &MyClass::setState>(&myInstanceOfMyClass);
```

Template Parameters

<i>T</i>	Class that defines a set execution state delegate method.
<i>METHOD</i>	A method of class T which is a set execution state delegate.

Parameters

<i>instance</i>	An instance of class T on which METHOD should be called.
-----------------	--

8.17.3.22 setGetCurrentDisassemblyModeDelegate()

```
void iris::IrisInstanceBuilder::setGetCurrentDisassemblyModeDelegate (
    GetCurrentDisassemblyModeDelegate delegate ) [inline]
```

disass_apis [IrisInstanceBuilder](#) disassembler APIs

Set the delegates to get the current disassembly mode

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

- [IrisInstanceBuilder.h](#)

8.18 iris::IrisInstanceCheckpoint Class Reference

Checkpoint add-on for [IrisInstance](#).

```
#include <IrisInstanceCheckpoint.h>
```

Public Member Functions

- void [attachTo](#) ([IrisInstance](#) *iris_instance_)
Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
- [IrisInstanceCheckpoint](#) ([IrisInstance](#) *iris_instance=nullptr)
- void [setCheckpointRestoreDelegate](#) ([CheckpointRestoreDelegate](#) delegate)
Set checkpoint restore delegate for all checkpoints related to this instance.
- void [setCheckpointSaveDelegate](#) ([CheckpointSaveDelegate](#) delegate)
Set checkpoint save delegate for all checkpoints related to this instance.

8.18.1 Detailed Description

Checkpoint add-on for [IrisInstance](#).

8.18.2 Member Function Documentation

8.18.2.1 attachTo()

```
void iris::IrisInstanceCheckpoint::attachTo (
    IrisInstance * iris_instance_ )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

Only use this method if nullptr was passed to the constructor.

Parameters

<i>iris_↔ instance_</i>	The IrisInstance to attach to.
-----------------------------	--

8.18.2.2 setCheckpointRestoreDelegate()

```
void iris::IrisInstanceCheckpoint::setCheckpointRestoreDelegate (
    CheckpointRestoreDelegate delegate )
```

Set checkpoint restore delegate for all checkpoints related to this instance.

Parameters

<i>delegate</i>	A CheckpointRestoreDelegate to call when restoring a checkpoint.
-----------------	--

8.18.2.3 setCheckpointSaveDelegate()

```
void iris::IrisInstanceCheckpoint::setCheckpointSaveDelegate (
    CheckpointSaveDelegate delegate )
```

Set checkpoint save delegate for all checkpoints related to this instance.

Parameters

<i>delegate</i>	A CheckpointSaveDelegate to call when saving a checkpoint.
-----------------	--

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

- [IrisInstanceCheckpoint.h](#)

8.19 iris::IrisInstanceDebuggableState Class Reference

Debuggable-state add-on for [IrisInstance](#).

```
#include <IrisInstanceDebuggableState.h>
```

Public Member Functions

- void [attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
- [IrisInstanceDebuggableState](#) ([IrisInstance](#) *iris_instance=nullptr)
- void [setGetAcknowledgeDelegate](#) ([DebuggableStateGetAcknowledgeDelegate](#) delegate)
Set the get acknowledge flag delegate.
- void [setSetRequestDelegate](#) ([DebuggableStateSetRequestDelegate](#) delegate)
Set the set request flag delegate.

8.19.1 Detailed Description

Debuggable-state add-on for [IrisInstance](#).

8.19.2 Member Function Documentation

8.19.2.1 attachTo()

```
void iris::IrisInstanceDebuggableState::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

Parameters

<i>irisInstance</i>	The IrisInstance to attach to.
---------------------	--

8.19.2.2 setGetAcknowledgeDelegate()

```
void iris::IrisInstanceDebuggableState::setGetAcknowledgeDelegate (
    DebuggableStateGetAcknowledgeDelegate delegate ) [inline]
```

Set the get acknowledge flag delegate.

Parameters

<i>delegate</i>	Delegate that will be called to get the debuggable-state acknowledge flag.
-----------------	--

8.19.2.3 setSetRequestDelegate()

```
void iris::IrisInstanceDebuggableState::setSetRequestDelegate (
    DebuggableStateSetRequestDelegate delegate ) [inline]
```

Set the set request flag delegate.

Parameters

<i>delegate</i>	Delegate that will be called to set or clear the debuggable-state request flag.
-----------------	---

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

- [IrisInstanceDebuggableState.h](#)

8.20 iris::IrisInstanceDisassembler Class Reference

Disassembler add-on for [IrisInstance](#).

```
#include <IrisInstanceDisassembler.h>
```

Public Member Functions

- void [addDisassemblyMode](#) (const std::string &name, const std::string &description)
Add a disassembly mode.
- void [attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
- [IrisInstanceDisassembler](#) ([IrisInstance](#) *irisInstance=nullptr)
Construct an [IrisInstanceDisassembler](#).
- void [setDisassembleOpcodeDelegate](#) ([DisassembleOpcodeDelegate](#) delegate)
Set the delegate to get the disassembly of Opcode.
- void [setGetCurrentModeDelegate](#) ([GetCurrentDisassemblyModeDelegate](#) delegate)
Set the delegate to get the current disassembly mode.
- void [setGetDisassemblyDelegate](#) ([GetDisassemblyDelegate](#) delegate)
Set the delegate to get the disassembly of a chunk of memory.

8.20.1 Detailed Description

Disassembler add-on for [IrisInstance](#).

This class is used by instances that want to support disassembly functionality.

It implements all [IrisDisassembler*](#)() functions.

Example usage:

```
irisInstanceDisassembler = new iris::IrisInstanceDisassembler(irisInstance);
irisInstanceDisassembler->setGetCurrentModeDelegate(dasmCurrentModeGetDel); // Get the current disassembly
mode
irisInstanceDisassembler->setGetDisassemblyDelegate(dasmDisassemblyGetDel); // Get the disassembly of a
chunk of memory
irisInstanceDisassembler->setDisassembleOpcodeDelegate(dasmOpcodeDasmGetDel); // Disassemble specific
opcode
```

See [DummyComponent.h](#) for a working example.

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

- [IrisInstanceDisassembler.h](#)

8.21 iris::IrisInstanceEvent Class Reference

Event add-on for [IrisInstance](#).

```
#include <IrisInstanceEvent.h>
```

Classes

- struct [EventSourceInfoAndDelegate](#)
Contains the metadata and delegates for a single EventSource.
- struct [ProxyEventInfo](#)
Contains information for a single proxy EventSource.

Public Member Functions

- `uint64_t addEventSource` (const [EventSourceInfoAndDelegate](#) &info)
Add metadata for an event source.
- [EventSourceInfoAndDelegate](#) & `addEventSource` (const std::string &name, bool isHidden=false)
Add metadata for an event source.
- `void attachTo` ([IrisInstance](#) *irisInstance)
Attach this [IrisInstanceEvent](#) add-on to a specific [IrisInstance](#).
- `void deleteEventSource` (const std::string &eventName)
Delete metadata for an event source.
- `void destroyAllEventStreams` ()
Destroy all event streams.
- `bool destroyEventStream` (EventStreamId esId)
Destroy event stream (direct variant of `eventStream_destroy()`).
- [EventSourceInfoAndDelegate](#) & `enhanceEventSource` (const std::string &name)
Enhance existing event source.
- `void eventBufferClear` (EventBufferId evBufId)
Clear event buffer.
- `const uint64_t * eventBufferGetSyncStepResponse` (EventBufferId evBufId, RequestId requestId)
Get response to `step_syncStep()`, containing event data.
- `const EventSourceInfo * getEventSourceInfo` (EventSourceId evSrcId) const
Get [EventSourceInfo](#) for [EventSourceId](#).
- `bool hasEventSource` (const std::string &eventName)
Check if event source already exists.
- `IrisInstanceEvent` ([IrisInstance](#) *irisInstance=nullptr)
Construct an [IrisInstanceEvent](#) add-on.
- `bool isValidEvBufId` (EventBufferId evBufId) const
Check whether event buffer id is valid.
- `void renameEventSource` (const std::string &name, const std::string &newName)
Rename existing event source.
- `void setDefaultEsCreateDelegate` ([EventStreamCreateDelegate](#) delegate)
Set the default delegate for creating [EventStreams](#) for the attached instance.

Friends

- struct [EventBuffer](#)

8.21.1 Detailed Description

Event add-on for [IrisInstance](#).

This class is used by instances to support event functionality. Generally, there are two kinds of event sources:

- Iris-specific event sources. These are defined in the Iris spec, for example `IRIS_BREAKPOINT_HIT` and `IRIS_SIMULATION_TIME_EVENT`.
- Simulation-specific event sources. These are not defined in the Iris spec. They could be quite different for different simulations or instances. For example `INST` (every instruction executed).

This class implements all Iris `event*()` functions. It maintains event source information that is added by [addEventSource\(\)](#) and exposed by `event_getEventSources()` or `event_getEventSource()`. This class maintains all event streams. Iris-specific event streams are created by this add-on. Simulation-specific event streams are created by a delegate, which could be different for different simulations or instances.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 IrisInstanceEvent()

```
iris::IrisInstanceEvent::IrisInstanceEvent (
    IrisInstance * irisInstance = nullptr )
```

Construct an [IrisInstanceEvent](#) add-on.

Parameters

<i>irisInstance</i>	The IrisInstance to which to attach this add-on.
---------------------	--

8.21.3 Member Function Documentation

8.21.3.1 addEventSource() [1/2]

```
uint64_t iris::IrisInstanceEvent::addEventSource (
    const EventSourceInfoAndDelegate & info )
```

Add metadata for an event source.

Parameters

<i>info</i>	The metadata and event-specific delegates (if applicable) for a new event to add.
-------------	---

Returns

The evSrcId of the newly added event source.

8.21.3.2 addEventSource() [2/2]

```
EventSourceInfoAndDelegate & iris::IrisInstanceEvent::addEventSource (
    const std::string & name,
    bool isHidden = false )
```

Add metadata for an event source.

Parameters

<i>name</i>	The name of the event source.
<i>isHidden</i>	If true, this event source is hidden. The EventSourceInfo is not included in the list of event sources returned by <code>event_getEventSources()</code> but can still be accessed by <code>event_getEventSource()</code> if the client knows the name of the hidden event.

Returns

A reference to an object which keeps the metadata and event-specific delegates (if applicable) for this event. The reference is valid until the next call to [addEventSource\(\)](#).

8.21.3.3 attachTo()

```
void iris::IrisInstanceEvent::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstanceEvent](#) add-on to a specific [IrisInstance](#).

This should only be used if no instance was attached when this object was constructed.

Parameters

<i>irisInstance</i>	The IrisInstance to which to attach this add-on.
---------------------	--

8.21.3.4 deleteEventSource()

```
void iris::IrisInstanceEvent::deleteEventSource (
    const std::string & eventName )
```

Delete metadata for an event source.

Parameters

<i>eventName</i>	The name of the event source.
------------------	-------------------------------

8.21.3.5 destroyAllEventStreams()

```
void iris::IrisInstanceEvent::destroyAllEventStreams ( )
```

Destroy all event streams.

All event streams are always automatically destroyed when [IrisInstance](#) (and so [IrisInstanceEvent](#)) is destroyed. This function allows to destroy all event streams to be destroyed before [IrisInstance](#).

This is necessary when the event streams use other resources (like MTI traces) which go out of scope before [IrisInstance](#) does.

8.21.3.6 destroyEventStream()

```
bool iris::IrisInstanceEvent::destroyEventStream (
    EventStreamId esId )
```

Destroy event stream (direct variant of `eventStream_destroy()`).

If the event stream id is valid, disable and destroy the event stream. If disabling the event stream fails this is silently ignored (unlike `eventStream_destroy()`).

Returns

True if the event stream id was valid, else false.

8.21.3.7 enhanceEventSource()

```
EventSourceInfoAndDelegate & iris::IrisInstanceEvent::enhanceEventSource (
    const std::string & name )
```

Enhance existing event source.

Parameters

<i>name</i>	The name of the event source.
-------------	-------------------------------

Returns

A reference to an object which keeps the metadata and event-specific delegates (if applicable) for this event. The reference is valid until the next call to [addEventSource\(\)](#).

8.21.3.8 eventBufferClear()

```
void iris::IrisInstanceEvent::eventBufferClear (
```

```
EventBufferId evBufId )
```

Clear event buffer.

This is separate from [eventBufferGetSyncStepResponse\(\)](#) so the message writer can be used to send the message without taking an unnecessary copy.

Parameters

<i>evBufId</i>	The event buffer which is to be cleared.
----------------	--

8.21.3.9 eventBufferGetSyncStepResponse()

```
const uint64_t * iris::IrisInstanceEvent::eventBufferGetSyncStepResponse (
    EventBufferId evBufId,
    RequestId requestId )
```

Get response to step_syncStep(), containing event data.

Parameters

<i>evBufId</i>	The data of this event buffer is returned. This is set beforehand with step_syncStepSetup().
<i>requestId</i>	This is the request id of the original step_syncStep() for which this function generates the answer.

Returns

Response message to step_syncStep() call, containing the event data.

8.21.3.10 getEventSourceInfo()

```
const EventSourceInfo * iris::IrisInstanceEvent::getEventSourceInfo (
    EventSourceId evSrcId ) const
```

Get EventSourceInfo for EventSourceId.

Returns nullptr if the event source id is not found.

8.21.3.11 hasEventSource()

```
bool iris::IrisInstanceEvent::hasEventSource (
    const std::string & eventName )
```

Check if event source already exists.

Parameters

<i>eventName</i>	The name of the event source.
------------------	-------------------------------

Returns

True iff event source already exists.

8.21.3.12 isValidEvBufId()

```
bool iris::IrisInstanceEvent::isValidEvBufId (
    EventBufferId evBufId ) const
```

Check whether event buffer id is valid.

This function is use to validate event buffer ids.

Returns

Returns true iff evBufId is a valid event buffer id.

8.21.3.13 renameEventSource()

```
void iris::IrisInstanceEvent::renameEventSource (
    const std::string & name,
    const std::string & newName )
```

Rename existing event source.

If an event source "newName" already exists, it is deleted/overwritten.

Parameters

<i>name</i>	The old name of the event source.
<i>newName</i>	The new name of the event source.

8.21.3.14 setDefaultEsCreateDelegate()

```
void iris::IrisInstanceEvent::setDefaultEsCreateDelegate (
    EventStreamCreateDelegate delegate )
```

Set the default delegate for creating EventStreams for the attached instance.

Parameters

<i>delegate</i>	A delegate that will be called to create an event stream for event sources in the attached instance that have not set an event source-specific delegate.
-----------------	--

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

- [IrisInstanceEvent.h](#)

8.22 iris::IrisInstanceFactoryBuilder Class Reference

A builder class to construct instantiation parameter metadata.

#include <IrisInstanceFactoryBuilder.h>

Inherited by [iris::IrisPluginFactoryBuilder](#).

Public Member Functions

- [IrisParameterBuilder addBoolParameter](#) (const std::string &name, const std::string &description)
Add a new boolean parameter.
- [IrisParameterBuilder addHiddenBoolParameter](#) (const std::string &name, const std::string &description)
Add a new hidden boolean parameter.
- [IrisParameterBuilder addHiddenParameter](#) (const std::string &name, uint64_t bitWidth, const std::string &description)
Add a new hidden numeric parameter.
- [IrisParameterBuilder addHiddenStringParameter](#) (const std::string &name, const std::string &description)
Add a new hidden string parameter.
- [IrisParameterBuilder addParameter](#) (const std::string &name, uint64_t bitWidth, const std::string &description)

- Add a new numeric parameter.*
- [IrisParameterBuilder addStringParameter](#) (const std::string &name, const std::string &description)
- Add a new string parameter.*
- const std::vector< ResourceInfo > & [getHiddenParameterInfo](#) () const
- Get all ResourceInfo for hidden parameters.*
- const std::vector< ResourceInfo > & [getParameterInfo](#) () const
- Get all ResourceInfo for non-hidden parameters.*
- **IRIS_DEPRECATED** ("use [addBoolParameter](#)() instead") IrisParameterBuilder addBooleanParameter(const std
- **IRIS_DEPRECATED** ("use [addHiddenBoolParameter](#)() instead") IrisParameterBuilder addHiddenBooleanParameter(const std
- [IrisInstanceFactoryBuilder](#) (const std::string &prefix)
- Construct an [IrisInstanceFactoryBuilder](#).*

8.22.1 Detailed Description

A builder class to construct instantiation parameter metadata.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 IrisInstanceFactoryBuilder()

```
iris::IrisInstanceFactoryBuilder::IrisInstanceFactoryBuilder (
    const std::string & prefix ) [inline]
```

Construct an [IrisInstanceFactoryBuilder](#).

Parameters

<i>prefix</i>	All parameters added to this builder are prefixed with this string.
---------------	---

8.22.3 Member Function Documentation

8.22.3.1 addBoolParameter()

```
IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addBoolParameter (
    const std::string & name,
    const std::string & description ) [inline]
```

Add a new boolean parameter.

Boolean parameters are numeric parameters with a bitWidth of 1 and "true" and "false" enum symbols.

Parameters

<i>name</i>	Name of the parameter.
<i>description</i>	Description of the parameter.

Returns

An [IrisParameterBuilder](#) object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.22.3.2 addHiddenBoolParameter()

```
IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addHiddenBoolParameter (
    const std::string & name,
    const std::string & description ) [inline]
```

Add a new hidden boolean parameter.

Boolean parameters are numeric parameters with a bitWidth of 1 and "true" and "false" enum symbols.

Parameters

<i>name</i>	Name of the parameter.
<i>description</i>	Description of the parameter.

Returns

An [IrisParameterBuilder](#) object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.22.3.3 addHiddenParameter()

```
IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addHiddenParameter (
    const std::string & name,
    uint64_t bitWidth,
    const std::string & description ) [inline]
```

Add a new hidden numeric parameter.

Parameters

<i>name</i>	Name of the parameter.
<i>bitWidth</i>	Width of the parameter in bits.
<i>description</i>	Description of the parameter.

Returns

An [IrisParameterBuilder](#) object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.22.3.4 addHiddenStringParameter()

```
IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addHiddenStringParameter (
    const std::string & name,
    const std::string & description ) [inline]
```

Add a new hidden string parameter.

Parameters

<i>name</i>	Name of the parameter.
<i>description</i>	Description of the parameter.

Returns

An [IrisParameterBuilder](#) object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.22.3.5 addParameter()

```
IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addParameter (
    const std::string & name,
    uint64_t bitWidth,
    const std::string & description ) [inline]
```

Add a new numeric parameter.

Parameters

<i>name</i>	Name of the parameter.
<i>bitWidth</i>	Width of the parameter in bits.
<i>description</i>	Description of the parameter.

Returns

An [IrisParameterBuilder](#) object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.22.3.6 addStringParameter()

```
IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addStringParameter (
    const std::string & name,
    const std::string & description ) [inline]
```

Add a new string parameter.

Parameters

<i>name</i>	Name of the parameter.
<i>description</i>	Description of the parameter.

Returns

An [IrisParameterBuilder](#) object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.22.3.7 getHiddenParameterInfo()

```
const std::vector< ResourceInfo > & iris::IrisInstanceFactoryBuilder::getHiddenParameterInfo (
) const [inline]
```

Get all ResourceInfo for hidden parameters.

Returns

A vector of ResourceInfo. Iterators for this vector are invalidated if a new hidden parameter is added.

8.22.3.8 getParameterInfo()

```
const std::vector< ResourceInfo > & iris::IrisInstanceFactoryBuilder::getParameterInfo (
) const [inline]
```

Get all ResourceInfo for non-hidden parameters.

Returns

A vector of ResourceInfo. Iterators for this vector are invalidated if a new non-hidden parameter is added.

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

- [IrisInstanceFactoryBuilder.h](#)

8.23 iris::IrisInstanceImage Class Reference

Image loading add-on for [IrisInstance](#).

```
#include <IrisInstanceImage.h>
```

Public Member Functions

- void [attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
- [IrisInstanceImage](#) ([IrisInstance](#) *irisInstance=0)
Construct a new [IrisInstanceImage](#).
- void [setLoadImageDataDelegate](#) ([ImageLoadDataDelegate](#) delegate)
Set image loading from (pushed/pulled) data delegate.
- void [setLoadImageFileDelegate](#) ([ImageLoadFileDelegate](#) delegate)
Set image loading from file delegate.

Static Public Member Functions

- static IrisErrorCode [readFileData](#) (const std::string &fileName, std::vector< uint8_t > &data)
Read file data into a uint8_t array.

8.23.1 Detailed Description

Image loading add-on for [IrisInstance](#).

This class is used by instances to support image loading. It is also used by instances that want to use `image_loadDataPull()` to implement the `image_loadDataRead()` callback.

This class implements the `iris_image*()` functions. It maintains or implements two main things:

- Functions to load images:
 - From a file, by `image_loadFile()`, or from a data buffer, by `image_loadData()` or `image_loadDataPull()`.
 - As raw data, by specifying `rawAddr` and `rawSpaceId`.
- Image meta information, which is exposed by `image_getMetaInfoList()` or cleared by `image_clearMetaInfoList()`.

See DummyComponent.h for a working example.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 IrisInstanceImage()

```
iris::IrisInstanceImage::IrisInstanceImage (
    IrisInstance * irisInstance = 0 )
```

Construct a new [IrisInstanceImage](#).

Parameters

<i>irisInstance</i>	The IrisInstance to attach this add-on to.
---------------------	--

8.23.3 Member Function Documentation

8.23.3.1 attachTo()

```
void iris::IrisInstanceImage::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

Parameters

<i>irisInstance</i>	The IrisInstance to attach this add-on to.
---------------------	--

8.23.3.2 readFileData()

```
static IrisErrorCode iris::IrisInstanceImage::readFileData (
    const std::string & fileName,
    std::vector< uint8_t > & data ) [static]
```

Read file data into a `uint8_t` array.

Parameters

<i>fileName</i>	Name of the file to read.
<i>data</i>	A reference to a vector which is populated with the file contents.

Returns

An error code indicating success or failure.

8.23.3.3 setLoadImageDataDelegate()

```
void iris::IrisInstanceImage::setLoadImageDataDelegate (
    ImageLoadDataDelegate delegate )
```

Set image loading from (pushed/pulled) data delegate.

Parameters

<i>delegate</i>	The delegate that will be called to load an image from a data buffer.
-----------------	---

8.23.3.4 setLoadImageFileDelegate()

```
void iris::IrisInstanceImage::setLoadImageFileDelegate (
    ImageLoadFileDelegate delegate )
```

Set image loading from file delegate.

Parameters

<i>delegate</i>	The delegate that will be called to load an image from a file.
-----------------	--

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

- [IrisInstancelImage.h](#)

8.24 iris::IrisInstanceImage_Callback Class Reference

Image loading add-on for [IrisInstance](#) clients implementing `image_loadDataRead()`.

```
#include <IrisInstanceImage.h>
```

Public Member Functions

- void [attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
- [IrisInstanceImage_Callback](#) ([IrisInstance](#) *irisInstance=0)
Construct an [IrisInstanceImage_Callback](#) add-on.
- uint64_t [openImage](#) (const std::string &fileName)
Open an image for reading.

Protected Member Functions

- void [impl_image_loadDataRead](#) (IrisReceivedRequest &request)
Implementation of the [Iris](#) function [image_loadDataRead\(\)](#).

8.24.1 Detailed Description

Image loading add-on for [IrisInstance](#) clients implementing `image_loadDataRead()`.

This is used by instances that call the instances supporting `image_loadDataPull()`.

This class maintains/implements:

- [Iris](#) `image_loadDataRead()` function.
- Image opening, data reading.
- Tags of images.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 IrisInstanceImage_Callback()

```
iris::IrisInstanceImage_Callback::IrisInstanceImage_Callback (
    IrisInstance * irisInstance = 0 )
```

Construct an [IrisInstanceImage_Callback](#) add-on.

Parameters

<i>irisInstance</i>	The IrisInstance to attach this add-on to.
---------------------	--

8.24.3 Member Function Documentation

8.24.3.1 attachTo()

```
void iris::IrisInstanceImage_Callback::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

Parameters

<i>irisInstance</i>	The IrisInstance to attach this add-on to.
---------------------	--

8.24.3.2 openImage()

```
uint64_t iris::IrisInstanceImage_Callback::openImage (
    const std::string & fileName )
```

Open an image for reading.

Parameters

<i>fileName</i>	File name of the image file to read.
-----------------	--------------------------------------

Returns

An opaque tag number that is passed to `image_loadDataRead()` to identify the file to read from. This returns `iris::IRIS_UINT64_MAX` on failure to open the image.

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

- [IrisInstanceImage.h](#)

8.25 iris::IrisInstanceMemory Class Reference

Memory add-on for [IrisInstance](#).

```
#include <IrisInstanceMemory.h>
```

Classes

- struct [AddressTranslationInfoAndAccess](#)
Contains static address translation information.
- struct [SpaceInfoAndAccess](#)
Entry in 'spaceInfos'.

Public Member Functions

- [AddressTranslationInfoAndAccess](#) & [addAddressTranslation](#) (MemorySpaceId inSpaceId, MemorySpaceId outSpaceId, const std::string &description)
Add one memory address translation as well as the translate interface.
- [SpaceInfoAndAccess](#) & [addMemorySpace](#) (const std::string &name)
Add meta information for one memory space.
- void [attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
- [IrisInstanceMemory](#) ([IrisInstance](#) *irisInstance=0)
Construct an [IrisInstanceMemory](#).
- void [setDefaultGetSidebandInfoDelegate](#) ([MemoryGetSidebandInfoDelegate](#) delegate=[MemoryGetSidebandInfoDelegate](#)())
Set the default delegate to retrieve sideband information.
- void [setDefaultReadDelegate](#) ([MemoryReadDelegate](#) delegate=[MemoryReadDelegate](#)())
Set default read function for all subsequently added memory spaces.
- void [setDefaultTranslateDelegate](#) ([MemoryAddressTranslateDelegate](#) delegate=[MemoryAddressTranslateDelegate](#)())
Set the default memory translation delegate.
- void [setDefaultWriteDelegate](#) ([MemoryWriteDelegate](#) delegate=[MemoryWriteDelegate](#)())
Set default write function for all subsequently added memory spaces.

8.25.1 Detailed Description

Memory add-on for [IrisInstance](#).

This class is used by instances to expose their own memory.

It implements all `Iris memory*()` functions. It maintains/implements two main things:

- Memory space meta information (exposed by `memory_getMemorySpaces()`).
- Forwarding memory read/write and address translate accesses to functions with a simple prototype which is easy to implement by components, hiding a lot of the complexity of `memory_read()`, `memory_write()`, and `memory_translateAddress()`.

Example usage:

```
irisInstance = new iris::IrisInstance(irisInterface, instanceName);
irisInstanceMemory = new iris::IrisInstanceMemory(irisInstance);
// Use these delegates for read/write for all following memory spaces.
irisInstanceMemory->setDefaultReadDelegate<DummyComponent, &DummyComponent::readMemory>(this);
irisInstanceMemory->setDefaultWriteDelegate<DummyComponent, &DummyComponent::writeMemory>(this);
irisInstanceMemory->addMemorySpace("Memory"); // Add a memory address space.
```

See [setDefaultReadDelegate\(\)](#) for an example of read/write delegates.

See `DummyComponent.h` for a working example.

See also

[IrisInstanceBuilder](#) [memory APIs](#)

8.25.2 Constructor & Destructor Documentation

8.25.2.1 IrisInstanceMemory()

```
iris::IrisInstanceMemory::IrisInstanceMemory (
    IrisInstance * irisInstance = 0 )
```

Construct an [IrisInstanceMemory](#).

Optionally attaches to an [IrisInstance](#).

Parameters

<i>irisInstance</i>	The IrisInstance to attach to.
---------------------	--

8.25.3 Member Function Documentation

8.25.3.1 addAddressTranslation()

```
AddressTranslationInfoAndAccess & iris::IrisInstanceMemory::addAddressTranslation (
    MemorySpaceId inSpaceId,
    MemorySpaceId outSpaceId,
    const std::string & description )
```

Add one memory address translation as well as the translate interface.

Parameters

<i>inSpaceId</i>	Memory space id for the input memory space of this translation.
<i>out↔SpaceId</i>	Memory space id for the output memory space of this translation.
<i>description</i>	A human-readable description of this translation.

Returns

A reference to an [AddressTranslationInfoAndAccess](#) object for the new translation. This reference is valid until the next time [addAddressTranslation\(\)](#) is called.

8.25.3.2 addMemorySpace()

```
SpaceInfoAndAccess & iris::IrisInstanceMemory::addMemorySpace (
    const std::string & name )
```

Add meta information for one memory space.

Parameters

<i>name</i>	Name of the memory space.
-------------	---------------------------

Returns

A reference to a [SpaceInfoAndAccess](#) object for this new memory space. This reference is valid until the next time [addMemorySpace\(\)](#) is called.

8.25.3.3 attachTo()

```
void iris::IrisInstanceMemory::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

Parameters

<i>irisInstance</i>	The IrisInstance to attach to.
---------------------	--

8.25.3.4 setDefaultGetSidebandInfoDelegate()

```
void iris::IrisInstanceMemory::setDefaultGetSidebandInfoDelegate (
    MemoryGetSidebandInfoDelegate delegate = MemoryGetSidebandInfoDelegate() ) [inline]
```

Set the default delegate to retrieve sideband information.

Parameters

<i>delegate</i>	Delegate object which will be called to get sideband information for a memory space.
-----------------	--

8.25.3.5 setDefaultReadDelegate()

```
void iris::IrisInstanceMemory::setDefaultReadDelegate (
    MemoryReadDelegate delegate = MemoryReadDelegate() ) [inline]
```

Set default read function for all subsequently added memory spaces.

Parameters

<i>delegate</i>	Delegate object which will be called to read memory.
-----------------	--

8.25.3.6 setDefaultTranslateDelegate()

```
void iris::IrisInstanceMemory::setDefaultTranslateDelegate (
    MemoryAddressTranslateDelegate delegate = MemoryAddressTranslateDelegate() )
[inline]
```

Set the default memory translation delegate.

Parameters

<i>delegate</i>	Delegate object which will be called to translate addresses.
-----------------	--

8.25.3.7 setDefaultWriteDelegate()

```
void iris::IrisInstanceMemory::setDefaultWriteDelegate (
    MemoryWriteDelegate delegate = MemoryWriteDelegate() ) [inline]
```

Set default write function for all subsequently added memory spaces.

Parameters

<i>delegate</i>	Delegate object which will be called to write memory.
-----------------	---

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

- [IrisInstanceMemory.h](#)

8.26 iris::IrisInstancePerInstanceExecution Class Reference

Per-instance execution control add-on for [IrisInstance](#).

```
#include <IrisInstancePerInstanceExecution.h>
```

Public Member Functions

- void [attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstancePerInstanceExecution](#) add-on to a specific [IrisInstance](#).
- [IrisInstancePerInstanceExecution](#) ([IrisInstance](#) *irisInstance=nullptr)
Construct an [IrisInstancePerInstanceExecution](#) add-on.
- void [setExecutionStateGetDelegate](#) ([PerInstanceExecutionStateGetDelegate](#) delegate)
Set the delegate for getting execution state.
- void [setExecutionStateSetDelegate](#) ([PerInstanceExecutionStateSetDelegate](#) delegate)
Set the delegate for setting execution state.

8.26.1 Detailed Description

Per-instance execution control add-on for [IrisInstance](#).

This class is used by instances to support per-instance execution control functionality.

This class implements all [Iris perInstanceExecution*\(\)](#) functions.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 IrisInstancePerInstanceExecution()

```
iris::IrisInstancePerInstanceExecution::IrisInstancePerInstanceExecution (
    IrisInstance * irisInstance = nullptr )
```

Construct an [IrisInstancePerInstanceExecution](#) add-on.

Parameters

<i>irisInstance</i>	The IrisInstance to attach this add-on to.
---------------------	--

8.26.3 Member Function Documentation

8.26.3.1 attachTo()

```
void iris::IrisInstancePerInstanceExecution::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstancePerInstanceExecution](#) add-on to a specific [IrisInstance](#).

This should only be used if no instance was attached when this object was constructed.

Parameters

<i>irisInstance</i>	The IrisInstance to attach this add-on to.
---------------------	--

8.26.3.2 setExecutionStateGetDelegate()

```
void iris::IrisInstancePerInstanceExecution::setExecutionStateGetDelegate (
    PerInstanceExecutionStateGetDelegate delegate )
```

Set the delegate for getting execution state.

Parameters

<i>delegate</i>	A delegate object which will be called to get the current execution state for the attached instance.
-----------------	--

8.26.3.3 setExecutionStateSetDelegate()

```
void iris::IrisInstancePerInstanceExecution::setExecutionStateSetDelegate (
    PerInstanceExecutionStateSetDelegate delegate )
```

Set the delegate for setting execution state.

Parameters

<i>delegate</i>	A delegate object which will be called to set execution state for the attached instance.
-----------------	--

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

- [IrisInstancePerInstanceExecution.h](#)

8.27 iris::IrisInstanceResource Class Reference

Resource add-on for [IrisInstance](#).

```
#include <IrisInstanceResource.h>
```

Classes

- struct [ResourceInfoAndAccess](#)

Entry in 'resourceInfos'.

Public Member Functions

- [ResourceInfoAndAccess](#) & [addResource](#) (const std::string &type, const std::string &name, const std::string &description)
Add a new resource.
- void [attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
- void [beginResourceGroup](#) (const std::string &name, const std::string &description, uint64_t startSubRscId=IRIS_UINT64_MAX, const std::string &cname=std::string())
Begin a new resource group.
- [ResourceInfoAndAccess](#) * [getResourceInfo](#) (ResourceId rsclId)
Get the resource info for a resource that was already added.
- [IrisInstanceResource](#) ([IrisInstance](#) *irisInstance=0)
Construct an [IrisInstanceResource](#).
- void [setNextSubRscId](#) (ResourceId nextSubRscId_)
Set next subRscId.
- void [setTag](#) (ResourceId rsclId, const std::string &tag)
Set a tag for a specific resource.

Static Public Member Functions

- static void [calcHierarchicalNames](#) (std::vector< ResourceInfo > &resourceInfos)
Calculate hierarchicalName and hierarchicalCName for all RegisterInfos.
- static void [makeNamesHierarchical](#) (std::vector< ResourceInfo > &resourceInfos)
Make name and cname of RegisterInfos hierarchical.

Protected Member Functions

- void [impl_resource_getList](#) (IrisReceivedRequest &request)
- void [impl_resource_getListOfResourceGroups](#) (IrisReceivedRequest &request)
- void [impl_resource_getResourceInfo](#) (IrisReceivedRequest &request)
- void [impl_resource_read](#) (IrisReceivedRequest &request)
- void [impl_resource_write](#) (IrisReceivedRequest &request)

8.27.1 Detailed Description

Resource add-on for [IrisInstance](#).

This class implements all Iris resource*() functions. It maintains/implements two main things:

- Resource meta information that is exposed by [resource_getList\(\)](#) and [resource_getListOfResourceGroups\(\)](#).
- Forwarding resource read/write accesses to functions with a simple prototype which is easy to implement by components, hiding a lot of the complexity of [resource_read\(\)](#) and [resource_write\(\)](#).

In most cases, an instance should not use [IrisInstanceResource](#) directly but should use [IrisInstanceBuilder](#) instead.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 IrisInstanceResource()

```
iris::IrisInstanceResource::IrisInstanceResource (
    IrisInstance * irisInstance = 0 )
```

Construct an [IrisInstanceResource](#).

Optionally attaches to an [IrisInstance](#).

Parameters

<i>irisInstance</i>	The IrisInstance to attach to.
---------------------	--

8.27.3 Member Function Documentation

8.27.3.1 addResource()

```
ResourceInfoAndAccess & iris::IrisInstanceResource::addResource (
    const std::string & type,
    const std::string & name,
    const std::string & description )
```

Add a new resource.

Parameters

<i>type</i>	The type of the resource. This should be one of: <ul style="list-style-type: none"> • "numeric" • "numericFp" • "String" • "noValue"
<i>name</i>	The name of the resource.
<i>description</i>	A human-readable description of the resource.

Returns

A reference to a [ResourceInfoAndAccess](#) object for this new resource. This reference is valid until the next time [addResource\(\)](#) is called.

8.27.3.2 attachTo()

```
void iris::IrisInstanceResource::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

Parameters

<i>irisInstance</i>	The IrisInstance to attach to.
---------------------	--

8.27.3.3 beginResourceGroup()

```
void iris::IrisInstanceResource::beginResourceGroup (
    const std::string & name,
    const std::string & description,
    uint64_t startSubRscId = IRIS_UINT64_MAX,
    const std::string & cname = std::string() )
```

Begin a new resource group.

This method has these effects:

- Add a resource group (only if it does not yet exist).
- Assign all resources that are added through [addResource\(\)](#) calls to this group.

Parameters

<i>name</i>	The name of the resource group.
<i>description</i>	A description of this resource group.
<i>startSub↔ RscId</i>	If not IRIS_UINT64_MAX start counting from this subRscId when new resources are added.
<i>cname</i>	A C identifier version of the resource name if different from <i>name</i> .

8.27.3.4 calcHierarchicalNames()

```
static void iris::IrisInstanceResource::calcHierarchicalNames (
    std::vector< ResourceInfo > & resourceInfos ) [static]
```

Calculate hierarchicalName and hierarchicalCName for all RegisterInfos.

RegisterInfo.hierarchicalName and RegisterInfo.hierarchicalCName are set to the hierarchical name for each resource such that a child register X of parent FLAGS gets hierarchicalName=FLAGS.X and hierarchical↔CName=FLAGS_X, similarly also for deeper nesting levels.

This functionality is not an Iris interface but just a convenience function for simple clients. The ResourceInfos returned by [IrisInstance::getResourceInfo*\(\)](#) have already hierarchical names.

No errors are generated for missing parent resources. parentRscId links to missing parent resources are silently ignored. The intended usage is to call this function on a list containing all resources or all registers of an instance, so that all parent links can be resolved.

Parameters

<i>resourceInfos</i>	Array of all ResourceInfos of an instance.
----------------------	--

8.27.3.5 getResourceInfo()

```
ResourceInfoAndAccess * iris::IrisInstanceResource::getResourceInfo (
    ResourceId rscId )
```

Get the resource info for a resource that was already added.

Parameters

<i>rsc↔ Id</i>	A resource id for a resource that was already added.
--------------------	--

Returns

A pointer to the [ResourceInfoAndAccess](#) object for the requested resource. This pointer is valid until the next call to [addResource\(\)](#). If *rscId* is not a valid id, this function returns nullptr.

8.27.3.6 makeNamesHierarchical()

```
static void iris::IrisInstanceResource::makeNamesHierarchical (
    std::vector< ResourceInfo > & resourceInfos ) [static]
```

Make name and cname of RegisterInfos hierarchical.

Legacy function overwriting ResourceInfo.name/cname.

This function calculates the hierarchical names using [calcHierarchicalNames\(\)](#) and then copies ResourceInfo.↔ hierarchicalName/hierarchicalCName into ResourceInfo.name/cname info, respectively.

Consider using [calcHierarchicalNames\(\)](#) which does not alter the original resource information.

Parameters

<i>resourceInfos</i>	Array of all ResourceInfos of an instance.
----------------------	--

8.27.3.7 setNextSubRscId()

```
void iris::IrisInstanceResource::setNextSubRscId (
    ResourceId nextSubRscId_ ) [inline]
```

Set next subRscId.

Resources that are added following this call are assigned subRscIds starting at nextSubRscId unless nextSubRscId is IRIS_UINT64_MAX, in which case all further resources are assigned IRIS_UINT64_MAX as the subRscId

Parameters

<i>nextSubRsc↔ Id_</i>	Next subRscId
----------------------------	------------------

8.27.3.8 setTag()

```
void iris::IrisInstanceResource::setTag (
    ResourceId rscId,
    const std::string & tag )
```

Set a tag for a specific resource.

Parameters

<i>rsc↔ Id</i>	Resource Id for the resource that will have this tag set.
<i>tag</i>	Name of the boolean tag which will be set to true.

See also

[IrisInstanceBuilder::setTag](#)

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

- [IrisInstanceResource.h](#)

8.28 iris::IrisInstanceSemihosting Class Reference

Public Member Functions

- void [attachTo](#) ([IrisInstance](#) *iris_instance)

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

- void **enableExtensions** ()

Instances that support semihosting extensions should call this method to enable the IRIS_SEMIHOSTING_↔ CALL_EXTENSION event.

- **IrisInstanceSemihosting** ([IrisInstance](#) *iris_instance=nullptr, [IrisInstanceEvent](#) *inst_event=nullptr)
- std::vector< uint8_t > [readData](#) (uint64_t fDes, uint64_t max_size=0, uint64_t flags=semihost::DEFAULT)
Read data for a given file descriptor.
- std::pair< bool, uint64_t > [semihostedCall](#) (uint64_t operation, uint64_t parameter)
Allow a client to perform a semihosting extension defined by operation and parameter.
- void [setEventHandler](#) ([IrisInstanceEvent](#) *handler)
Set the corresponding [IrisInstanceEvent](#) object to use to manage semihosting events.
- void **unblock** ()
Request premature exit from any blocking requests that are currently blocked.
- bool **writeData** (uint64_t fDes, const uint8_t *data, uint64_t size)

8.28.1 Member Function Documentation

8.28.1.1 attachTo()

```
void iris::IrisInstanceSemihosting::attachTo (
    IrisInstance * iris_instance )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

Parameters

<i>iris_instance</i>	The instance to attach to.
----------------------	----------------------------

8.28.1.2 readData()

```
std::vector< uint8_t > iris::IrisInstanceSemihosting::readData (
    uint64_t fDes,
    uint64_t max_size = 0,
    uint64_t flags = semihost::DEFAULT )
```

Read data for a given file descriptor.

The exact behavior of this method depends on the value of the max_size and flags parameters. If the NONBLOCK flag is set, the method returns immediately with whatever data is already buffered, if any. If NONBLOCK is not set, the method blocks until data is available. Iris messages continue to be processed while this methods blocks. If max_size is not zero, then at most max_size bytes will be returned.

Parameters

<i>fDes</i>	File descriptor to read from. Usually semihost::STDIN.
<i>max_size</i>	The maximum amount of bytes to read or zero for no limit.
<i>flags</i>	A bitwise OR of Semihosting data request flag constants

Returns

A vector of data that was read.

8.28.1.3 semihostedCall()

```
std::pair< bool, uint64_t > iris::IrisInstanceSemihosting::semihostedCall (
    uint64_t operation,
    uint64_t parameter )
```

Allow a client to perform a semihosting extension defined by *operation* and *parameter*.

This might implement a user-defined operation or override the default implementation for a predefined operation.

Parameters

<i>operation</i>	A number indicating the operation to perform. This is defined by the semihosting standard for standard operations or by the client for user-defined operations.
<i>parameter</i>	A parameter to the operation. This meaning of this parameter is defined by the operation.

Returns

A pair of (bool success, uint64_t result). If status is true, a client performed the function and returned the value in result. If status is false, no client performed the function and result is 0.

8.28.1.4 setEventHandler()

```
void iris::IrisInstanceSemihosting::setEventHandler (
    IrisInstanceEvent * handler )
```

Set the corresponding [IrisInstanceEvent](#) object to use to manage semihosting events.

This must not be called more than once and must be called with an Event add-on that is attached to the same [IrisInstance](#) as this semihosting add-on.

Parameters

<i>handler</i>	The event add-on for this Iris instance.
----------------	--

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

- [IrisInstanceSemihosting.h](#)

8.29 iris::IrisInstanceSimulation Class Reference

An [IrisInstance](#) add-on that adds simulation functions for the SimulationEngine instance.

```
#include <IrisInstanceSimulation.h>
```

Public Member Functions

- void [attachTo](#) ([IrisInstance](#) *iris_instance)
Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
- void [enterPostInstantiationPhase](#) ()
Move from the pre-instantiation to the post-instantiation phase.
- [IrisInstanceSimulation](#) ([IrisInstance](#) *iris_instance=nullptr, [IrisConnectionInterface](#) *connection_interface=nullptr)
Construct an [IrisInstanceSimulation](#) add-on.
- void [notifySimPhase](#) (uint64_t time, [IrisSimulationPhase](#) phase)
Emit an IRIS_SIM_PHASE event for the supplied phase.*
- void [registerSimEventsOnGlobalInstance](#) ()
Register all simulation engine events as proxy events on the global iris instance.
- void [setConnectionInterface](#) ([IrisConnectionInterface](#) *connection_interface_)
Set the [IrisConnectionInterface](#) to use for the instantiation.
- void [setEventHandler](#) ([IrisInstanceEvent](#) *handler)
Set up IRIS_SIM_PHASE events.*
- template<[IrisErrorCode](#)(*)(std::vector< [ResourceInfo](#) > &) FUNC>
void [setGetParameterInfoDelegate](#) (bool cache_result=true)
Set the [getParameterInfo\(\)](#) delegate.
- void [setGetParameterInfoDelegate](#) ([SimulationGetParameterInfoDelegate](#) delegate, bool cache_result=true)

- Set the `getParameterInfo()` delegate.*

 - `template<typename T , IrisErrorCode(T::*)(std::vector< ResourceInfo > &) METHOD>`
`void setGetParameterInfoDelegate (T *instance, bool cache_result=true)`

Set the `getParameterInfo()` delegate.

 - `template<IrisErrorCode(*) (InstantiationResult &) FUNC>`
`void setInstantiateDelegate ()`

Set the `instantiate()` delegate.

 - `void setInstantiateDelegate (SimulationInstantiateDelegate delegate)`

Set the `instantiate()` delegate.

 - `template<typename T , IrisErrorCode(T::*)(InstantiationResult &) METHOD>`
`void setInstantiateDelegate (T *instance)`

Set the `instantiate()` delegate.

 - `void setLogLevel (unsigned logLevel_)`

Set log level (0-1).

 - `template<IrisErrorCode(*)() FUNC>`
`void setRequestShutdownDelegate ()`

Set the `requestShutdown()` delegate.

 - `void setRequestShutdownDelegate (SimulationRequestShutdownDelegate delegate)`

Set the `requestShutdown()` delegate.

 - `template<typename T , IrisErrorCode(T::*)() METHOD>`
`void setRequestShutdownDelegate (T *instance)`

Set the `requestShutdown()` delegate.

 - `template<IrisErrorCode(*) (const IrisSimulationResetContext &) FUNC>`
`void setResetDelegate ()`

Set the `reset()` delegate.

 - `void setResetDelegate (SimulationResetDelegate delegate)`

Set the `reset()` delegate.

 - `template<typename T , IrisErrorCode(T::*)(const IrisSimulationResetContext &) METHOD>`
`void setResetDelegate (T *instance)`

Set the `reset()` delegate.

 - `template<IrisErrorCode(*) (const InstantiationParameterValue &) FUNC>`
`void setSetParameterValueDelegate ()`

Set the `setParameterValue()` delegate.

 - `void setSetParameterValueDelegate (SimulationSetParameterValueDelegate delegate)`

Set the `setParameterValue()` delegate.

 - `template<typename T , IrisErrorCode(T::*)(const InstantiationParameterValue &) METHOD>`
`void setSetParameterValueDelegate (T *instance)`

Set the `setParameterValue()` delegate.

Static Public Member Functions

- `static std::string getSimulationPhaseDescription (IrisSimulationPhase phase)`
Get description string for a simulation phase.
- `static std::string getSimulationPhaseName (IrisSimulationPhase phase)`
Get name of the enum symbol for name.

8.29.1 Detailed Description

An [IrisInstance](#) add-on that adds simulation functions for the `SimulationEngine` instance.

8.29.2 Constructor & Destructor Documentation

8.29.2.1 IrisInstanceSimulation()

```
iris::IrisInstanceSimulation::IrisInstanceSimulation (
    IrisInstance * iris_instance = nullptr,
    IrisConnectionInterface * connection_interface = nullptr )
```

Construct an [IrisInstanceSimulation](#) add-on.

Parameters

<i>iris_instance</i>	The IrisInstance to attach this add-on to.
<i>connection_interface</i>	The connection interface that will be used when the simulation is instantiated.

8.29.3 Member Function Documentation

8.29.3.1 attachTo()

```
void iris::IrisInstanceSimulation::attachTo (
    IrisInstance * iris_instance )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

Parameters

<i>iris_instance</i>	The IrisInstance to attach to.
----------------------	--

8.29.3.2 enterPostInstantiationPhase()

```
void iris::IrisInstanceSimulation::enterPostInstantiationPhase ( )
```

Move from the pre-instantiation to the post-instantiation phase.

This effects which functions are published. Only call this function if the simulation is instantiated outside of Iris. This object automatically enters post-instantiation phase when the simulation is successfully instantiated by an Iris call to `simulation_instantiate()`.

8.29.3.3 getSimulationPhaseDescription()

```
static std::string iris::IrisInstanceSimulation::getSimulationPhaseDescription (
    IrisSimulationPhase phase ) [static]
```

Get description string for a simulation phase.

This is a free form single line text ending with a dot.

8.29.3.4 getSimulationPhaseName()

```
static std::string iris::IrisInstanceSimulation::getSimulationPhaseName (
    IrisSimulationPhase phase ) [static]
```

Get name of the enum symbol for name.

Example: `getSimulationPhaseName(IRIS_SIM_PHASE_INIT)` returns "IRIS_SIM_PHASE_INIT".

8.29.3.5 notifySimPhase()

```
void iris::IrisInstanceSimulation::notifySimPhase (
    uint64_t time,
    IrisSimulationPhase phase )
```

Emit an `IRIS_SIM_PHASE*` event for the supplied phase.

Parameters

<i>time</i>	The simulation time at which the event occurred.
<i>phase</i>	The simulation phase that was reached.

8.29.3.6 registerSimEventsOnGlobalInstance()

```
void iris::IrisInstanceSimulation::registerSimEventsOnGlobalInstance ( )
```

Register all simulation engine events as proxy events on the global iris instance.

This function should be called after an iris instance has been attached to [IrisInstanceSimulation](#) object ([IrisInstanceSimulation::attachTo](#)). This will ensure that the simulation engine iris instance i.e. `iris_instance` is available to call the register API. This function should be called after event handler has been set for [IrisInstanceSimulation](#) object ([IrisInstanceSimulation::setEventHandler](#)). This will ensure that all simulation engine events are available in simulation engine event handler. This function should be called after an `IrisInstanceEvent` has been attached to `iris_instance` ([IrisInstanceEvent::attachTo](#)). This will ensure that event functions have been registered on simulation engine iris instance.

8.29.3.7 setConnectionInterface()

```
void iris::IrisInstanceSimulation::setConnectionInterface (
    IrisConnectionInterface * connection_interface_ ) [inline]
```

Set the `IrisConnectionInterface` to use for the instantiation.

This will be passed to the `instantiate()` delegate when the simulation is instantiated.

8.29.3.8 setEventHandler()

```
void iris::IrisInstanceSimulation::setEventHandler (
    IrisInstanceEvent * handler )
```

Set up `IRIS_SIM_PHASE*` events.

Parameters

<i>handler</i>	An IrisInstanceEvent add-on that is attached to the same instance as this add-on.
----------------	---

8.29.3.9 setGetParameterInfoDelegate() [1/3]

```
template<IrisErrorCode(*) (std::vector< ResourceInfo > &) FUNC>
void iris::IrisInstanceSimulation::setGetParameterInfoDelegate (
    bool cache_result = true ) [inline]
```

Set the `getParameterInfo()` delegate.

Set the delegate to a global function.

Template Parameters

<i>FUNC</i>	A function that is a <code>getParameterInfo</code> delegate.
-------------	--

Parameters

<i>cache_result</i>	If true, the delegate is only called once and the result is cached and used for subsequent calls to <code>simulation_getInstantiationParameterInfo()</code> . If false, the result is not cached and the delegate is called every time.
---------------------	---

8.29.3.10 setGetParameterInfoDelegate() [2/3]

```
void iris::IrisInstanceSimulation::setGetParameterInfoDelegate (
    SimulationGetParameterInfoDelegate delegate,
    bool cache_result = true ) [inline]
```

Set the getParameterInfo() delegate.

Parameters

<i>delegate</i>	A delegate object that is called to get instantiation parameter information for the simulation.
<i>cache_result</i>	If true, the delegate is only called once and the result is cached and used for subsequent calls to <code>simulation_getInstantiationParameterInfo()</code> . If false, the result is not cached and the delegate is called every time.

8.29.3.11 setGetParameterInfoDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(std::vector< ResourceInfo > &) METHOD>
void iris::IrisInstanceSimulation::setGetParameterInfoDelegate (
    T * instance,
    bool cache_result = true ) [inline]
```

Set the getParameterInfo() delegate.

Set the delegate to call a method in class T.

Template Parameters

<i>T</i>	Class that defines a getParameterInfo delegate method.
<i>METHOD</i>	A method of class <i>T</i> that is a getParameterInfo delegate.

Parameters

<i>instance</i>	An instance of class <i>T</i> on which <i>METHOD</i> should be called.
<i>cache_result</i>	If true, the delegate is called once and the result is cached and used for subsequent calls to <code>simulation_getInstantiationParameterInfo()</code> . If false, the result is not cached and the delegate is called every time.

8.29.3.12 setInstantiateDelegate() [1/3]

```
template<IrisErrorCode(*) (InstantiationResult &) FUNC>
void iris::IrisInstanceSimulation::setInstantiateDelegate ( ) [inline]
```

Set the instantiate() delegate.

Set the delegate to a global function.

Template Parameters

<i>FUNC</i>	A function that is an instantiate delegate.
-------------	---

8.29.3.13 setInstantiateDelegate() [2/3]

```
void iris::IrisInstanceSimulation::setInstantiateDelegate (
    SimulationInstantiateDelegate delegate ) [inline]
```

Set the instantiate() delegate.

Parameters

<i>delegate</i>	A delegate object that will be called to instantiate the simulation.
-----------------	--

8.29.3.14 setInstantiateDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(InstantiationResult &) METHOD>
void iris::IrisInstanceSimulation::setInstantiateDelegate (
    T * instance ) [inline]
```

Set the instantiate() delegate.

Set the delegate to call a method in class T.

Template Parameters

<i>T</i>	Class that defines an instantiate delegate method.
<i>METHOD</i>	A method of class <i>T</i> that is an instantiate delegate.

Parameters

<i>instance</i>	An instance of class <i>T</i> on which <i>METHOD</i> should be called.
-----------------	--

8.29.3.15 setLogLevel()

```
void iris::IrisInstanceSimulation::setLogLevel (
    unsigned logLevel_ )
```

Set log level (0-1).

Set log level (0-1).

8.29.3.16 setRequestShutdownDelegate() [1/3]

```
template<IrisErrorCode(*)() FUNC>
void iris::IrisInstanceSimulation::setRequestShutdownDelegate ( ) [inline]
```

Set the requestShutdown() delegate.

Set the delegate to a global function.

Template Parameters

<i>FUNC</i>	A function that is a requestShutdown delegate.
-------------	--

8.29.3.17 setRequestShutdownDelegate() [2/3]

```
void iris::IrisInstanceSimulation::setRequestShutdownDelegate (
    SimulationRequestShutdownDelegate delegate ) [inline]
```

Set the requestShutdown() delegate.

Parameters

<i>delegate</i>	A delegate object that will be called to request that the simulation be shut down.
-----------------	--

8.29.3.18 setRequestShutdownDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)() METHOD>
void iris::IrisInstanceSimulation::setRequestShutdownDelegate (
    T * instance ) [inline]
```

Set the requestShutdown() delegate.

Set the delegate to call a method in class T.

Template Parameters

<i>T</i>	Class that defines a requestShutdown delegate method.
<i>METHOD</i>	A method of class <i>T</i> that is a requestShutdown delegate.

Parameters

<i>instance</i>	An instance of class <i>T</i> on which <i>METHOD</i> should be called.
-----------------	--

8.29.3.19 setResetDelegate() [1/3]

```
template<IrisErrorCode(*) (const IrisSimulationResetContext &) FUNC>
void iris::IrisInstanceSimulation::setResetDelegate ( ) [inline]
```

Set the reset() delegate.

Set the delegate to a global function.

Template Parameters

<i>FUNC</i>	A function that is a reset delegate.
-------------	--------------------------------------

8.29.3.20 setResetDelegate() [2/3]

```
void iris::IrisInstanceSimulation::setResetDelegate (
    SimulationResetDelegate delegate ) [inline]
```

Set the reset() delegate.

Parameters

<i>delegate</i>	A delegate object which will be called to reset the simulation.
-----------------	---

8.29.3.21 setResetDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*) (const IrisSimulationResetContext &) METHOD>
void iris::IrisInstanceSimulation::setResetDelegate (
    T * instance ) [inline]
```

Set the reset() delegate.

Set the delegate to call a method in class T.

Template Parameters

<i>T</i>	Class that defines a reset delegate method.
----------	---

Template Parameters

<i>METHOD</i>	A method of class <i>T</i> that is a reset delegate.
---------------	--

Parameters

<i>instance</i>	An instance of class <i>T</i> on which <i>METHOD</i> should be called.
-----------------	--

8.29.3.22 setSetParameterValueDelegate() [1/3]

```
template<IrisErrorCode(*) (const InstantiationParameterValue &) FUNC>
void iris::IrisInstanceSimulation::setSetParameterValueDelegate ( ) [inline]
```

Set the setParameterValue() delegate.
Set the delegate to a global function.

Template Parameters

<i>FUNC</i>	A function that is a setParameterValue delegate.
-------------	--

8.29.3.23 setSetParameterValueDelegate() [2/3]

```
void iris::IrisInstanceSimulation::setSetParameterValueDelegate (
    SimulationSetParameterValueDelegate delegate ) [inline]
```

Set the setParameterValue() delegate.

Parameters

<i>delegate</i>	A delegate object that is called to set instantiation parameter values before instantiation.
-----------------	--

8.29.3.24 setSetParameterValueDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*) (const InstantiationParameterValue &) METHOD>
void iris::IrisInstanceSimulation::setSetParameterValueDelegate (
    T * instance ) [inline]
```

Set the setParameterValue() delegate.
Set the delegate to call a method in class T.

Template Parameters

<i>T</i>	Class that defines a setParameterValue delegate method.
<i>METHOD</i>	A method of class <i>T</i> that is a setParameterValue delegate.

Parameters

<i>instance</i>	An instance of class <i>T</i> on which <i>METHOD</i> should be called.
-----------------	--

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

- [IrisInstanceSimulation.h](#)

8.30 iris::IrisInstanceSimulationTime Class Reference

Simulation time add-on for [IrisInstance](#).

```
#include <IrisInstanceSimulationTime.h>
```

Public Member Functions

- void [attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).
- [IrisInstanceSimulationTime](#) ([IrisInstance](#) *iris_instance=nullptr, [IrisInstanceEvent](#) *inst_event=nullptr)
Construct an [IrisInstanceSimulationTime](#) add-on.
- void [notifySimulationTimeEvent](#) (uint64_t reason=TIME_EVENT_UNKNOWN)
Generate the IRIS_SIMULATION_TIME_EVENT event callback.
- void [registerSimTimeEventsOnGlobalInstance](#) ()
Register all simulation time events as proxy events on the global iris instance.
- void [setEventHandler](#) ([IrisInstanceEvent](#) *handler)
Set the event handler to use to send simulation time-related events.
- template<IrisErrorCode(*)>(uint64_t &, uint64_t &, bool &) FUNC<
void [setSimTimeGetDelegate](#) ()
Set the getTime() delegate.
- void [setSimTimeGetDelegate](#) ([SimulationTimeGetDelegate](#) delegate)
Set the getTime() delegate.
- template<typename T , IrisErrorCode(T::*)>(uint64_t &, uint64_t &, bool &) METHOD<
void [setSimTimeGetDelegate](#) (T *instance)
Set the getTime() delegate.
- void [setSimTimeNotifyStateChanged](#) (std::function< void()> func)
Set the notifyStateChanged() delegate.
- template<IrisErrorCode(*)>() FUNC<
void [setSimTimeRunDelegate](#) ()
Set the run() delegate.
- void [setSimTimeRunDelegate](#) ([SimulationTimeRunDelegate](#) delegate)
Set the run() delegate.
- template<typename T , IrisErrorCode(T::*)>() METHOD<
void [setSimTimeRunDelegate](#) (T *instance)
Set the run() delegate.
- template<IrisErrorCode(*)>() FUNC<
void [setSimTimeStopDelegate](#) ()
Set the stop() delegate.
- void [setSimTimeStopDelegate](#) ([SimulationTimeStopDelegate](#) delegate)
Set the stop() delegate.
- template<typename T , IrisErrorCode(T::*)>() METHOD<
void [setSimTimeStopDelegate](#) (T *instance)
Set the stop() delegate.

8.30.1 Detailed Description

Simulation time add-on for [IrisInstance](#).

8.30.2 Constructor & Destructor Documentation

8.30.2.1 IrisInstanceSimulationTime()

```
iris::IrisInstanceSimulationTime::IrisInstanceSimulationTime (
    IrisInstance * iris_instance = nullptr,
    IrisInstanceEvent * inst_event = nullptr )
```

Construct an [IrisInstanceSimulationTime](#) add-on.

Parameters

<i>iris_instance</i>	An IrisInstance to attach this add-on to.
<i>inst_event</i>	An IrisInstanceEvent add-on that is already attached to IrisInstance . This is used to set up simulation time events.

8.30.3 Member Function Documentation

8.30.3.1 attachTo()

```
void iris::IrisInstanceSimulationTime::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstance](#) add-on to a specific [IrisInstance](#).

Parameters

<i>irisInstance</i>	An IrisInstance to attach this add-on to.
---------------------	---

8.30.3.2 registerSimTimeEventsOnGlobalInstance()

```
void iris::IrisInstanceSimulationTime::registerSimTimeEventsOnGlobalInstance ( )
```

Register all simulation time events as proxy events on the global iris instance.

This function should be called after an iris instance has been attached to [IrisInstanceSimulationTime](#) object ([IrisInstanceSimulationTime::attachTo](#)). This will ensure that the simulation time iris instance i.e. `iris_` instance is available to call the register API. This function should be called after event handler has been set for [IrisInstanceSimulationTime](#) object ([IrisInstanceSimulationTime::setEventHandler](#)). This will ensure that all simulation time events are available in simulation time event handler. This function should be called after an [IrisInstanceEvent](#) has been attached to `iris_instance` ([IrisInstanceEvent::attachTo](#)). This will ensure that event functions have been registered on simulation time iris instance.

8.30.3.3 setEventHandler()

```
void iris::IrisInstanceSimulationTime::setEventHandler (
    IrisInstanceEvent * handler )
```

Set the event handler to use to send simulation time-related events.

Parameters

<i>handler</i>	An IrisInstanceEvent add-on that is already attached to IrisInstance . This is used to set up simulation time events.
----------------	---

8.30.3.4 setSimTimeGetDelegate() [1/3]

```
template<IrisErrorCode(*) (uint64_t &, uint64_t &, bool &) FUNC>
void iris::IrisInstanceSimulationTime::setSimTimeGetDelegate ( ) [inline]
```

Set the getTime() delegate.
Set the delegate to a global function.

Template Parameters

<i>FUNC</i>	A function that is a getTime delegate.
-------------	--

8.30.3.5 setSimTimeGetDelegate() [2/3]

```
void iris::IrisInstanceSimulationTime::setSimTimeGetDelegate (
    SimulationTimeGetDelegate delegate ) [inline]
```

Set the getTime() delegate.

Parameters

<i>delegate</i>	A delegate that is called to get the current simulation time.
-----------------	---

8.30.3.6 setSimTimeGetDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(uint64_t &, uint64_t &, bool &) METHOD>
void iris::IrisInstanceSimulationTime::setSimTimeGetDelegate (
    T * instance ) [inline]
```

Set the getTime() delegate.

Template Parameters

<i>T</i>	Class that defines a getTime delegate method.
<i>METHOD</i>	A method of class <i>T</i> that is a getTime delegate.

Parameters

<i>instance</i>	An instance of class <i>T</i> on which <i>METHOD</i> should be called.
-----------------	--

8.30.3.7 setSimTimeNotifyStateChanged()

```
void iris::IrisInstanceSimulationTime::setSimTimeNotifyStateChanged (
    std::function< void()> func ) [inline]
```

Set the notifyStateChanged() delegate.

The semantics of this delegate is to emit a IRIS_SIMULATION_TIME_EVENT(REASON=STATE_CHANGED) event, usually by calling notifySimulationTimeEvent(TIME_EVENT_STATE_CHANGED). Ideally this is done with a small delay so that multiple successive calls to simulationTime_notifyStateChanged() cause only one IRIS_SIMULATION_TIME_EVENT(REASON=STATE_CHANGED) event. In other words multiple calls to simulationTime_notifyStateChanged() should be aggregated into one IRIS_SIMULATION_TIME_EVENT(REASON=STATE_CHANGED) event. The delay from the first call to simulationTime_notifyStateChanged() to the IRIS_SIMULATION_TIME_EVENT(REASON=STATE_CHANGED) event should be approximately 500 ms.

The default implementation of this delegate immediately emits a IRIS_SIMULATION_TIME_EVENT(REASON=STATE_CHANGED) event and does not aggregate multiple calls to simulationTime_notifyStateChanged().

Parameters

<i>func</i>	A function which calls notifySimulationTimeEvent() within the next 500 ms.
-------------	--

8.30.3.8 setSimTimeRunDelegate() [1/3]

```
template<IrisErrorCode(*)() FUNC>
void iris::IrisInstanceSimulationTime::setSimTimeRunDelegate ( ) [inline]
```

Set the run() delegate.

Set the delegate to a global function.

Template Parameters

<i>FUNC</i>	A function that is a run delegate.
-------------	------------------------------------

8.30.3.9 setSimTimeRunDelegate() [2/3]

```
void iris::IrisInstanceSimulationTime::setSimTimeRunDelegate (
    SimulationTimeRunDelegate delegate ) [inline]
```

Set the run() delegate.

Parameters

<i>delegate</i>	A delegate that is called to start/resume progress of simulation time.
-----------------	--

8.30.3.10 setSimTimeRunDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)() METHOD>
void iris::IrisInstanceSimulationTime::setSimTimeRunDelegate (
    T * instance ) [inline]
```

Set the run() delegate.

Template Parameters

<i>T</i>	Class that defines a run delegate method.
<i>METHOD</i>	A method of class <i>T</i> that is a run delegate.

Parameters

<i>instance</i>	An instance of class <i>T</i> on which <i>METHOD</i> should be called.
-----------------	--

8.30.3.11 setSimTimeStopDelegate() [1/3]

```
template<IrisErrorCode(*)() FUNC>
void iris::IrisInstanceSimulationTime::setSimTimeStopDelegate ( ) [inline]
```

Set the stop() delegate.

Set the delegate to a global function.

Template Parameters

<i>FUNC</i>	A function that is a stop delegate.
-------------	-------------------------------------

8.30.3.12 setSimTimeStopDelegate() [2/3]

```
void iris::IrisInstanceSimulationTime::setSimTimeStopDelegate (
    SimulationTimeStopDelegate delegate ) [inline]
```

Set the stop() delegate.

Parameters

<i>delegate</i>	A delegate that is called to stop the progress of simulation time.
-----------------	--

8.30.3.13 setSimTimeStopDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)() METHOD>
void iris::IrisInstanceSimulationTime::setSimTimeStopDelegate (
    T * instance ) [inline]
```

Set the stop() delegate.

Template Parameters

<i>T</i>	Class that defines a stop delegate method.
<i>METHOD</i>	A method of class <i>T</i> that is a stop delegate.

Parameters

<i>instance</i>	An instance of class <i>T</i> on which <i>METHOD</i> should be called.
-----------------	--

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

- [IrisInstanceSimulationTime.h](#)

8.31 iris::IrisInstanceStep Class Reference

Step add-on for [IrisInstance](#).

```
#include <IrisInstanceStep.h>
```

Public Member Functions

- void [attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstanceStep](#) add-on to a specific [IrisInstance](#).
- [IrisInstanceStep](#) ([IrisInstance](#) *irisInstance=nullptr)
Construct an [IrisInstanceStep](#) add-on.
- void [setRemainingStepGetDelegate](#) ([RemainingStepGetDelegate](#) delegate)
Set the delegate for getting the remaining steps.
- void [setRemainingStepSetDelegate](#) ([RemainingStepSetDelegate](#) delegate)
Set the delegate for setting the remaining steps.
- void [setStepCountGetDelegate](#) ([StepCountGetDelegate](#) delegate)
Set the delegate for getting the step count.

8.31.1 Detailed Description

Step add-on for [IrisInstance](#).

This is used by instances to support stepping functionality.

This class implements all Iris step*() functions.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 IrisInstanceStep()

```
iris::IrisInstanceStep::IrisInstanceStep (
    IrisInstance * irisInstance = nullptr )
```

Construct an [IrisInstanceStep](#) add-on.

Parameters

<i>irisInstance</i>	The IrisInstance to attach this add-on to.
---------------------	--

8.31.3 Member Function Documentation

8.31.3.1 attachTo()

```
void iris::IrisInstanceStep::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstanceStep](#) add-on to a specific [IrisInstance](#).

This should only be used if no instance was attached when this object was constructed.

Parameters

<i>irisInstance</i>	The IrisInstance to attach this add-on to.
---------------------	--

8.31.3.2 setRemainingStepGetDelegate()

```
void iris::IrisInstanceStep::setRemainingStepGetDelegate (
    RemainingStepGetDelegate delegate )
```

Set the delegate for getting the remaining steps.

Parameters

<i>delegate</i>	A delegate object that is called to get the remaining steps for the attached instance.
-----------------	--

8.31.3.3 setRemainingStepSetDelegate()

```
void iris::IrisInstanceStep::setRemainingStepSetDelegate (
    RemainingStepSetDelegate delegate )
```

Set the delegate for setting the remaining steps.

Parameters

<i>delegate</i>	A delegate object that is called to set the remaining steps for the attached instance.
-----------------	--

8.31.3.4 setStepCountGetDelegate()

```
void iris::IrisInstanceStep::setStepCountGetDelegate (
    StepCountGetDelegate delegate )
```

Set the delegate for getting the step count.

Parameters

<i>delegate</i>	A delegate object that is called to get the step count for the attached instance.
-----------------	---

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

- [IrisInstanceStep.h](#)

8.32 iris::IrisInstanceTable Class Reference

Table add-on for [IrisInstance](#).

```
#include <IrisInstanceTable.h>
```

Classes

- struct [TableInfoAndAccess](#)
Entry in 'tableInfos'.

Public Member Functions

- [TableInfoAndAccess](#) & [addTableInfo](#) (const std::string &name)
Add metadata for one table.
- void [attachTo](#) ([IrisInstance](#) *irisInstance)
Attach this [IrisInstanceTable](#) add-on to a specific [IrisInstance](#).
- [IrisInstanceTable](#) ([IrisInstance](#) *irisInstance=nullptr)
Construct an [IrisInstanceTable](#) add-on.
- void [setDefaultReadDelegate](#) ([TableReadDelegate](#) delegate=[TableReadDelegate](#)())
Set the default delegate for reading table data.
- void [setDefaultWriteDelegate](#) ([TableWriteDelegate](#) delegate=[TableWriteDelegate](#)())
Set the default delegate for writing table data.

8.32.1 Detailed Description

Table add-on for [IrisInstance](#).

This is used by instances to support table functionality.

8.32.2 Constructor & Destructor Documentation

8.32.2.1 IrisInstanceTable()

```
iris::IrisInstanceTable::IrisInstanceTable (
    IrisInstance * irisInstance = nullptr )
```

Construct an [IrisInstanceTable](#) add-on.

Parameters

<i>irisInstance</i>	The IrisInstance to attach this add-on to.
---------------------	--

8.32.3 Member Function Documentation

8.32.3.1 addTableInfo()

```
TableInfoAndAccess & iris::IrisInstanceTable::addTableInfo (
    const std::string & name )
```

Add metadata for one table.

Parameters

<i>name</i>	The name of this table.
-------------	-------------------------

Returns

A reference to a [TableInfoAndAccess](#) object that can be used to set metadata and access delegates for this table.

8.32.3.2 attachTo()

```
void iris::IrisInstanceTable::attachTo (
    IrisInstance * irisInstance )
```

Attach this [IrisInstanceTable](#) add-on to a specific [IrisInstance](#).

This should only be used if no instance was attached when this object was constructed.

Parameters

<i>irisInstance</i>	The IrisInstance to attach this add-on to.
---------------------	--

8.32.3.3 setDefaultReadDelegate()

```
void iris::IrisInstanceTable::setDefaultReadDelegate (
    TableReadDelegate delegate = TableReadDelegate() ) [inline]
```

Set the default delegate for reading table data.

Parameters

<i>delegate</i>	A delegate object that is called to read table data for tables in the attached instance that did not set a table-specific delegate.
-----------------	---

8.32.3.4 setDefaultWriteDelegate()

```
void iris::IrisInstanceTable::setDefaultWriteDelegate (
    TableWriteDelegate delegate = TableWriteDelegate() ) [inline]
```

Set the default delegate for writing table data.

Parameters

<i>delegate</i>	A delegate object that is called to write table data for tables in the attached instance that did not set a table-specific delegate.
-----------------	--

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

- [IrisInstanceTable.h](#)

8.33 iris::IrisInstantiationContext Class Reference

Provides context when instantiating an Iris instance from a factory.

```
#include <IrisInstantiationContext.h>
```

Public Member Functions

- void void void [error](#) (const std::string &code, const char *format,...) INTERNAL_IRIS_PRINTF(3)
Add an error to the InstantiationResult.
- bool [getBoolParameter](#) (const std::string &name)
Get the value of an instantiation parameter as boolean.
- IrisConnectionInterface * [getConnectionInterface](#) () const
Get the connection interface to use to register the instance being instantiated.
- std::string [getInstanceName](#) () const
Get the instance name to use when registering the instance being instantiated.
- const IrisValue & [getParameter](#) (const std::string &name)
Get the value of an instantiation parameter as IrisValue.
- void [getParameter](#) (const std::string &name, std::vector< uint64_t > &value)
Get the value of a large numeric instantiation parameter.
- template<typename T >
void [getParameter](#) (const std::string &name, T &value)
Get the value of an instantiation parameter.
- uint64_t [getRecommendedInstanceFlags](#) () const
Get the flags to use when registering the instance being instantiated.
- int64_t [getS64Parameter](#) (const std::string &name)
Get the value of an instantiation parameter as int64_t.
- std::string [getStringParameter](#) (const std::string &name)
Get the value of an instantiation parameter as string.
- [IrisInstantiationContext](#) * [getSubcomponentContext](#) (const std::string &child_name)
Get an IrisInstanceContext pointer for a subcomponent instance.
- uint64_t [getU64Parameter](#) (const std::string &name)
Get the value of an instantiation parameter as uint64_t.
- [IrisInstantiationContext](#) (IrisConnectionInterface *connection_interface_, InstantiationResult &result_↵
, const std::vector< ResourceInfo > ¶m_info_, const std::vector< InstantiationParameterValue >
¶m_values_, const std::string &prefix_, const std::string &component_name_, uint64_t instance_flags↵
)
- void void void void [parameterError](#) (const std::string &code, const std::string ¶meterName, const char
*format,...) INTERNAL_IRIS_PRINTF(4)
Add an error to the InstantiationResult.
- void void [parameterWarning](#) (const std::string &code, const std::string ¶meterName, const char
*format,...) INTERNAL_IRIS_PRINTF(4)
Add a warning to the InstantiationResult.
- void [warning](#) (const std::string &code, const char *format,...) INTERNAL_IRIS_PRINTF(3)
Add a warning to the InstantiationResult.

8.33.1 Detailed Description

Provides context when instantiating an Iris instance from a factory.

8.33.2 Member Function Documentation

8.33.2.1 error()

```
void void void iris::IrisInstantiationContext::error (
    const std::string & code,
    const char * format,
    ... )
```

Add an error to the InstantiationResult.

See also

[parameterError](#)

Parameters

<i>code</i>	An error code symbol. This should be one of the codes specified for the InstantiationError object.
<i>format</i>	A printf-style format string.
...	Printf substitution arguments.

8.33.2.2 getBoolParameter()

```
bool iris::IrisInstantiationContext::getBoolParameter (
    const std::string & name ) [inline]
```

Get the value of an instantiation parameter as boolean.

Parameters

<i>name</i>	The name of the parameter.
-------------	----------------------------

Returns

Boolean value.

8.33.2.3 getConnectionInterface()

```
IrisConnectionInterface * iris::IrisInstantiationContext::getConnectionInterface ( ) const
[inline]
```

Get the connection interface to use to register the instance being instantiated.

Returns

A value to use for the `connection_interface` argument of [IrisInstance::IrisInstance\(\)](#).

8.33.2.4 getInstanceName()

```
std::string iris::IrisInstantiationContext::getInstanceName ( ) const [inline]
```

Get the instance name to use when registering the instance being instantiated.

Returns

A value to use for the `instName` argument of [IrisInstance::IrisInstance\(\)](#) or [IrisInstance::registerInstance\(\)](#).

8.33.2.5 `getParameter()` [1/3]

```
const IrisValue & iris::IrisInstantiationContext::getParameter (
    const std::string & name ) [inline]
```

Get the value of an instantiation parameter as `IrisValue`.

This can be used as a fallback for all types not supported by the `get<type>Parameter()` functions below.

Parameters

<i>name</i>	The name of the parameter.
-------------	----------------------------

Returns

`IrisValue` of the parameter.

8.33.2.6 `getParameter()` [2/3]

```
void iris::IrisInstantiationContext::getParameter (
    const std::string & name,
    std::vector< uint64_t > & value )
```

Get the value of a large numeric instantiation parameter.

This is used for numeric parameters that are outside the range of `uint64_t/int64_t`.

Parameters

<i>name</i>	The name of the parameter.
<i>value</i>	A reference to a value of type <i>T</i> that receives the value of the named parameter.

8.33.2.7 `getParameter()` [3/3]

```
template<typename T >
void iris::IrisInstantiationContext::getParameter (
    const std::string & name,
    T & value ) [inline]
```

Get the value of an instantiation parameter.

Template Parameters

<i>T</i>	The type of the <i>value</i> . This must be a type that is appropriate to receive the value of this parameter.
----------	--

Parameters

<i>name</i>	The name of the parameter.
<i>value</i>	A reference to a value of type <i>T</i> that receives the value of the named parameter.

8.33.2.8 `getRecommendedInstanceFlags()`

```
uint64_t iris::IrisInstantiationContext::getRecommendedInstanceFlags ( ) const [inline]
```

Get the flags to use when registering the instance being instantiated.

Returns

A value to use for the flags argument of [IrisInstance::IrisInstance\(\)](#) or [IrisInstance::registerInstance\(\)](#).

8.33.2.9 getS64Parameter()

```
int64_t iris::IrisInstantiationContext::getS64Parameter (
    const std::string & name ) [inline]
```

Get the value of an instantiation parameter as int64_t.

Parameters

<i>name</i>	The name of the parameter.
-------------	----------------------------

Returns

S64 value.

8.33.2.10 getStringParameter()

```
std::string iris::IrisInstantiationContext::getStringParameter (
    const std::string & name ) [inline]
```

Get the value of an instantiation parameter as string.

Parameters

<i>name</i>	The name of the parameter.
-------------	----------------------------

Returns

String value.

8.33.2.11 getSubcomponentContext()

```
IrisInstantiationContext * iris::IrisInstantiationContext::getSubcomponentContext (
    const std::string & child_name )
```

Get an IrisInstanceContext pointer for a subcomponent instance.

For example, you might call `getSubcomponentContext("cpu0")` on the context "component.cluster0" to get the context to instantiate "component.cluster0.cpu0". The object pointed to by the return value is owned by its parent context and has the same lifetime as the parent context.

Parameters

<i>child_name</i>	The name of a child instance.
-------------------	-------------------------------

Returns

A pointer to an [IrisInstantiationContext](#) object for the named child.

8.33.2.12 getU64Parameter()

```
uint64_t iris::IrisInstantiationContext::getU64Parameter (
    const std::string & name ) [inline]
```

Get the value of an instantiation parameter as `uint64_t`.

Parameters

<i>name</i>	The name of the parameter.
-------------	----------------------------

Returns

U64 value.

8.33.2.13 parameterError()

```
void void void void iris::IrisInstantiationContext::parameterError (
    const std::string & code,
    const std::string & parameterName,
    const char * format,
    ... )
```

Add an error to the InstantiationResult.

See also

[error](#)

Parameters

<i>code</i>	An error code symbol. This should be one of the codes specified for the InstantiationError object.
<i>parameterName</i>	The name of the parameter this error relates to.
<i>format</i>	A printf-style format string.
...	Printf substitution arguments.

8.33.2.14 parameterWarning()

```
void void iris::IrisInstantiationContext::parameterWarning (
    const std::string & code,
    const std::string & parameterName,
    const char * format,
    ... )
```

Add a warning to the InstantiationResult.

See also

[warning](#)

Parameters

<i>code</i>	An error code symbol. This should be one of the codes specified for the InstantiationError object.
<i>parameterName</i>	The name of the parameter this warning relates to.
<i>format</i>	A printf-style format string.
...	Printf substitution arguments.

8.33.2.15 warning()

```
void iris::IrisInstantiationContext::warning (
    const std::string & code,
    const char * format,
    ... )
```

Add a warning to the InstantiationResult.

See also

[parameterWarning](#)

Parameters

<i>code</i>	An error code symbol. This should be one of the codes specified for the InstantiationError object.
<i>format</i>	A printf-style format string.
...	Printf substitution arguments.

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

- [IrisInstantiationContext.h](#)

8.34 iris::IrisNonFactoryPlugin< PLUGIN_CLASS > Class Template Reference

Wrapper to instantiate a non-factory plugin.

```
#include <IrisPluginFactory.h>
```

Public Member Functions

- **IrisNonFactoryPlugin** (IrisC_Funcions *functions, const std::string &pluginName)

Static Public Member Functions

- static int64_t **initPlugin** (IrisC_Funcions *functions, const std::string &pluginName)

8.34.1 Detailed Description

```
template<class PLUGIN_CLASS>
class iris::IrisNonFactoryPlugin< PLUGIN_CLASS >
```

Wrapper to instantiate a non-factory plugin.

Do not use this directly. Use the IRIS_NON_FACTORY_PLUGIN macro instead.

Template Parameters

<i>PLUGIN_CLASS</i>	Plugin class.
---------------------	---------------

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

- [IrisPluginFactory.h](#)

8.35 iris::IrisParameterBuilder Class Reference

Helper class to construct instantiation parameters.

```
#include <IrisParameterBuilder.h>
```

Public Member Functions

- [IrisParameterBuilder](#) & [addEnum](#) (const std::string &symbol, const IrisValue &value, const std::string &description=std::string())
Add an enum symbol for this parameter.
- [IrisParameterBuilder](#) & [addStringEnum](#) (const std::string &value, const std::string &description=std::string())
Add a string enum symbol for this parameter.
- [IrisParameterBuilder](#) (ResourceInfo &info_)
Construct a parameter builder for a given parameter resource.
- [IrisParameterBuilder](#) & [setBitWidth](#) (uint64_t bitWidth)
*Set the *bitWidth* field.*
- [IrisParameterBuilder](#) & [setDefault](#) (const std::string &value)
Set the default value for a string parameter.
- [IrisParameterBuilder](#) & [setDefault](#) (const std::vector< uint64_t > &value)
Set the default value for a numeric parameter.
- [IrisParameterBuilder](#) & [setDefault](#) (uint64_t value)
Set the default value for a numeric parameter.
- [IrisParameterBuilder](#) & [setDefaultFloat](#) (double value)
Set the default value for a numericFp parameter.
- [IrisParameterBuilder](#) & [setDefaultSigned](#) (const std::vector< uint64_t > &value)
Set the default value for a numericSigned parameter.
- [IrisParameterBuilder](#) & [setDefaultSigned](#) (int64_t value)
Set the default value for a numericSigned parameter.
- [IrisParameterBuilder](#) & [setDescr](#) (const std::string &description)
*Set the *description* field.*
- [IrisParameterBuilder](#) & [setFormat](#) (const std::string &format)
*Set the *format* field.*
- [IrisParameterBuilder](#) & [setHidden](#) (bool hidden)
Set the resource to hidden !
- [IrisParameterBuilder](#) & [setInitOnly](#) (bool value=true)
*Set the *initOnly* field.*
- [IrisParameterBuilder](#) & [setMax](#) (const std::vector< uint64_t > &max)
*Set the *max* field.*
- [IrisParameterBuilder](#) & [setMax](#) (uint64_t max)
*Set the *max* field.*
- [IrisParameterBuilder](#) & [setMaxFloat](#) (double max)
*Set the *max* field for floating-point parameters.*
- [IrisParameterBuilder](#) & [setMaxSigned](#) (const std::vector< uint64_t > &max)
*Set the *max* field.*
- [IrisParameterBuilder](#) & [setMaxSigned](#) (int64_t max)
*Set the *max* field.*
- [IrisParameterBuilder](#) & [setMin](#) (const std::vector< uint64_t > &min)
*Set the *min* field.*
- [IrisParameterBuilder](#) & [setMin](#) (uint64_t min)
*Set the *min* field.*
- [IrisParameterBuilder](#) & [setMinFloat](#) (double min)
*Set the *min* field for floating-point parameters.*
- [IrisParameterBuilder](#) & [setMinSigned](#) (const std::vector< uint64_t > &min)
*Set the *min* field.*
- [IrisParameterBuilder](#) & [setMinSigned](#) (int64_t min)
*Set the *min* field.*

- [IrisParameterBuilder](#) & [setName](#) (const std::string &name)
Set the `name` field.
- [IrisParameterBuilder](#) & [setRange](#) (const std::vector< uint64_t > &min, const std::vector< uint64_t > &max)
Set both the `min` field and the `max` field.
- [IrisParameterBuilder](#) & [setRange](#) (uint64_t min, uint64_t max)
Set both the `min` field and the `max` field.
- [IrisParameterBuilder](#) & [setRangeFloat](#) (double min, double max)
Set both the `min` field and the `max` field.
- [IrisParameterBuilder](#) & [setRangeSigned](#) (const std::vector< uint64_t > &min, const std::vector< uint64_t > &max)
Set both the `min` field and the `max` field.
- [IrisParameterBuilder](#) & [setRangeSigned](#) (int64_t min, int64_t max)
Set both the `min` field and the `max` field.
- [IrisParameterBuilder](#) & [setRwMode](#) (const std::string &rwMode)
Set the `rwMode` field.
- [IrisParameterBuilder](#) & [setSubRscId](#) (uint64_t subRscId)
Set the `subRscId` field.
- [IrisParameterBuilder](#) & [setTag](#) (const std::string &tag)
Set a boolean tag for this parameter resource.
- [IrisParameterBuilder](#) & [setTag](#) (const std::string &tag, const IrisValue &value)
Set a tag for this parameter resource.
- [IrisParameterBuilder](#) & [setTopology](#) (bool value=true)
Set the `topology` field.
- [IrisParameterBuilder](#) & [setType](#) (const std::string &type)
Set the type of this parameter.

8.35.1 Detailed Description

Helper class to construct instantiation parameters.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 IrisParameterBuilder()

```
iris::IrisParameterBuilder::IrisParameterBuilder (
    ResourceInfo & info_ ) [inline]
```

Construct a parameter builder for a given parameter resource.

Parameters

<i>info</i> ↔	The resource info object for the parameter being built.
—	

8.35.3 Member Function Documentation

8.35.3.1 addEnum()

```
IrisParameterBuilder & iris::IrisParameterBuilder::addEnum (
    const std::string & symbol,
```

```
const IrisValue & value,
const std::string & description = std::string() ) [inline]
```

Add an enum symbol for this parameter.

Parameters

<i>symbol</i>	The enum symbol that is being added.
<i>value</i>	The value associated with the symbol.
<i>description</i>	A description explaining the meaning of the symbol.

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.2 addStringEnum()

```
IrisParameterBuilder & iris::IrisParameterBuilder::addStringEnum (
    const std::string & value,
    const std::string & description = std::string() ) [inline]
```

Add a string enum symbol for this parameter.

For string enums, the symbol and value are the same.

Parameters

<i>value</i>	The value associated with the symbol.
<i>description</i>	A description explaining the meaning of the symbol.

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.3 setBitWidth()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setBitWidth (
    uint64_t bitWidth ) [inline]
```

Set the `bitWidth` field.

Parameters

<i>bitWidth</i>	The <code>bitWidth</code> field of the <code>ResourceInfo</code> object.
-----------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.4 setDefault() [1/3]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setDefault (
    const std::string & value ) [inline]
```

Set the default value for a string parameter.

Parameters

<i>value</i>	The defaultString field of the ParameterInfo object.
--------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.5 setDefault() [2/3]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setDefault (
    const std::vector< uint64_t > & value ) [inline]
```

Set the default value for a numeric parameter.

Use this variant for values that are $\geq 2 \times 64$.

Parameters

<i>value</i>	The defaultData field of the ParameterInfo object.
--------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.6 setDefault() [3/3]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setDefault (
    uint64_t value ) [inline]
```

Set the default value for a numeric parameter.

Parameters

<i>value</i>	The defaultData field of the ParameterInfo object.
--------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.7 setDefaultFloat()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setDefaultFloat (
    double value ) [inline]
```

Set the default value for a numericFp parameter.

Parameters

<i>value</i>	The defaultData field of the ParameterInfo object.
--------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.8 setDefaultSigned() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setDefaultSigned (
    const std::vector< uint64_t > & value ) [inline]
```

Set the default value for a numericSigned parameter.

Use this variant for values that are out of range for int64_t.

Parameters

<i>value</i>	The defaultData field of the ParameterInfo object.
--------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.9 setDefaultSigned() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setDefaultSigned (
    int64_t value ) [inline]
```

Set the default value for a numericSigned parameter.

Parameters

<i>value</i>	The defaultData field of the ParameterInfo object.
--------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.10 setDescr()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setDescr (
    const std::string & description ) [inline]
```

Set the description field.

Parameters

<i>description</i>	The description field of the ResourceInfo object.
--------------------	---

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.11 setFormat()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setFormat (
    const std::string & format ) [inline]
```

Set the format field.

Parameters

<i>format</i>	The format field of the ResourceInfo object.
---------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.12 setHidden()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setHidden (
    bool hidden ) [inline]
```

Set the resource to hidden !

Parameters

<i>hidden</i>	If true, this event source is not listed in resource_getList() calls but can still be accessed by resource_getResourceInfo() for clients that know the resource name. !
---------------	---

Returns

A reference to this TYPE object allowing calls to be chained together.

8.35.3.13 setInitOnly()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setInitOnly (
    bool value = true ) [inline]
```

Set the initOnly field.

Parameters

<i>value</i>	The initOnly field of the ParameterInfo object.
--------------	---

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.14 setMax() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMax (
    const std::vector< uint64_t > & max ) [inline]
```

Set the max field.

Use this variant to set values that are $\geq 2 \times 64$.

Parameters

<i>max</i>	The max field of the ParameterInfo object.
------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.15 setMax() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMax (
    uint64_t max ) [inline]
```

Set the `max` field.

Parameters

<i>max</i>	The <code>max</code> field of the <code>ParameterInfo</code> object.
------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.16 setMaxFloat()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMaxFloat (
    double max ) [inline]
```

Set the `max` field for floating-point parameters.

This implies that the parameter type is "numericFp".

Parameters

<i>max</i>	The <code>max</code> field of the <code>ParameterInfo</code> object.
------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.17 setMaxSigned() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMaxSigned (
    const std::vector< uint64_t > & max ) [inline]
```

Set the `max` field.

This implies that the parameter type is "numericSigned". Use this variant for signed values that are out of range for `int64_t`.

Parameters

<i>max</i>	The <code>max</code> field of the <code>ParameterInfo</code> object.
------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.18 setMaxSigned() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMaxSigned (
    int64_t max ) [inline]
```

Set the `max` field.

This implies that the parameter type is "numericSigned".

Parameters

<i>max</i>	The <code>max</code> field of the <code>ParameterInfo</code> object.
------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.19 setMin() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMin (
    const std::vector< uint64_t > & min ) [inline]
```

Set the `min` field.

Use this variant to set values that are $\geq 2 \times 64$.

Parameters

<i>min</i>	The <code>min</code> field of the <code>ParameterInfo</code> object.
------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.20 setMin() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMin (
    uint64_t min ) [inline]
```

Set the `min` field.

Parameters

<i>min</i>	The <code>min</code> field of the <code>ParameterInfo</code> object.
------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.21 setMinFloat()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMinFloat (
    double min ) [inline]
```

Set the `min` field for floating-point parameters.

This implies that the parameter type is "numericFp".

Parameters

<i>min</i>	The <code>min</code> field of the <code>ParameterInfo</code> object.
------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.22 setMinSigned() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMinSigned (
    const std::vector< uint64_t > & min ) [inline]
```

Set the `min` field.

This implies that the parameter type is "numericSigned". Use this variant for signed values that are out of range for `int64_t`.

Parameters

<i>min</i>	The <code>min</code> field of the <code>ParameterInfo</code> object.
------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.23 setMinSigned() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMinSigned (
    int64_t min ) [inline]
```

Set the `min` field.

This implies that the parameter type is "numericSigned".

Parameters

<i>min</i>	The <code>min</code> field of the <code>ParameterInfo</code> object.
------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.24 setName()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setName (
    const std::string & name ) [inline]
```

Set the `name` field.

Parameters

<i>name</i>	The <code>name</code> field of the <code>ResourceInfo</code> object.
-------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.25 setRange() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRange (
    const std::vector< uint64_t > & min,
    const std::vector< uint64_t > & max ) [inline]
```

Set both the `min` field and the `max` field.

Use this variant to set values that are $\geq 2 \times 64$.

Parameters

<i>min</i>	The <code>min</code> field of the <code>ParameterInfo</code> object.
<i>max</i>	The <code>max</code> field of the <code>ParameterInfo</code> object.

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.26 setRange() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRange (
    uint64_t min,
    uint64_t max ) [inline]
```

Set both the `min` field and the `max` field.

Parameters

<i>min</i>	The <code>min</code> field of the <code>ParameterInfo</code> object.
<i>max</i>	The <code>max</code> field of the <code>ParameterInfo</code> object.

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.27 setRangeFloat()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRangeFloat (
    double min,
    double max ) [inline]
```

Set both the `min` field and the `max` field.

This implies that the parameter type is "numericFp".

Parameters

<i>min</i>	The <code>min</code> field of the <code>ParameterInfo</code> object.
<i>max</i>	The <code>max</code> field of the <code>ParameterInfo</code> object.

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.28 setRangeSigned() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRangeSigned (
    const std::vector< uint64_t > & min,
    const std::vector< uint64_t > & max ) [inline]
```

Set both the `min` field and the `max` field.

This implies that the parameter type is "numericSigned". Use this variant for signed values that are out of range for `int64_t`.

Parameters

<i>min</i>	The <code>min</code> field of the <code>ParameterInfo</code> object.
<i>max</i>	The <code>max</code> field of the <code>ParameterInfo</code> object.

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.29 setRangeSigned() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRangeSigned (
    int64_t min,
    int64_t max ) [inline]
```

Set both the `min` field and the `max` field.

This implies that the parameter type is "numericSigned".

Parameters

<i>min</i>	The <code>min</code> field of the <code>ParameterInfo</code> object.
<i>max</i>	The <code>max</code> field of the <code>ParameterInfo</code> object.

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.30 setRwMode()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRwMode (
    const std::string & rwMode ) [inline]
```

Set the `rwMode` field.

Parameters

<i>rwMode</i>	The <code>rwMode</code> field of the <code>ResourceInfo</code> object.
---------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.31 setSubRscId()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setSubRscId (
    uint64_t subRscId ) [inline]
```

Set the `subRscId` field.

Parameters

<i>sub↔ RscId</i>	The <code>subRscId</code> field of the <code>ResourceInfo</code> object.
-----------------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.32 setTag() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setTag (
```

```
const std::string & tag ) [inline]
```

Set a boolean tag for this parameter resource.

Parameters

<i>tag</i>	The name of the tag to set.
------------	-----------------------------

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.33 setTag() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setTag (
    const std::string & tag,
    const IrisValue & value ) [inline]
```

Set a tag for this parameter resource.

Parameters

<i>tag</i>	The name of the tag to set.
<i>value</i>	The value to set for this tag.

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.34 setTopology()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setTopology (
    bool value = true ) [inline]
```

Set the topology field.

Parameters

<i>value</i>	The topology field of the ParameterInfo object.
--------------	---

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

8.35.3.35 setType()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setType (
    const std::string & type ) [inline]
```

Set the type of this parameter.

The bitWidth field must be set before setting the type.

Parameters

<i>type</i>	The type field of the ResourceInfo object.
-------------	--

Returns

A reference to this [IrisParameterBuilder](#) object allowing calls to be chained together.

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

- [IrisParameterBuilder.h](#)

8.36 iris::IrisPluginFactory< PLUGIN_CLASS > Class Template Reference

Public Member Functions

- **IrisPluginFactory** (IrisC_Funcions *iris_c_functions, const std::string &plugin_name)
- **IrisErrorCode unregisterInstance** ()

Static Public Member Functions

- static int64_t **initPlugin** (IrisC_Funcions *functions, const std::string &plugin_name)

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

- [IrisPluginFactory.h](#)

8.37 iris::IrisPluginFactoryBuilder Class Reference

Set meta data for instantiating a plug-in instance.

```
#include <IrisPluginFactory.h>
```

Inherits [iris::IrisInstanceFactoryBuilder](#).

Public Member Functions

- const std::string & [getDefaultInstanceName](#) () const
Get the default name to use for plug-in instances.
- const std::string & [getInstanceNamePrefix](#) () const
Get the prefix to use for instances of this plug-in.
- const std::string & [getPluginName](#) () const
Get the plug-in name.
- [IrisPluginFactoryBuilder](#) (const std::string &name)
- void [setDefaultInstanceName](#) (const std::string &name)
Override the default instance name for plug-in instances.
- void [setInstanceNamePrefix](#) (const std::string &prefix)
Override the instance name prefix. The default is "client.plugin".
- void [setPluginName](#) (const std::string &name)
Override the plug-in name.

8.37.1 Detailed Description

Set meta data for instantiating a plug-in instance.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 IrisPluginFactoryBuilder()

```
iris::IrisPluginFactoryBuilder::IrisPluginFactoryBuilder (
    const std::string & name ) [inline]
```

Parameters

<i>name</i>	The name of the plug-in to build.
-------------	-----------------------------------

8.37.3 Member Function Documentation

8.37.3.1 getDefaultInstanceName()

```
const std::string & iris::IrisPluginFactoryBuilder::getDefaultInstanceName ( ) const [inline]
```

Get the default name to use for plug-in instances.

Returns

The default name for plug-in instances.

8.37.3.2 getInstanceNamePrefix()

```
const std::string & iris::IrisPluginFactoryBuilder::getInstanceNamePrefix ( ) const [inline]
```

Get the prefix to use for instances of this plug-in.

Returns

The prefix to use for instances of this plug-in.

8.37.3.3 getPluginName()

```
const std::string & iris::IrisPluginFactoryBuilder::getPluginName ( ) const [inline]
```

Get the plug-in name.

Returns

The name of the plug-in.

8.37.3.4 setDefaultInstanceName()

```
void iris::IrisPluginFactoryBuilder::setDefaultInstanceName (
    const std::string & name ) [inline]
```

Override the default instance name for plug-in instances.

The factory provides a sensible default for this name so it should only be overridden if there is a good reason to do so.

Parameters

<i>name</i>	The default name for plug-in instances.
-------------	---

8.37.3.5 setInstanceNamePrefix()

```
void iris::IrisPluginFactoryBuilder::setInstanceNamePrefix (
    const std::string & prefix ) [inline]
```

Override the instance name prefix. The default is "client.plugin".

The factory provides a sensible default for this prefix so it should only be overridden if there is a good reason to do so.

Parameters

<i>prefix</i>	The prefix that will be used for instances of this plug-in.
---------------	---

8.37.3.6 setPluginName()

```
void iris::IrisPluginFactoryBuilder::setPluginName (
    const std::string & name ) [inline]
```

Override the plug-in name.

The factory provides a sensible default for this name so it should only be overridden if there is a good reason to do so.

Parameters

<i>name</i>	The name of the plug-in.
-------------	--------------------------

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

- [IrisPluginFactory.h](#)

8.38 iris::IrisRegisterReadEventEmitter< REG_T, ARGS > Class Template Reference

An EventEmitter class for register read events.

```
#include <IrisRegisterEventEmitter.h>
```

Inherits IrisRegisterEventEmitterBase.

Public Member Functions

- void [operator\(\)](#) (ResourceId rscId, bool debug, REG_T value, ARGS... args)
Emit an event.

8.38.1 Detailed Description

```
template<typename REG_T, typename... ARGS>
```

```
class iris::IrisRegisterReadEventEmitter< REG_T, ARGS >
```

An EventEmitter class for register read events.

Template Parameters

<i>REG_T</i>	The type of the register being read.
<i>ARGS</i>	The types of any custom fields that this event source defines, in addition to the standard fields defined for register read events.

Use [IrisRegisterReadEventEmitter](#) with [IrisInstanceBuilder](#) to add register read events to your Iris instance:

```
// Declare an event emitter
iris::IrisRegisterReadEventEmitter<uint64_t> reg_read_event;
// Add it to an Iris instance
iris::IrisInstance my_instance(...);
iris::IrisInstanceBuilder *builder = my_instance->getBuilder();
builder->setRegisterReadEvent("READ_REG", reg_read_event);
// Add some registers that will be traced by this event
```

```

builder->setNextRscId(0x1000);
builder->addRegister("X0", 64, "Register X0");
builder->addRegister("X1", 64, "Register X1");
builder->addRegister("X2", 64, "Register X2");
builder->addRegister("X3", 64, "Register X3");
// Now that the Instance builder has the metadata for the registers, we need
// to finalize the register read event to populate the event metadata.
builder->finalizeRegisterReadEvent();
uint64_t readRegister(unsigned reg_index, bool is_debug)
{
    uint64_t value = readRegValue(reg_index);
    // Emit an event
    reg_read_event(0x1000 | reg_index, is_debug, value);
    return value;
}

```

8.38.2 Member Function Documentation

8.38.2.1 operator()

```

template<typename REG_T , typename... ARGS>
void iris::IrisRegisterReadEventEmitter< REG_T, ARGS >::operator() (
    ResourceId rscId,
    bool debug,
    REG_T value,
    ARGS... args ) [inline]

```

Emit an event.

Parameters

<i>rscId</i>	Resource id for the register that was accessed.
<i>debug</i>	True if this access originated from a debug access.
<i>value</i>	The register value that was read during this event.
<i>args</i>	Any additional custom fields for this event.

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

- [IrisRegisterEventEmitter.h](#)

8.39 iris::IrisRegisterUpdateEventEmitter< REG_T, ARGS > Class Template Reference

An EventEmitter class for register update events.

```
#include <IrisRegisterEventEmitter.h>
```

Inherits IrisRegisterEventEmitterBase.

Public Member Functions

- void [operator\(\)](#) (ResourceId rscId, bool debug, REG_T old_value, REG_T new_value, ARGS... args)
Emit an event.

8.39.1 Detailed Description

```
template<typename REG_T, typename... ARGS>
```

```
class iris::IrisRegisterUpdateEventEmitter< REG_T, ARGS >
```

An EventEmitter class for register update events.

Template Parameters

<i>REG_T</i>	The type of the register being read.
<i>ARGS</i>	Types of any custom fields that this event source defines, in addition to the standard fields defined for register update events.

Use [IrisRegisterUpdateEventEmitter](#) with [IrisInstanceBuilder](#) to add register update events to your Iris instance:

```
// Declare an event emitter
iris::IrisRegisterUpdateEventEmitter<uint64_t> reg_update_event;
// Add it to an Iris instance
iris::IrisInstance my_instance(...);
iris::IrisInstanceBuilder *builder = my_instance->getBuilder();
builder->setRegisterUpdateEvent("WRITE_REG", reg_update_event);
// Add some registers that will be traced by this event
builder->setNextRscId(0x1000);
builder->addRegister("X0", 64, "Register X0");
builder->addRegister("X1", 64, "Register X1");
builder->addRegister("X2", 64, "Register X2");
builder->addRegister("X3", 64, "Register X3");
// Now that the Instance builder has the metadata for the registers, we need
// to finalize the register update event to populate the event metadata.
builder->finalizeRegisterUpdateEvent();
void writeRegister(unsigned reg_index, bool is_debug, uint64_t new_value)
{
    uint64_t old_value = readRegValue(reg_index);
    writeRegValue(reg_index, new_value);
    // Emit an event
    reg_update_event(0x1000 | reg_index, is_debug, old_value, new_value);
}
```

8.39.2 Member Function Documentation

8.39.2.1 operator()()

```
template<typename REG_T , typename... ARGS>
void iris::IrisRegisterUpdateEventEmitter< REG_T, ARGS >::operator() (
    ResourceId rscId,
    bool debug,
    REG_T old_value,
    REG_T new_value,
    ARGS... args ) [inline]
```

Emit an event.

Parameters

<i>rscId</i>	Resource id for the register that was accessed.
<i>debug</i>	True if this access originated from a debug access.
<i>old_value</i>	The register value before the event.
<i>new_value</i>	The register value after the event.
<i>args</i>	Any additional custom fields for this event.

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

- [IrisRegisterEventEmitter.h](#)

8.40 iris::IrisSimulationResetContext Class Reference

Provides context to a reset delegate call.

```
#include <IrisInstanceSimulation.h>
```

Public Member Functions

- bool [getAllowPartialReset](#) () const
Get the allowPartialReset flag.
- void [setAllowPartialReset](#) (bool value=true)

8.40.1 Detailed Description

Provides context to a reset delegate call.

8.40.2 Member Function Documentation

8.40.2.1 getAllowPartialReset()

```
bool iris::IrisSimulationResetContext::getAllowPartialReset ( ) const [inline]
```

Get the allowPartialReset flag.

Returns

Returns true if simulation_reset() was called with allowPartialReset=true.

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

- [IrisInstanceSimulation.h](#)

8.41 iris::IrisInstanceBuilder::MemorySpaceBuilder Class Reference

Used to set metadata for a memory space.

```
#include <IrisInstanceBuilder.h>
```

Public Member Functions

- [MemorySpaceBuilder](#) & [addAttribute](#) (const std::string &name, AttributeInfo attrib)
Add an attribute to the attrib field.
- MemorySpaceId [getSpaceId](#) () const
Get the memory space id for this memory space.
- [MemorySpaceBuilder](#) ([IrisInstanceMemory::SpaceInfoAndAccess](#) &info_)
- [MemorySpaceBuilder](#) & [setAttributeDefault](#) (const std::string &name, IrisValue value)
Set the default value for an attribute in the attrib field.
- [MemorySpaceBuilder](#) & [setAttributes](#) (const AttributeInfoMap &attribInfoMap)
Add attributes to the attrib field.
- [MemorySpaceBuilder](#) & [setCanonicalMsn](#) (uint64_t canonicalMsn)
Set the canonicalMsn field.
- [MemorySpaceBuilder](#) & [setDescription](#) (const std::string &description)
Set the description field.
- [MemorySpaceBuilder](#) & [setEndianness](#) (const std::string &endianness)
Set the endianness field.
- [MemorySpaceBuilder](#) & [setMaxAddr](#) (uint64_t maxAddr)
Set the maxAddr field.
- [MemorySpaceBuilder](#) & [setMinAddr](#) (uint64_t minAddr)
Set the minAddr field.
- [MemorySpaceBuilder](#) & [setName](#) (const std::string &name)
Set the name field.

- `template<IrisErrorCode(*)>(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) FUNC>`
[MemorySpaceBuilder](#) & [setReadDelegate](#) ()
Set the delegate to read this memory space.
- [MemorySpaceBuilder](#) & [setReadDelegate](#) ([MemoryReadDelegate](#) delegate)
Set the delegate to read this memory space.
- `template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) METHOD>`
[MemorySpaceBuilder](#) & [setReadDelegate](#) (T *instance)
Set the delegate to read this memory space.
- `template<IrisErrorCode(*)>(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) FUNC>`
[MemorySpaceBuilder](#) & [setSidebandDelegate](#) ()
Set the delegate to read sideband information.
- [MemorySpaceBuilder](#) & [setSidebandDelegate](#) ([MemoryGetSidebandInfoDelegate](#) delegate)
Set the delegate to read sideband information.
- `template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) METHOD>`
[MemorySpaceBuilder](#) & [setSidebandDelegate](#) (T *instance)
Set the delegate to read sideband information.
- [MemorySpaceBuilder](#) & [setSupportedByteWidths](#) (uint64_t supportedByteWidths)
Set the `supportedByteWidths` field.
- `template<IrisErrorCode(*)>(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) FUNC>`
[MemorySpaceBuilder](#) & [setWriteDelegate](#) ()
Set the delegate to write to this memory space.
- [MemorySpaceBuilder](#) & [setWriteDelegate](#) ([MemoryWriteDelegate](#) delegate)
Set the delegate to write to this memory space.
- `template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>`
[MemorySpaceBuilder](#) & [setWriteDelegate](#) (T *instance)
Set the delegate to write to this memory space.

8.41.1 Detailed Description

Used to set metadata for a memory space.

8.41.2 Member Function Documentation

8.41.2.1 addAttribute()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::addAttribute (
    const std::string & name,
    AttributeInfo attrib ) [inline]
```

Add an attribute to the `attrib` field.

Parameters

<i>name</i>	The name of this attribute.
<i>attrib</i>	AttributeInfo for this attribute.

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.2 getSpaceId()

```
MemorySpaceId iris::IrisInstanceBuilder::MemorySpaceBuilder::getSpaceId ( ) const [inline]
```

Get the memory space id for this memory space.

This can be useful for setting up address translations and to map access requests to the correct memory space in memory access delegates.

Returns

The memory space id for this memory space.

8.41.2.3 setAttributeDefault()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setAttributeDefault (
    const std::string & name,
    IrisValue value ) [inline]
```

Set the default value for an attribute in the `attrib` field.

Parameters

<i>name</i>	The name of this attribute.
<i>value</i>	Default value of the named attribute.

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.4 setAttributes()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setAttributes (
    const AttributeInfoMap & attribInfoMap ) [inline]
```

Add attributes to the `attrib` field.

Parameters

<i>attribInfoMap</i>	The attributes of this memory space.
----------------------	--------------------------------------

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.5 setCanonicalMsn()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setCanonicalMsn (
    uint64_t canonicalMsn ) [inline]
```

Set the `canonicalMsn` field.

Parameters

<i>canonicalMsn</i>	The canonicalMsn field of the MemorySpaceInfo object.
---------------------	---

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.6 setDescription()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setDescription (
    const std::string & description ) [inline]
```

Set the description field.

Parameters

<i>description</i>	The description field of the MemorySpaceInfo object.
--------------------	--

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.7 setEndianness()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setEndianness (
    const std::string & endianness ) [inline]
```

Set the endianness field.

Parameters

<i>endianness</i>	The endianness field of the MemorySpaceInfo object.
-------------------	---

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.8 setMaxAddr()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setMaxAddr (
    uint64_t maxAddr ) [inline]
```

Set the maxAddr field.

Parameters

<i>maxAddr</i>	The maxAddr field of the MemorySpaceInfo object.
----------------	--

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.9 setMinAddr()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setMinAddr (
    uint64_t minAddr ) [inline]
```

Set the minAddr field.

Parameters

<i>minAddr</i>	The minAddr field of the MemorySpaceInfo object.
----------------	--

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.10 setName()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setName (
    const std::string & name ) [inline]
```

Set the name field.

Parameters

<i>name</i>	The name field of the MemorySpaceInfo object.
-------------	---

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.11 setReadDelegate() [1/3]

```
template<IrisErrorCode(*) (const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const Attribute↔
ValueMap &, MemoryReadResult &) FUNC>
```

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setReadDelegate ( ) [inline]
```

Set the delegate to read this memory space.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultMemoryReadDelegate](#)

Template Parameters

<i>FUNC</i>	A memory read delegate function.
-------------	----------------------------------

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.12 setReadDelegate() [2/3]

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setReadDelegate (
    MemoryReadDelegate delegate ) [inline]
```

Set the delegate to read this memory space.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultMemoryReadDelegate](#)

Parameters

<i>delegate</i>	MemoryReadDelegate object.
-----------------	----------------------------

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.13 setReadDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) METHOD>
```

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setReadDelegate (
    T * instance ) [inline]
```

Set the delegate to read this memory space.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultMemoryReadDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a memory read delegate.
<i>METHOD</i>	A memory read delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.14 setSidebandDelegate() [1/3]

```
template<IrisErrorCode(*) (const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) FUNC>
```

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setSidebandDelegate ( )
[inline]
```

Set the delegate to read sideband information.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate](#)

Template Parameters

<i>FUNC</i>	A memory sideband information delegate function.
-------------	--

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.15 setSidebandDelegate() [2/3]

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setSidebandDelegate (
    MemoryGetSidebandInfoDelegate delegate ) [inline]
```

Set the delegate to read sideband information.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate](#)

Parameters

<i>delegate</i>	MemoryGetSidebandInfoDelegate object.
-----------------	---------------------------------------

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.16 setSidebandDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*) (const MemorySpaceInfo &, uint64_t, const IrisValue↵
Map &, const std::vector< std::string > &, IrisValueMap &) METHOD>
```

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setSidebandDelegate (
    T * instance ) [inline]
```

Set the delegate to read sideband information.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a memory sideband information delegate.
<i>METHOD</i>	A memory sideband information delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.17 setSupportedByteWidths()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setSupportedByteWidths (
    uint64_t supportedByteWidths ) [inline]
```

Set the `supportedByteWidths` field.

Usage:

`setSupportedByteWidths(1+2+4+8+16);` // Indicate support for byteWidth 1, 2, 4, 8, and 16.

Parameters

<i>supportedByteWidths</i>	Outer envelope of all supported byteWidth values Bit mask: Bit N==1 means byteWidth 1 << N is supported.
----------------------------	--

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.18 setWriteDelegate() [1/3]

```
template<IrisErrorCode(*) (const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeMap &, const uint64_t *, MemoryWriteResult &) FUNC>
```

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setWriteDelegate ( ) [inline]
```

Set the delegate to write to this memory space.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultMemoryWriteDelegate](#)

Template Parameters

<i>FUNC</i>	A memory write delegate function.
-------------	-----------------------------------

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.19 setWriteDelegate() [2/3]

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setWriteDelegate (
    MemoryWriteDelegate delegate ) [inline]
```

Set the delegate to write to this memory space.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultMemoryWriteDelegate](#)

Parameters

<i>delegate</i>	MemoryWriteDelegate object.
-----------------	-----------------------------

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

8.41.2.20 setWriteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setWriteDelegate (
    T * instance ) [inline]
```

Set the delegate to write to this memory space.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultMemoryWriteDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a memory write delegate.
<i>METHOD</i>	A memory write delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [MemorySpaceBuilder](#) object allowing calls to be chained together.

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

- [IrisInstanceBuilder.h](#)

8.42 iris::IrisCommandLineParser::Option Struct Reference

[Option](#) container.

```
#include <IrisCommandLineParser.h>
```

Public Member Functions

- [Option](#) & [setList](#) (char sep=',')

Friends

- class [IrisCommandLineParser](#)

8.42.1 Detailed Description

[Option](#) container.

8.42.2 Member Function Documentation

8.42.2.1 setList()

```
Option & iris::IrisCommandLineParser::Option::setList (
    char sep = ',' ) [inline]
```

Make this option a "list" option which can be specified multiple times. The value is stored as a single string and the elements are separated by "sep". Use [getList\(\)](#) or [getMap\(\)](#) to extract the elements.

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

- [IrisCommandLineParser.h](#)

8.43 iris::IrisInstanceBuilder::ParameterBuilder Class Reference

Used to set metadata on a parameter.

```
#include <IrisInstanceBuilder.h>
```

Public Member Functions

- [ParameterBuilder](#) & [addEnum](#) (const std::string &symbol, const IrisValue &value, const std::string &description=std::string())
Add a symbol to the enums field for numeric resources.
- [ParameterBuilder](#) & [addStringEnum](#) (const std::string &stringValue, const std::string &description=std::string())
Add a symbol to the enums field for string resources.
- ResourceId [getRsclId](#) () const
Return the rsclId that was allocated for this resource.
- [ParameterBuilder](#) & [getRsclId](#) (ResourceId &rsclIdOut)
Get the rsclId that was allocated for this resource.
- [ParameterBuilder](#) ([IrisInstanceResource::ResourceInfoAndAccess](#) &info_)
- [ParameterBuilder](#) & [setBitWidth](#) (uint64_t bitWidth)
Set the bitWidth field.
- [ParameterBuilder](#) & [setName](#) (const std::string &cname)
Set the cname field.
- template<typename T >
[ParameterBuilder](#) & [setDefaultData](#) (std::initializer_list< T > &&t)
Set the default value for wide numeric parameters.
- [ParameterBuilder](#) & [setDefaultData](#) (uint64_t value)
Set the default value for numeric parameter to a value <= 64 bit.
- template<typename Container >
[ParameterBuilder](#) & [setDefaultDataFromContainer](#) (const Container &container)
Set the default value for wide numeric parameters.
- [ParameterBuilder](#) & [setDefaultString](#) (const std::string &defaultString)
Set the defaultData field for wide numeric parameters (bitWidth > 64 bit).
- [ParameterBuilder](#) & [setDescription](#) (const std::string &description)
Obsolete alias for setDescription(). Do not use.
- [ParameterBuilder](#) & [setDescription](#) (const std::string &description)
Set the description field.
- [ParameterBuilder](#) & [setFormat](#) (const std::string &format)
Set the format field.
- [ParameterBuilder](#) & [setHidden](#) (bool hidden=true)
Set the resource to hidden.
- [ParameterBuilder](#) & [setInitOnly](#) (bool initOnly=true)
Set the initOnly flag of a parameter.
- template<typename T >
[ParameterBuilder](#) & [setMax](#) (std::initializer_list< T > &&t)

- Set the `max` field for wide numeric parameters.*

 - [ParameterBuilder & setMax](#) (uint64_t value)

Set the `max` field to a value ≤ 64 bit.
- template<typename Container >

 - [ParameterBuilder & setMaxFromContainer](#) (const Container &container)

Set the `max` field for wide numeric parameters.
- template<typename T >

 - [ParameterBuilder & setMin](#) (std::initializer_list< T > &&t)

Set the `min` field for wide numeric parameters.
- [ParameterBuilder & setMin](#) (uint64_t value)

Set the `min` field to a value ≤ 64 bit.
- template<typename Container >

 - [ParameterBuilder & setMinFromContainer](#) (const Container &container)

Set the `min` field for wide numeric parameters.
- [ParameterBuilder & setName](#) (const std::string &name)

Set the `name` field.
- [ParameterBuilder & setParentRscId](#) (ResourceId parentRscId)

Set the `parentRscId` field.
- template<IrisErrorCode(*)>(const ResourceInfo &, ResourceReadResult &) FUNC>

 - [ParameterBuilder & setReadDelegate](#) ()

Set the delegate to read the resource.
- [ParameterBuilder & setReadDelegate](#) (ResourceReadDelegate readDelegate)

Set the delegate to read the resource.
- template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>

 - [ParameterBuilder & setReadDelegate](#) (T *instance)

Set the delegate to read the resource.
- [ParameterBuilder & setRwMode](#) (const std::string &rwMode)

Set the `rwMode` field.
- [ParameterBuilder & setSubRscId](#) (uint64_t subRscId)

Set the `subRscId` field.
- [ParameterBuilder & setTag](#) (const std::string &tag)

Set the named boolean tag to true (e.g. `isPc`)
- [ParameterBuilder & setTag](#) (const std::string &tag, const IrisValue &value)

Set a tag to the specified value.
- [ParameterBuilder & setType](#) (const std::string &type)

Set the `type` field.
- template<IrisErrorCode(*)>(const ResourceInfo &, const ResourceWriteValue &) FUNC>

 - [ParameterBuilder & setWriteDelegate](#) ()

Set the delegate to write the resource.
- [ParameterBuilder & setWriteDelegate](#) (ResourceWriteDelegate writeDelegate)

Set the delegate to write the resource.
- template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &) METHOD>

 - [ParameterBuilder & setWriteDelegate](#) (T *instance)

Set the delegate to write the resource.

8.43.1 Detailed Description

Used to set metadata on a parameter.

8.43.2 Member Function Documentation

8.43.2.1 addEnum()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::addEnum (
    const std::string & symbol,
    const IrisValue & value,
    const std::string & description = std::string() ) [inline]
```

Add a symbol to the enums field for numeric resources.

This should be called multiple times to add multiple symbols.

Parameters

<i>symbol</i>	The symbol string to be associated with the specified value.
<i>value</i>	The value of this symbol.
<i>description</i>	A description of this symbol.

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.2 addStringEnum()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::addStringEnum (
    const std::string & stringValue,
    const std::string & description = std::string() ) [inline]
```

Add a symbol to the enums field for string resources.

This should be called multiple times to add multiple symbols.

Parameters

<i>value</i>	The string value of this symbol. This is also used as the symbols string.
<i>description</i>	A description of this symbol.

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.3 getRscId() [1/2]

```
ResourceId iris::IrisInstanceBuilder::ParameterBuilder::getRscId ( ) const [inline]
```

Return the rscId that was allocated for this resource.

Returns

The rscId that was allocated for this resource.

8.43.2.4 getRscId() [2/2]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::getRscId (
    ResourceId & rscIdOut ) [inline]
```

Get the rscId that was allocated for this resource.

This variant is useful to get the ResourceId of fields added in a chained call where return values are not practical.

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.5 setBitWidth()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setBitWidth (
    uint64_t bitWidth ) [inline]
```

Set the `bitWidth` field.

Parameters

<i>bitWidth</i>	The <code>bitWidth</code> field of the <code>ResourceInfo</code> object.
-----------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.6 setCname()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setCname (
    const std::string & cname ) [inline]
```

Set the `cname` field.

Parameters

<i>cname</i>	The <code>cname</code> field of the <code>ResourceInfo</code> object.
--------------	---

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.7 setDefaultData() [1/2]

```
template<typename T >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setDefaultData (
    std::initializer_list< T > && t ) [inline]
```

Set the default value for wide numeric parameters.

This function accepts a braced initializer-list and is otherwise identical to [setDefaultDataFromContainer\(\)](#).

Each element will be promoted/narrowed to `uint64_t`.

Parameters

<i>t</i>	Braced initializer-list.
----------	--------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.8 setDefaultData() [2/2]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setDefaultData (
```

```
uint64_t value ) [inline]
```

Set the default value for numeric parameter to a value ≤ 64 bit.

If the parameter is wider than the passed value the value is zero extended.

If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>value</i>	The defaultData field of the ParameterInfo object.
--------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.9 setDefaultDataFromContainer()

```
template<typename Container >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setDefaultDataFromContainer (
    const Container & container ) [inline]
```

Set the default value for wide numeric parameters.

Container must be a type which allows to iterate over uint64_t bit chunks of the value, least significant bits first, for example `std::array<uint64_t>` or `std::vector<uint64_t>`.

Each element of the container will be promoted/narrowed to uint64_t.

If the parameter is wider than the passed value the value is zero extended.

If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>container</i>	Container containing the value in 64-bit chunks.
------------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.10 setDefaultString()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setDefaultString (
    const std::string & defaultString ) [inline]
```

Set the defaultData field for wide numeric parameters (bitWidth > 64 bit).

Set the default value for string parameters.

Parameters

<i>defaultString</i>	The defaultString field of the ParameterInfo object.
----------------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.11 setDescription()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setDescription (
    const std::string & description ) [inline]
```

Set the description field.

Parameters

<i>description</i>	The description field of the ResourceInfo object.
--------------------	---

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.12 setFormat()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setFormat (
    const std::string & format ) [inline]
```

Set the `format` field.

Parameters

<i>format</i>	The format field of the ResourceInfo object.
---------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.13 setHidden()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setHidden (
    bool hidden = true ) [inline]
```

Set the resource to hidden.

Parameters

<i>hidden</i>	If true, this resource is not listed in <code>resource_getList()</code> calls
---------------	---

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.14 setInitOnly()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setInitOnly (
    bool initOnly = true ) [inline]
```

Set the `initOnly` flag of a parameter.

This also implicitly sets the parameter to read-only.

Parameters

<i>initOnly</i>	The <code>initOnly</code> flag of a parameter.
-----------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.15 setMax() [1/2]

```
template<typename T >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMax (
    std::initializer_list< T > && t ) [inline]
```

Set the `max` field for wide numeric parameters.

This function accepts a braced initializer-list and is otherwise identical to [setMaxFromContainer\(\)](#).

Each element will be promoted/narrowed to `uint64_t`.

Parameters

<i>t</i>	Braced initializer-list.
----------	--------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.16 setMax() [2/2]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMax (
    uint64_t value ) [inline]
```

Set the `max` field to a value ≤ 64 bit.

If the parameter is wider than the passed value the value is zero extended.

If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>value</i>	Max value of the parameter.
--------------	-----------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.17 setMaxFromContainer()

```
template<typename Container >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMaxFromContainer (
    const Container & container ) [inline]
```

Set the `max` field for wide numeric parameters.

Container must be a type which allows to iterate over `uint64_t` bit chunks of the value, least significant bits first, for example `std::array<uint64_t>` or `std::vector<uint64_t>`.

Each element of the container will be promoted/narrowed to `uint64_t`.

If the parameter is wider than the passed value the value is zero extended.

If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>container</i>	Container containing the value in 64-bit chunks.
------------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.18 setMin() [1/2]

```
template<typename T >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMin (
    std::initializer_list< T > && t ) [inline]
```

Set the `min` field for wide numeric parameters.

This function accepts a braced initializer-list and is otherwise identical to [setMinFromContainer\(\)](#).

Each element will be promoted/narrowed to `uint64_t`.

Parameters

<i>t</i>	Braced initializer-list.
----------	--------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.19 setMin() [2/2]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMin (
    uint64_t value ) [inline]
```

Set the `min` field to a value ≤ 64 bit.

If the parameter is wider than the passed value the value is zero extended.

If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>value</i>	min value of the parameter.
--------------	-----------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.20 setMinFromContainer()

```
template<typename Container >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMinFromContainer (
    const Container & container ) [inline]
```

Set the `min` field for wide numeric parameters.

Container must be a type which allows to iterate over `uint64_t` bit chunks of the value, least significant bits first, for example `std::array<uint64_t>` or `std::vector<uint64_t>`.

Each element of the container will be promoted/narrowed to `uint64_t`.

If the parameter is wider than the passed value the value is zero extended.

If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>container</i>	Container containing the value in 64-bit chunks.
------------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.21 setName()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setName (
    const std::string & name ) [inline]
```

Set the name field.

Parameters

<i>name</i>	The name field of the ResourceInfo object.
-------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.22 setParentRscId()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setParentRscId (
    ResourceId parentRscId ) [inline]
```

Set the parentRscId field.

This function makes this register a child of the specified parent. It is not necessary to call this function when adding child registers using the addField() function.

Parameters

<i>parent↔ RscId</i>	The rscl of the parent register.
--------------------------	----------------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.23 setReadDelegate() [1/3]

```
template<IrisErrorCode(*)>(const ResourceInfo &, ResourceReadResult &) FUNC>
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setReadDelegate ( ) [inline]
```

Set the delegate to read the resource.

Set a delegate which calls function FUNC().

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceReadDelegate](#)

Template Parameters

<i>FUNC</i>	A resource read delegate function.
-------------	------------------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.24 setReadDelegate() [2/3]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setReadDelegate (
```

```
ResourceReadDelegate readDelegate ) [inline]
```

Set the delegate to read the resource.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceReadDelegate](#)

Parameters

<i>readDelegate</i>	ResourceReadDelegate object.
---------------------	------------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.25 setReadDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setReadDelegate (
    T * instance ) [inline]
```

Set the delegate to read the resource.

Set a delegate which calls METHOD() on an instance of class T.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceReadDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a resource read delegate.
<i>METHOD</i>	A resource read delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.26 setRwMode()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setRwMode (
    const std::string & rwMode ) [inline]
```

Set the `rwMode` field.

Parameters

<i>rwMode</i>	The <code>rwMode</code> field of the <code>ResourceInfo</code> object.
---------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.27 setSubRscId()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setSubRscId (
    uint64_t subRscId ) [inline]
```

Set the subRscId field.

Parameters

<i>sub↔ RscId</i>	The subRscId field of the ResourceInfo object.
-----------------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.28 setTag() [1/2]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setTag (
    const std::string & tag ) [inline]
```

Set the named boolean tag to true (e.g. isPc)

Parameters

<i>tag</i>	The name of the tag to set.
------------	-----------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.29 setTag() [2/2]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setTag (
    const std::string & tag,
    const IrisValue & value ) [inline]
```

Set a tag to the specified value.

Parameters

<i>tag</i>	The name of the tag to set.
<i>value</i>	The value to set the tag to.

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.30 setType()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setType (
    const std::string & type ) [inline]
```

Set the `type` field.

Parameters

<i>type</i>	The type field of the ResourceInfo object.
-------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.31 setWriteDelegate() [1/3]

```
template<IrisErrorCode(*) (const ResourceInfo &, const ResourceWriteValue &) FUNC>
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setWriteDelegate ( ) [inline]
```

Set the delegate to write the resource.

Set a delegate which calls function FUNC().

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceWriteDelegate](#)

Template Parameters

<i>FUNC</i>	A resource write delegate function.
-------------	-------------------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.32 setWriteDelegate() [2/3]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setWriteDelegate (
    ResourceWriteDelegate writeDelegate ) [inline]
```

Set the delegate to write the resource.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceWriteDelegate](#)

Parameters

<i>writeDelegate</i>	ResourceWriteDelegate object.
----------------------	-------------------------------

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

8.43.2.33 setWriteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &)
METHOD>
```

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setWriteDelegate (
    T * instance ) [inline]
```

Set the delegate to write the resource.

Set a delegate which calls METHOD() on an instance of class T.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceWriteDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a resource write delegate.
<i>METHOD</i>	A resource write delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [ParameterBuilder](#) object allowing calls to be chained together.

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

- [IrisInstanceBuilder.h](#)

8.44 iris::IrisInstanceEvent::ProxyEventInfo Struct Reference

Contains information for a single proxy EventSource.

```
#include <IrisInstanceEvent.h>
```

Public Attributes

- `std::vector< EventStreamId > evStreamIds`
- EventSourceId **targetEvSrcId** {}
- InstanceId **targetInstId** {}

8.44.1 Detailed Description

Contains information for a single proxy EventSource.

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

- [IrisInstanceEvent.h](#)

8.45 iris::IrisInstanceBuilder::RegisterBuilder Class Reference

Used to set metadata on a register resource.

```
#include <IrisInstanceBuilder.h>
```

Public Member Functions

- [RegisterBuilder](#) & [addEnum](#) (const std::string &symbol, const IrisValue &value, const std::string &description=std::string())

Add a symbol to the enums field for numeric resources.

- **FieldBuilder** **addField** (const std::string &name, uint64_t lsbOffset, uint64_t bitWidth, const std::string &description)
Add a subregister field to this register. By default, the field copies attributes from its parent register, but any field can be overridden.
- **FieldBuilder** **addLogicalField** (const std::string &name, uint64_t bitWidth, const std::string &description)
Add a logical subregister field to this register. A logical field is a field which has a bitwidth, but which does not have an lsbOffset. It is usually used to represent non-contiguous fields which are distributed across multiple chunks in the parent register as a single contiguous register. This allows to attach enums to such a field.
- **RegisterBuilder** & **addStringEnum** (const std::string &stringValue, const std::string &description=std::string())
Add a symbol to the enums field for string resources.
- ResourceId **getRscId** () const
Return the rscId that was allocated for this resource.
- **RegisterBuilder** & **getRscId** (ResourceId &rscIdOut)
Get the rscId that was allocated for this resource.
- **RegisterBuilder** (**IrisInstanceResource::ResourceInfoAndAccess** &info_, **IrisInstanceResource** *inst_ ↵ resource_, **IrisInstanceBuilder** *instance_builder_)
- **RegisterBuilder** & **setAddressOffset** (uint64_t addressOffset)
Set the addressOffset field.
- **RegisterBuilder** & **setBitWidth** (uint64_t bitWidth)
Set the bitWidth field.
- **RegisterBuilder** & **setBreakpointSupportInfo** (const std::string &supported)
Set the breakpointSupport field.
- **RegisterBuilder** & **setCanonicalRn** (uint64_t canonicalRn)
Set the canonicalRn field.
- **RegisterBuilder** & **setCanonicalRnElfDwarf** (uint16_t architecture, uint16_t dwarfRegNum)
Set the canonicalRn field for "ElfDwarf" scheme.
- **RegisterBuilder** & **setCname** (const std::string &cname)
Set the cname field.
- **RegisterBuilder** & **setDescr** (const std::string &description)
Obsolete alias for setDescription(). Do not use.
- **RegisterBuilder** & **setDescription** (const std::string &description)
Set the description field.
- **RegisterBuilder** & **setFormat** (const std::string &format)
Set the format field.
- **RegisterBuilder** & **setLsbOffset** (uint64_t lsbOffset)
Set the lsbOffset field.
- **RegisterBuilder** & **setName** (const std::string &name)
Set the name field.
- **RegisterBuilder** & **setParentRscId** (ResourceId parentRscId)
Set the parentRscId field.
- template<IrisErrorCode(*)>(const ResourceInfo &, ResourceReadResult &) FUNC>
RegisterBuilder & **setReadDelegate** ()
Set the delegate to read the resource.
- **RegisterBuilder** & **setReadDelegate** (**ResourceReadDelegate** readDelegate)
Set the delegate to read the resource.
- template<typename T, IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>
RegisterBuilder & **setReadDelegate** (T *instance)
Set the delegate to read the resource.
- template<typename T>
RegisterBuilder & **setResetData** (std::initializer_list< T > &&t)
Set the resetData field for wide registers.
- **RegisterBuilder** & **setResetData** (uint64_t value)

- Set the `resetData` field to a value ≤ 64 bit.*

 - `template<typename Container >`
[RegisterBuilder](#) & [setResetDataFromContainer](#) (const Container &container)

Set the `resetData` field for wide registers.

 - [RegisterBuilder](#) & [setResetString](#) (const std::string &resetString)

Set the `resetString` field.

 - [RegisterBuilder](#) & [setRwMode](#) (const std::string &rwMode)

Set the `rwMode` field.

 - [RegisterBuilder](#) & [setSubRscId](#) (uint64_t subRscId)

Set the `subRscId` field.

 - [RegisterBuilder](#) & [setTag](#) (const std::string &tag)

Set the named boolean tag to true (e.g. `isPc`)

 - [RegisterBuilder](#) & [setTag](#) (const std::string &tag, const IrisValue &value)

Set a tag to the specified value.

 - [RegisterBuilder](#) & [setType](#) (const std::string &type)

Set the `type` field.

 - `template<IrisErrorCode(*)>(const ResourceInfo &, const ResourceWriteValue &) FUNC>`
[RegisterBuilder](#) & [setWriteDelegate](#) ()

Set the delegate to write the resource.

 - [RegisterBuilder](#) & [setWriteDelegate](#) ([ResourceWriteDelegate](#) writeDelegate)

Set the delegate to write the resource.

 - `template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &) METHOD>`
[RegisterBuilder](#) & [setWriteDelegate](#) (T *instance)

Set the delegate to write the resource.

 - `template<typename T >`
[RegisterBuilder](#) & [setWriteMask](#) (std::initializer_list< T > &&t)

Set the `writeMask` field for wide registers.

 - [RegisterBuilder](#) & [setWriteMask](#) (uint64_t value)

Set the `writeMask` field to a value ≤ 64 bit.

 - `template<typename Container >`
[RegisterBuilder](#) & [setWriteMaskFromContainer](#) (const Container &container)

Set the `writeMask` field for wide registers.

8.45.1 Detailed Description

Used to set metadata on a register resource.

8.45.2 Member Function Documentation

8.45.2.1 addEnum()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::addEnum (
    const std::string & symbol,
    const IrisValue & value,
    const std::string & description = std::string() ) [inline]
```

Add a symbol to the enums field for numeric resources.

This should be called multiple times to add multiple symbols.

Parameters

<i>symbol</i>	The symbol string to be associated with the specified value.
<i>value</i>	The value of this symbol.
<i>description</i>	A description of this symbol.

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.2 addField()

```
FieldBuilder iris::IrisInstanceBuilder::RegisterBuilder::addField (
    const std::string & name,
    uint64_t lsbOffset,
    uint64_t bitWidth,
    const std::string & description )
```

Add a subregister field to this register. By default, the field copies attributes from its parent register, but any field can be overridden.

Parameters

<i>name</i>	Name of the register field.
<i>lsbOffset</i>	The bit offset of this field inside its parent register.
<i>bitWidth</i>	The size of the field.
<i>description</i>	Description of this field.

Returns

A [FieldBuilder](#) object that allows the caller to set attributes for this field.

8.45.2.3 addLogicalField()

```
FieldBuilder iris::IrisInstanceBuilder::RegisterBuilder::addLogicalField (
    const std::string & name,
    uint64_t bitWidth,
    const std::string & description )
```

Add a logical subregister field to this register. A logical field is a field which has a bitwidth, but which does not have an lsbOffset. It is usually used to represent non-contiguous fields which are distributed across multiple chunks in the parent register as a single contiguous register. This allows to attach enums to such a field. By default, the field copies attributes from its parent register, but any field can be overridden.

Parameters

<i>name</i>	Name of the register field.
<i>bitWidth</i>	The size of the field.
<i>description</i>	Description of this field.

Returns

A [FieldBuilder](#) object that allows the caller to set attributes for this field.

8.45.2.4 addStringEnum()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::addStringEnum (
    const std::string & stringValue,
    const std::string & description = std::string() ) [inline]
```

Add a symbol to the enums field for string resources.

This should be called multiple times to add multiple symbols.

Parameters

<i>value</i>	The string value of this symbol. This is also used as the symbols string.
<i>description</i>	A description of this symbol.

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.5 getRscId() [1/2]

```
ResourceId iris::IrisInstanceBuilder::RegisterBuilder::getRscId ( ) const [inline]
```

Return the rscId that was allocated for this resource.

Returns

The rscId that was allocated for this resource.

8.45.2.6 getRscId() [2/2]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::getRscId (
    ResourceId & rscIdOut ) [inline]
```

Get the rscId that was allocated for this resource.

This variant is useful to get the ResourceId of fields added in a chained call where return values are not practical.

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.7 setAddressOffset()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setAddressOffset (
    uint64_t addressOffset ) [inline]
```

Set the addressOffset field.

Parameters

<i>addressOffset</i>	The addressOffset field of the RegisterInfo object.
----------------------	---

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.8 setBitWidth()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setBitWidth (
    uint64_t bitWidth ) [inline]
```

Set the bitWidth field.

Parameters

<i>bitWidth</i>	The bitWidth field of the ResourceInfo object.
-----------------	--

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.9 setBreakpointSupportInfo()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setBreakpointSupportInfo (
    const std::string & supported ) [inline]
```

Set the breakpointSupport field.

Parameters

<i>supported</i>	The breakpointSupport field of the RegisterInfo object.
------------------	---

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.10 setCanonicalRn()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setCanonicalRn (
    uint64_t canonicalRn_ ) [inline]
```

Set the canonicalRn field.

Note: Use [setCanonicalRnElfDwarf\(\)](#) when using the "ElfDwarf" scheme.

Parameters

<i>canonicalRn</i>	The canonicalRn field of the RegisterInfo object.
--------------------	---

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.11 setCanonicalRnElfDwarf()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setCanonicalRnElfDwarf (
    uint16_t architecture,
    uint16_t dwarfRegNum ) [inline]
```

Set the canonicalRn field for "ElfDwarf" scheme.

Parameters

<i>architecture</i>	ELF EM_* constant for architecture.
<i>dwarfRegNum</i>	DWARF register number for architecture.

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.12 setName()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setName (
```



```
const std::string & cname ) [inline]
```

Set the `cname` field.

Parameters

<i>cname</i>	The <code>cname</code> field of the <code>ResourceInfo</code> object.
--------------	---

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.13 setDescription()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setDescription (
    const std::string & description ) [inline]
```

Set the `description` field.

Parameters

<i>description</i>	The <code>description</code> field of the <code>ResourceInfo</code> object.
--------------------	---

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.14 setFormat()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setFormat (
    const std::string & format ) [inline]
```

Set the `format` field.

Parameters

<i>format</i>	The <code>format</code> field of the <code>ResourceInfo</code> object.
---------------	--

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.15 setLsbOffset()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setLsbOffset (
    uint64_t lsbOffset ) [inline]
```

Set the `lsbOffset` field.

Parameters

<i>lsbOffset</i>	The <code>lsbOffset</code> field of the <code>RegisterInfo</code> object.
------------------	---

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.16 setName()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setName (
    const std::string & name ) [inline]
```

Set the name field.

Parameters

<i>name</i>	The name field of the ResourceInfo object.
-------------	--

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.17 setParentRscId()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setParentRscId (
    ResourceId parentRscId ) [inline]
```

Set the parentRscId field.

This function makes this register a child of the specified parent. It is not necessary to call this function when adding child registers using the [addField\(\)](#) function.

Parameters

<i>parent↔ RscId</i>	The rscId of the parent register.
--------------------------	-----------------------------------

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.18 setReadDelegate() [1/3]

```
template<IrisErrorCode(*)>(const ResourceInfo &, ResourceReadResult &) FUNC>
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setReadDelegate ( ) [inline]
```

Set the delegate to read the resource.

Set a delegate which calls function FUNC().

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceReadDelegate](#)

Template Parameters

<i>FUNC</i>	A resource read delegate function.
-------------	------------------------------------

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.19 setReadDelegate() [2/3]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setReadDelegate (
```

```
ResourceReadDelegate readDelegate ) [inline]
```

Set the delegate to read the resource.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceReadDelegate](#)

Parameters

<i>readDelegate</i>	ResourceReadDelegate object.
---------------------	------------------------------

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.20 setReadDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setReadDelegate (
    T * instance ) [inline]
```

Set the delegate to read the resource.

Set a delegate which calls METHOD() on an instance of class T.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceReadDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a resource read delegate.
<i>METHOD</i>	A resource read delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.21 setResetData() [1/2]

```
template<typename T >
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setResetData (
    std::initializer_list< T > && t ) [inline]
```

Set the `resetData` field for wide registers.

This function accepts a braced initializer-list and is otherwise identical to

[setResetDataFromContainer\(\)](#).

Each element will be promoted/narrowed to `uint64_t`.

Parameters

<i>t</i>	Braced initializer-list.
----------	--------------------------

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.22 setResetData() [2/2]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setResetData (
    uint64_t value ) [inline]
```

Set the `resetData` field to a value ≤ 64 bit.

If the register is wider than the passed value the value is zero extended.

If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>value</i>	resetData value of the register.
--------------	----------------------------------

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.23 setResetDataFromContainer()

```
template<typename Container >
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setResetDataFromContainer (
    const Container & container ) [inline]
```

Set the `resetData` field for wide registers.

Container must be a type which allows to iterate over `uint64_t` bit chunks of the value, least significant bits first, for example `std::array<uint64_t>` or `std::vector<uint64_t>`.

Each element of the container will be promoted/narrowed to `uint64_t`.

If the register is wider than the passed value the value is zero extended.

If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>container</i>	Container containing the value in 64-bit chunks.
------------------	--

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.24 setResetString()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setResetString (
    const std::string & resetString ) [inline]
```

Set the `resetString` field.

Set the reset value for string registers.

Parameters

<i>resetString</i>	The resetString field of the RegisterInfo object.
--------------------	---

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.25 setRwMode()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setRwMode (
    const std::string & rwMode ) [inline]
```

Set the `rwMode` field.

Parameters

<i>rwMode</i>	The <code>rwMode</code> field of the ResourceInfo object.
---------------	---

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.26 setSubRscId()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setSubRscId (
    uint64_t subRscId ) [inline]
```

Set the `subRscId` field.

Parameters

<i>sub↔ RscId</i>	The <code>subRscId</code> field of the ResourceInfo object.
-----------------------	---

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.27 setTag() [1/2]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setTag (
    const std::string & tag ) [inline]
```

Set the named boolean tag to true (e.g. `isPc`)

Parameters

<i>tag</i>	The name of the tag to set.
------------	-----------------------------

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.28 setTag() [2/2]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setTag (
    const std::string & tag,
    const IrisValue & value ) [inline]
```

Set a tag to the specified value.

Parameters

<i>tag</i>	The name of the tag to set.
<i>value</i>	The value to set the tag to.

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.29 setType()

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setType (
    const std::string & type ) [inline]
```

Set the `type` field.

Parameters

<i>type</i>	The type field of the ResourceInfo object.
-------------	--

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.30 setWriteDelegate() [1/3]

```
template<IrisErrorCode(*)> (const ResourceInfo &, const ResourceWriteValue &) FUNC>
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteDelegate ( ) [inline]
```

Set the delegate to write the resource.

Set a delegate which calls function `FUNC()`.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceWriteDelegate](#)

Template Parameters

<i>FUNC</i>	A resource write delegate function.
-------------	-------------------------------------

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.31 setWriteDelegate() [2/3]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteDelegate (
    ResourceWriteDelegate writeDelegate ) [inline]
```

Set the delegate to write the resource.
If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceWriteDelegate](#)

Parameters

<i>writeDelegate</i>	ResourceWriteDelegate object.
----------------------	-------------------------------

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.32 setWriteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &)
METHOD>
```

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteDelegate (
    T * instance ) [inline]
```

Set the delegate to write the resource.
Set a delegate which calls METHOD() on an instance of class T.
If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultResourceWriteDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a resource write delegate.
<i>METHOD</i>	A resource write delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.33 setWriteMask() [1/2]

```
template<typename T >
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteMask (
    std::initializer_list< T > && t ) [inline]
```

Set the writeMask field for wide registers.
This function accepts a braced initializer-list and is otherwise identical to [setWriteMaskFromContainer\(\)](#).
Each element will be promoted/narrowed to uint64_t.

Parameters

<i>t</i>	Braced initializer-list.
----------	--------------------------

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.34 setWriteMask() [2/2]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteMask (
    uint64_t value ) [inline]
```

Set the `writeMask` field to a value ≤ 64 bit.

If the register is wider than the passed value the value is zero extended.

If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>value</i>	writeMask value of the register.
--------------	----------------------------------

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

8.45.2.35 setWriteMaskFromContainer()

```
template<typename Container >
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteMaskFromContainer (
    const Container & container ) [inline]
```

Set the `writeMask` field for wide registers.

Container must be a type which allows to iterate over `uint64_t` bit chunks of the value, least significant bits first, for example `std::array<uint64_t>` or `std::vector<uint64_t>`.

Each element of the container will be promoted/narrowed to `uint64_t`.

If the register is wider than the passed value the value is zero extended.

If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

<i>container</i>	Container containing the value in 64-bit chunks.
------------------	--

Returns

A reference to this [RegisterBuilder](#) object allowing calls to be chained together.

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

- [IrisInstanceBuilder.h](#)

8.46 iris::IrisInstanceResource::ResourceInfoAndAccess Struct Reference

Entry in 'resourceInfos'.

```
#include <IrisInstanceResource.h>
```


Public Attributes

- [ResourceReadDelegate](#) **readDelegate**
- ResourceInfo **resourceInfo**
- [ResourceWriteDelegate](#) **writeDelegate**

8.46.1 Detailed Description

Entry in 'resourceInfos'.

Contains static resource information and information on how to access the resource.

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

- [IrisInstanceResource.h](#)

8.47 iris::ResourceWriteValue Struct Reference

```
#include <IrisInstanceResource.h>
```

Public Attributes

- const uint64_t * **data** {}
 - const std::string * **str** {}
- Non-null for non-string resources.*

8.47.1 Detailed Description

Write value for ResourceWriteDelegate. This struct is used as a union. At most one of the two pointers is non-null when ResourceWriteDelegate is invoked.

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

- [IrisInstanceResource.h](#)

8.48 iris::IrisInstanceBuilder::SemihostingManager Class Reference

semihosting_apis [IrisInstanceBuilder](#) semihosting APIs

```
#include <IrisInstanceBuilder.h>
```

Public Member Functions

- void **enableExtensions** ()
Instances that support semihosting extensions should call this function to enable the `IRIS_SEMIHOSTING_CALL_EXTENSION` event.
- std::vector< uint8_t > **readData** (uint64_t fDes, size_t max_size=0, uint64_t flags=semihost::DEFAULT)
Read data for a given file descriptor.
- std::pair< bool, uint64_t > **semihostedCall** (uint64_t operation, uint64_t parameter)
Allow a client to perform a semihosting extension defined by operation and parameter.
- **SemihostingManager** ([IrisInstanceSemihosting](#) *inst_semihost_)
- void **unblock** ()
- bool **writeData** (uint64_t fDes, const std::vector< uint8_t > &data)
- bool **writeData** (uint64_t fDes, const uint8_t *data, size_t size)

8.48.1 Detailed Description

semihosting_apis [IrisInstanceBuilder](#) semihosting APIs

Manage semihosting functionality

8.48.2 Member Function Documentation

8.48.2.1 readData()

```
std::vector< uint8_t > iris::IrisInstanceBuilder::SemihostingManager::readData (
    uint64_t fDes,
    size_t max_size = 0,
    uint64_t flags = semihost::DEFAULT ) [inline]
```

Read data for a given file descriptor.

The exact behavior of this method depends on the value of the `max_size` and `flags` parameters. If the `NONBLOCK` flag is set, the method returns immediately with whatever data is already buffered, if any. If `NONBLOCK` is not set, the method blocks until data is available. Iris messages continue to be processed while this methods blocks. If `max_size` is not zero, then at most `max_size` bytes will be returned.

Parameters

<i>fDes</i>	File descriptor to read from. Usually <code>semihost::STDIN</code> .
<i>max_size</i>	The maximum amount of bytes to read or zero for no limit.
<i>flags</i>	A bitwise OR of Semihosting data request flag constants .

Returns

A vector of data that was read.

8.48.2.2 semihostedCall()

```
std::pair< bool, uint64_t > iris::IrisInstanceBuilder::SemihostingManager::semihostedCall (
    uint64_t operation,
    uint64_t parameter ) [inline]
```

Allow a client to perform a semihosting extension defined by *operation* and *parameter*.

This might implement a user-defined operation or override the default implementation for a predefined operation.

Parameters

<i>operation</i>	A number indicating the operation to perform. This is defined by the semihosting standard for standard operations or by the client for user-defined operations.
<i>parameter</i>	A parameter to the operation. The meaning of this parameter is defined by the operation.

Returns

A pair of (bool success, uint64_t result). If success is true, a client performed the function and returned the value in result. If success is false, no client performed the function and result is 0.

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

- [IrisInstanceBuilder.h](#)

8.49 iris::IrisInstanceMemory::SpaceInfoAndAccess Struct Reference

Entry in 'spaceInfos'.

```
#include <IrisInstanceMemory.h>
```

Public Attributes

- [MemoryReadDelegate](#) `readDelegate`

- [MemoryGetSidebandInfoDelegate](#) **sidebandDelegate**
- [MemorySpaceInfo](#) **spaceInfo**
- [MemoryWriteDelegate](#) **writeDelegate**

8.49.1 Detailed Description

Entry in 'spaceInfos'.

Contains static memory space information and information on how to access the space.

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

- [IrisInstanceMemory.h](#)

8.50 iris::IrisInstanceBuilder::TableBuilder Class Reference

Used to set metadata for a table.

```
#include <IrisInstanceBuilder.h>
```

Public Member Functions

- [TableColumnBuilder](#) **addColumn** (const std::string &name)
Add a new column.
- [TableBuilder](#) & **addColumnInfo** (const TableColumnInfo &columnInfo)
Add a column with a preconstructed TableColumnInfo.
- [TableBuilder](#) & **setDescription** (const std::string &description)
Set the description field.
- [TableBuilder](#) & **setFormatLong** (const std::string &format)
Set the formatLong field.
- [TableBuilder](#) & **setFormatShort** (const std::string &format)
Set the formatShort field.
- [TableBuilder](#) & **setIndexFormatHint** (const std::string &hint)
Set the indexFormatHint field.
- [TableBuilder](#) & **setMaxIndex** (uint64_t maxIndex)
Set the maxIndex field.
- [TableBuilder](#) & **setMinIndex** (uint64_t minIndex)
Set the minIndex field.
- [TableBuilder](#) & **setName** (const std::string &name)
Set the name field.
- template<IrisErrorCode(*)>(const TableInfo &, uint64_t, uint64_t, TableReadResult &) FUNC>
[TableBuilder](#) & **setReadDelegate** ()
Set the delegate to read the table.
- template<typename T , IrisErrorCode(T::*)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) METHOD>
[TableBuilder](#) & **setReadDelegate** (T *instance)
Set the delegate to read the table.
- [TableBuilder](#) & **setReadDelegate** (TableReadDelegate delegate)
Set the delegate to read the table.
- template<IrisErrorCode(*)>(const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>
[TableBuilder](#) & **setWriteDelegate** ()
Set the delegate to write to the table.
- template<typename T , IrisErrorCode(T::*)(const TableInfo &, const TableRecords &, TableWriteResult &) METHOD>
[TableBuilder](#) & **setWriteDelegate** (T *instance)
Set the delegate to write to the table.
- [TableBuilder](#) & **setWriteDelegate** (TableWriteDelegate delegate)
Set the delegate to write to the table.
- [TableBuilder](#) ([IrisInstanceTable::TableInfoAndAccess](#) &info_)

8.50.1 Detailed Description

Used to set metadata for a table.

8.50.2 Member Function Documentation

8.50.2.1 addColumn()

```
IrisInstanceBuilder::TableColumnBuilder iris::IrisInstanceBuilder::TableBuilder::addColumn (
    const std::string & name ) [inline]
```

Add a new column.

Call this multiple times for multiple columns

See also

[AddColumnInfo](#)

Parameters

<i>name</i>	The name of the new column.
-------------	-----------------------------

Returns

A [TableColumnBuilder](#) object that can be used to add metadata for the new column.

8.50.2.2 addColumnInfo()

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::addColumnInfo (
    const TableColumnInfo & columnInfo ) [inline]
```

Add a column with a preconstructed TableColumnInfo.

Call this multiple times for multiple columns.

See also

[addColumn](#)

Parameters

<i>columnInfo</i>	A preconstructed TableColumnInfo object for the new column.
-------------------	---

Returns

A reference to this [TableBuilder](#) allowing calls to be chained together.

8.50.2.3 setDescription()

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setDescription (
    const std::string & description ) [inline]
```

Set the description field.

Parameters

<i>description</i>	The description field of the TableInfo object.
--------------------	--

Returns

A reference to this [TableBuilder](#) allowing calls to be chained together.

8.50.2.4 setFormatLong()

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setFormatLong (
    const std::string & format ) [inline]
```

Set the `formatLong` field.

Parameters

<i>format</i>	The <code>formatLong</code> field of the <code>TableInfo</code> object.
---------------	---

Returns

A reference to this [TableBuilder](#) allowing calls to be chained together.

8.50.2.5 setFormatShort()

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setFormatShort (
    const std::string & format ) [inline]
```

Set the `formatShort` field.

Parameters

<i>format</i>	The <code>formatShort</code> field of the <code>TableInfo</code> object.
---------------	--

Returns

A reference to this [TableBuilder](#) allowing calls to be chained together.

8.50.2.6 setIndexFormatHint()

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setIndexFormatHint (
    const std::string & hint ) [inline]
```

Set the `indexFormatHint` field.

Parameters

<i>hint</i>	The <code>indexFormatHint</code> field of the <code>TableInfo</code> object.
-------------	--

Returns

A reference to this [TableBuilder](#) allowing calls to be chained together.

8.50.2.7 setMaxIndex()

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setMaxIndex (
    uint64_t maxIndex ) [inline]
```

Set the `maxIndex` field.

Parameters

<i>maxIndex</i>	The maxIndex field of the TableInfo object.
-----------------	---

Returns

A reference to this [TableBuilder](#) allowing calls to be chained together.

8.50.2.8 setMinIndex()

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setMinIndex (
    uint64_t minIndex ) [inline]
```

Set the minIndex field.

Parameters

<i>minIndex</i>	The minIndex field of the TableInfo object.
-----------------	---

Returns

A reference to this [TableBuilder](#) allowing calls to be chained together.

8.50.2.9 setName()

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setName (
    const std::string & name ) [inline]
```

Set the name field.

Parameters

<i>name</i>	The name field of the TableInfo object.
-------------	---

Returns

A reference to this [TableBuilder](#) allowing calls to be chained together.

8.50.2.10 setReadDelegate() [1/3]

```
template<IrisErrorCode(*)>(const TableInfo &, uint64_t, uint64_t, TableReadResult &) FUNC>
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setReadDelegate ( ) [inline]
```

Set the delegate to read the table.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultTableReadDelegate](#)

Template Parameters

<i>FUNC</i>	A table read delegate function.
-------------	---------------------------------

Returns

A reference to this [TableBuilder](#) object allowing calls to be chained together.

8.50.2.11 setReadDelegate() [2/3]

```
template<typename T , IrisErrorCode(T::*)(const TableInfo &, uint64_t, uint64_t, TableRead←
Result &) METHOD>
```

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setReadDelegate (
    T * instance ) [inline]
```

Set the delegate to read the table.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultTableReadDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a table read delegate.
<i>METHOD</i>	A table read delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [TableBuilder](#) object allowing calls to be chained together.

8.50.2.12 setReadDelegate() [3/3]

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setReadDelegate (
    TableReadDelegate delegate ) [inline]
```

Set the delegate to read the table.

If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultTableReadDelegate](#)

Parameters

<i>delegate</i>	TableReadDelegate object.
-----------------	---------------------------

Returns

A reference to this [TableBuilder](#) object allowing calls to be chained together.

8.50.2.13 setWriteDelegate() [1/3]

```
template<IrisErrorCode(*) (const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setWriteDelegate ( ) [inline]
```

Set the delegate to write to the table.
If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultTableWriteDelegate](#)

Template Parameters

<i>FUNC</i>	A table write delegate function.
-------------	----------------------------------

Returns

A reference to this [TableBuilder](#) object allowing calls to be chained together.

8.50.2.14 setWriteDelegate() [2/3]

```
template<typename T , IrisErrorCode(T::*)(const TableInfo &, const TableRecords &, Table←
WriteResult &) METHOD>
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setWriteDelegate (
    T * instance ) [inline]
```

Set the delegate to write to the table.
If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultTableWriteDelegate](#)

Template Parameters

<i>T</i>	A class that defines a method with the right signature to be a table write delegate.
<i>METHOD</i>	A table write delegate method in class T.

Parameters

<i>instance</i>	The instance of class T on which to call METHOD.
-----------------	--

Returns

A reference to this [TableBuilder](#) object allowing calls to be chained together.

8.50.2.15 setWriteDelegate() [3/3]

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setWriteDelegate (
    TableWriteDelegate delegate ) [inline]
```

Set the delegate to write to the table.
If this is not set, the default delegate is used.

See also

[IrisInstanceBuilder::setDefaultTableWriteDelegate](#)

Parameters

<i>delegate</i>	TableWriteDelegate object.
-----------------	----------------------------

Returns

A reference to this [TableBuilder](#) object allowing calls to be chained together.

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

- [IrisInstanceBuilder.h](#)

8.51 iris::IrisInstanceBuilder::TableColumnBuilder Class Reference

Used to set metadata for a table column.

```
#include <IrisInstanceBuilder.h>
```

Public Member Functions

- [TableColumnBuilder](#) **addColumn** (const std::string &name)
Add another new column.
- [TableBuilder](#) & **addColumnInfo** (const TableColumnInfo &columnInfo)
Add another column with a preconstructed TableColumnInfo.
- [TableBuilder](#) & **endColumn** ()
Stop building this column and go back to the parent table.
- [TableColumnBuilder](#) & **setBitWidth** (uint64_t bitWidth)
Set the bitWidth field.
- [TableColumnBuilder](#) & **setDescription** (const std::string &description)
Set the description field.
- [TableColumnBuilder](#) & **setFormat** (const std::string &format)
Set the format field.
- [TableColumnBuilder](#) & **setFormatLong** (const std::string &format)
Set the formatLong field.
- [TableColumnBuilder](#) & **setFormatShort** (const std::string &format)
Set the formatShort field.
- [TableColumnBuilder](#) & **setName** (const std::string &name)
Set the name field.
- [TableColumnBuilder](#) & **setRwMode** (const std::string &rwMode)
Set the rwMode field.
- [TableColumnBuilder](#) & **setType** (const std::string &type)
Set the type field.
- **TableColumnBuilder** ([TableBuilder](#) &parent_, TableColumnInfo &info_)

8.51.1 Detailed Description

Used to set metadata for a table column.

8.51.2 Member Function Documentation

8.51.2.1 addColumn()

```
TableColumnBuilder iris::IrisInstanceBuilder::TableColumnBuilder::addColumn (
    const std::string & name ) [inline]
```

Add another new column.

Call this multiple times for multiple columns

See also

[TableBuilder::addColumn](#)

Parameters

<i>name</i>	The name of the new column.
-------------	-----------------------------

Returns

A [TableColumnBuilder](#) object that can be used to add metadata for the new column.

8.51.2.2 addColumnInfo()

```
TableBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::addColumnInfo (
    const TableColumnInfo & columnInfo ) [inline]
```

Add another column with a preconstructed TableColumnInfo.

See also

[TableBuilder::addColumnInfo](#)

[addColumn](#)

Parameters

<i>columnInfo</i>	A preconstructed TableColumnInfo object for the new column.
-------------------	---

Returns

A reference to the parent [TableBuilder](#) for this table.

8.51.2.3 endColumn()

```
TableBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::endColumn ( ) [inline]
```

Stop building this column and go back to the parent table.

See also

[addColumn](#)

[addColumnInfo](#)

Returns

The parent [TableBuilder](#) for this table.

8.51.2.4 setBitWidth()

```
TableColumnBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::setBitWidth (
    uint64_t bitWidth ) [inline]
```

Set the `bitWidth` field.

Parameters

<i>bitWidth</i>	The bitWidth field of the TableColumnInfo object.
-----------------	---

Returns

A [TableColumnBuilder](#) object that can be used to add metadata for the new column.

8.51.2.5 setDescription()

```
TableColumnBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::setDescription (
    const std::string & description ) [inline]
```

Set the description field.

Parameters

<i>description</i>	The description field of the TableColumnInfo object.
--------------------	--

Returns

A [TableColumnBuilder](#) object that can be used to add metadata for the new column.

8.51.2.6 setFormat()

```
TableColumnBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::setFormat (
    const std::string & format ) [inline]
```

Set the format field.

Parameters

<i>format</i>	The format field of the TableColumnInfo object.
---------------	---

Returns

A [TableColumnBuilder](#) object that can be used to add metadata for the new column.

8.51.2.7 setFormatLong()

```
TableColumnBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::setFormatLong (
    const std::string & format ) [inline]
```

Set the formatLong field.

Parameters

<i>format</i>	The formatLong field of the TableColumnInfo object.
---------------	---

Returns

A [TableColumnBuilder](#) object that can be used to add metadata for the new column.

8.51.2.8 setFormatShort()

```
TableColumnBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::setFormatShort (
    const std::string & format ) [inline]
```

Set the `formatShort` field.

Parameters

<i>format</i>	The <code>formatShort</code> field of the <code>TableColumnInfo</code> object.
---------------	--

Returns

A [TableColumnBuilder](#) object that can be used to add metadata for the new column.

8.51.2.9 setName()

```
TableColumnBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::setName (
    const std::string & name ) [inline]
```

Set the `name` field.

Parameters

<i>name</i>	The <code>name</code> field of the <code>TableColumnInfo</code> object.
-------------	---

Returns

A [TableColumnBuilder](#) object that can be used to add metadata for the new column.

8.51.2.10 setRwMode()

```
TableColumnBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::setRwMode (
    const std::string & rwMode ) [inline]
```

Set the `rwMode` field.

Parameters

<i>rwMode</i>	The <code>rwMode</code> field of the <code>TableColumnInfo</code> object.
---------------	---

Returns

A [TableColumnBuilder](#) object that can be used to add metadata for the new column.

8.51.2.11 setType()

```
TableColumnBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::setType (
    const std::string & type ) [inline]
```

Set the `type` field.

Parameters

<i>type</i>	The type field of the TableColumnInfo object.
-------------	---

Returns

A [TableColumnBuilder](#) object that can be used to add metadata for the new column.

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

- [IrisInstanceBuilder.h](#)

8.52 iris::IrisInstanceTable::TableInfoAndAccess Struct Reference

Entry in 'tableInfos'.

```
#include <IrisInstanceTable.h>
```

Public Attributes

- [TableReadDelegate](#) **readDelegate**
Can be empty, in which case defaultReadDelegate is used.
- TableInfo **tableInfo**
- [TableWriteDelegate](#) **writeDelegate**
Can be empty, in which case defaultWriteDelegate is used.

8.52.1 Detailed Description

Entry in 'tableInfos'.

Contains static table information and information on how to access the table.

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

- [IrisInstanceTable.h](#)

Chapter 9

File Documentation

9.1 IrisCanonicalMsnArm.h File Reference

Constants for the memory.canonicalMsnScheme arm.com/memoryspaces.

```
#include "iris/detail/IrisInterface.h"
#include "iris/detail/IrisCommon.h"
```

Enumerations

- enum **CanonicalMsnArm** : uint64_t {
 CanonicalMsnArm_SecureMonitor = 0x1000 , **CanonicalMsnArm_Secure** = 0x1000 , **CanonicalMsnArm_Guest** = 0x1001 , **CanonicalMsnArm_Normal** = 0x1001 ,
 CanonicalMsnArm_NSHyp = 0x1002 , **CanonicalMsnArm_Memory** = 0x1003 , **CanonicalMsnArm_HypApp** = 0x1004 , **CanonicalMsnArm_Host** = 0x1005 ,
 CanonicalMsnArm_Current = 0x10ff , **CanonicalMsnArm_IPA** = 0x1100 , **CanonicalMsnArm_PhysicalMemorySecure** = 0x1200 , **CanonicalMsnArm_PhysicalMemoryNonSecure** = 0x1201 ,
 CanonicalMsnArm_PhysicalMemory = 0x1202 , **CanonicalMsnArm_PhysicalMemoryRoot** = 0x1203 ,
 CanonicalMsnArm_PhysicalMemoryRealm = 0x1204 }

9.1.1 Detailed Description

Constants for the memory.canonicalMsnScheme arm.com/memoryspaces.

Date

Copyright ARM Limited 2022. All Rights Reserved.

9.2 IrisCanonicalMsnArm.h

[Go to the documentation of this file.](#)

```
1
2
3 8 #ifndef ARM_INCLUDE_IrisCanonicalMsnArm_h
4 9 #define ARM_INCLUDE_IrisCanonicalMsnArm_h
5 10
6 11 #include "iris/detail/IrisInterface.h" // uint64_t
7 12 #include "iris/detail/IrisCommon.h"   // namespace iris
8 13
9 14 NAMESPACE_IRIS_START
10 15
11 16 enum CanonicalMsnArm: uint64_t
12 17 {
13 18     CanonicalMsnArm_SecureMonitor = 0x1000,    CanonicalMsnArm_Secure      = 0x1000,
14 19     CanonicalMsnArm_Guest         = 0x1001,    CanonicalMsnArm_Normal     = 0x1001,
15 20     CanonicalMsnArm_NSHyp         = 0x1002,
16 21     CanonicalMsnArm_Memory        = 0x1003,    // Virtual memory for cores which do not have TrustZone.
17 22     CanonicalMsnArm_HypApp        = 0x1004,
18 23     CanonicalMsnArm_Host          = 0x1005,
19 24
20 25     CanonicalMsnArm_Current       = 0x10ff,
21 26
```



```

27     CanonicalMsnArm_IPA           = 0x1100,
28
29     CanonicalMsnArm_PhysicalMemorySecure = 0x1200,
30     CanonicalMsnArm_PhysicalMemoryNonSecure = 0x1201,
31     CanonicalMsnArm_PhysicalMemory      = 0x1202,
32     CanonicalMsnArm_PhysicalMemoryRoot  = 0x1203,
33     CanonicalMsnArm_PhysicalMemoryRealm  = 0x1204
34 }; // enum CanonicalMsnArm
35
36 NAMESPACE_IRIS_END
37
38 #endif // ARM_INCLUDE_IrisCanonicalMsnArm_h
39

```

9.3 IrisCConnection.h File Reference

IrisConnectionInterface implementation based on IrisC.

```

#include "iris/detail/IrisC.h"
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisErrorException.h"
#include "iris/detail/IrisInterface.h"
#include "iris/detail/IrisJsonProducer.h"
#include <string>

```

Classes

- class [iris::IrisCConnection](#)

Provide an IrisConnectionInterface which loads an IrisC library.

9.3.1 Detailed Description

IrisConnectionInterface implementation based on IrisC.

Copyright

Copyright (C) 2017-2023 Arm Limited. All rights reserved.

9.4 IrisCConnection.h

[Go to the documentation of this file.](#)

```

1
7 #ifndef ARM_INCLUDE_IrisCConnection_h
8 #define ARM_INCLUDE_IrisCConnection_h
9
10 #include "iris/detail/IrisC.h"
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisErrorException.h"
13 #include "iris/detail/IrisInterface.h"
14 #include "iris/detail/IrisJsonProducer.h"
15
16 #include <string>
17
18 NAMESPACE_IRIS_START
19
25 class IrisCConnection : public IrisConnectionInterface
26 {
27 private:
28     IrisC_HandleMessageFunction    handleMessage_function;
29
30     IrisC_RegisterChannelFunction   registerChannel_function;
31     IrisC_UnregisterChannelFunction unregisterChannel_function;
32
33     IrisC_ProcessAsyncMessagesFunction processAsyncMessages_function;
34
35     class RemoteInterface : public IrisInterface
36     {
37     private:
38         IrisCConnection* irisc;
39
40     public:
41         RemoteInterface(IrisCConnection* irisc_)
42

```

```

43         : irisc(irisc_)
44     {
45     }
46
47     public: // IrisInterface
48         virtual void irisHandleMessage(const uint64_t* message) IRIS_OVERRIDE
49     {
50         // Forward to the IrisC library
51         int64_t status = irisc->IrisC_handleMessage(message);
52
53         if (status != E_ok)
54         {
55             throw IrisErrorException(IrisErrorCode(status));
56         }
57     }
58     } remote_interface;
59
60     // Helper function to bridge IrisC_HandleMessageFunction to IrisInterface::irisHandleMessage
61     static int64_t handleMessageToIrisInterface(void* context, const uint64_t* message)
62     {
63         if (context == nullptr)
64         {
65             return E_invalid_context;
66         }
67         try
68         {
69             static_cast<IrisInterface*>(context)->irisHandleMessage(message);
70         }
71         catch (std::exception& e)
72         {
73             // Catch and print all exceptions here as they usually get silently dropped when going
74             // back through the C function.
75             // These are always programming errors (e.g. in plugin event callbacks) and not
76             // valid error return values of Iris functions.
77             std::cout << "Caught exception on plugin C boundary: " << e.what() << "\n";
78             std::cout << "Call was: " << messageToString(message) << "\n";
79
80             // Some compilers can transport exceptions through C functions, some not.
81             // Do whatever the compiler can do.
82             throw;
83         }
84     }
85     return E_ok;
86 }
87
88 protected:
89     void* iris_c_context;
90
91     IrisCConnection()
92     : handleMessage_function(nullptr)
93     , registerChannel_function(nullptr)
94     , unregisterChannel_function(nullptr)
95     , processAsyncMessages_function(nullptr)
96     , remote_interface(this)
97     , iris_c_context(nullptr)
98     {
99     }
100
101     int64_t IrisC_handleMessage(const uint64_t* message)
102     {
103         return (*handleMessage_function)(iris_c_context, message);
104     }
105
106     int64_t IrisC_registerChannel(IrisC_CommunicationChannel* channel, uint64_t* channel_id_out)
107     {
108         return (*registerChannel_function)(iris_c_context, channel, channel_id_out);
109     }
110
111     int64_t IrisC_unregisterChannel(uint64_t channel_id)
112     {
113         return (*unregisterChannel_function)(iris_c_context, channel_id);
114     }
115
116     int64_t IrisC_processAsyncMessages(bool waitForAMessage)
117     {
118         return (*processAsyncMessages_function)(iris_c_context, waitForAMessage);
119     }
120
121     public:
122     IrisCConnection(IrisC_Functions* functions)
123     : handleMessage_function(functions->handleMessage_function)
124     , registerChannel_function(functions->registerChannel_function)
125     , unregisterChannel_function(functions->unregisterChannel_function)
126     , processAsyncMessages_function(functions->processAsyncMessages_function)
127     , remote_interface(this)
128     , iris_c_context(functions->iris_c_context)

```

```

133     {
134     }
135
136 public: // IrisConnectionInterface
141     virtual uint64_t registerIrisInterfaceChannel(IrisInterface* iris_interface, const std::string&
connectionInfo) IRIS_OVERRIDE
142     {
143         (void)connectionInfo;
144         IrisC_CommunicationChannel channel;
145
146         channel.CommunicationChannel_version = 0;
147         channel.handleMessage_function      = &IrisCConnection::handleMessageToIrisInterface;
148         channel.handleMessage_context      = static_cast<void*>(iris_interface);
149
150         uint64_t channelId = IRIS_UINT64_MAX;
151
152         IrisErrorCode status = static_cast<IrisErrorCode>(IrisC_registerChannel(&channel, &channelId));
153
154         if (status != E_ok)
155         {
156             throw IrisErrorException(status);
157         }
158
159         return channelId;
160     }
161
162     virtual void unregisterIrisInterfaceChannel(uint64_t channelId) IRIS_OVERRIDE
163     {
164         IrisErrorCode status = static_cast<IrisErrorCode>(IrisC_unregisterChannel(channelId));
165
166         if (status != E_ok)
167         {
168             throw IrisErrorException(status);
169         }
170     }
171
172     virtual IrisErrorCode processAsyncMessages(bool waitForAMessage) IRIS_OVERRIDE
173     {
174         return static_cast<IrisErrorCode>(IrisC_processAsyncMessages(waitForAMessage));
175     }
176
177     virtual IrisInterface* getIrisInterface() IRIS_OVERRIDE
178     {
179         return &remote_interface;
180     }
181 };
182
183 namespace IRIS_END
184
185 #endif // ARM_INCLUDE_IrisCConnection_h

```

9.5 IrisClient.h File Reference

Iris client which supports multiple methods to connect to other Iris executables.

```

#include "iris/IrisInstance.h"
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisErrorCode.h"
#include "iris/detail/IrisInterface.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisUtils.h"
#include "iris/detail/IrisCommaSeparatedParameters.h"
#include "iris/impl/IrisChannelRegistry.h"
#include "iris/impl/IrisMessageQueue.h"
#include "iris/impl/IrisPlugin.h"
#include "iris/impl/IrisProcessEventsThread.h"
#include "iris/impl/IrisRpcAdapterTcp.h"
#include "iris/impl/IrisTcpSocket.h"
#include <map>
#include <memory>
#include <mutex>
#include <queue>
#include <thread>
#include <vector>

```

Classes

- class [iris::IrisClient](#)

Functions

- **NAMESPACE_IRIS_INTERNAL_START** (service) class IrisServiceTcpServer

9.5.1 Detailed Description

Iris client which supports multiple methods to connect to other Iris executables.

Date

Copyright ARM Limited 2015-2022 All Rights Reserved.

9.6 IrisClient.h

[Go to the documentation of this file.](#)

```

1
2
3
4
5
6
7 #ifndef ARM_INCLUDE_IrisClient_h
8 #define ARM_INCLUDE_IrisClient_h
9
10 #include "iris/IrisInstance.h"
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisErrorCode.h"
14 #include "iris/detail/IrisInterface.h"
15 #include "iris/detail/IrisLogger.h"
16 #include "iris/detail/IrisUtils.h"
17 #include "iris/detail/IrisCommaSeparatedParameters.h"
18
19 #include "iris/impl/IrisChannelRegistry.h"
20 #include "iris/impl/IrisMessageQueue.h"
21 #include "iris/impl/IrisPlugin.h"
22 #include "iris/impl/IrisProcessEventsThread.h"
23 #include "iris/impl/IrisRpcAdapterTcp.h"
24 #include "iris/impl/IrisTcpSocket.h"
25 #include "iris/IrisInstance.h"
26
27 #include <map>
28 #include <memory>
29 #include <mutex>
30 #include <queue>
31 #include <thread>
32 #include <vector>
33 #if defined(__linux__) || defined(__APPLE__)
34 #include <csignal>
35 #include <sys/types.h>
36 #include <sys/wait.h>
37 #endif
38 #if defined(__linux__)
39 #include <sys/prctl.h>
40 #endif
41
42 NAMESPACE_IRIS_INTERNAL_START(service)
43 class IrisServiceTcpServer;
44 NAMESPACE_IRIS_INTERNAL_END
45
46 NAMESPACE_IRIS_START
47
48 class IrisClient
49 : public IrisInterface
50 , public impl::IrisProcessEventsInterface
51 , public IrisConnectionInterface
52 {
53 public:
54     IrisClient(const std::string& instName = std::string(), const std::string& connectionSpec =
55         std::string())
56     {
57         init(IRIS_TCP_CLIENT, instName);
58         if (!connectionSpec.empty())
59         {
60             connect(connectionSpec);
61         }
62     }
63
64     IrisClient(const service::IrisServiceTcpServer*, const std::string& instName = std::string())
65     {
66 
```

```

67     init(IRIS_SERVICE_SERVER, instName);
68 }
69
80 IrisClient(const std::string& hostname, uint16_t port, const std::string& instName = std::string())
81 {
82     init(IRIS_TCP_CLIENT, instName);
83     std::string ignored_error;
84     IrisErrorCode status = connect(hostname, port, port ? 1000 : 100, ignored_error);
85     if (status != E_ok)
86     {
87         throw IrisErrorExceptionString(status, "Failed to connect to Iris TCP server");
88     }
89 }
90
92 virtual ~IrisClient()
93 {
94     disconnect();
95
96     // Do not rely on destructor order. The socket_thread expects this
97     // object to be fully alive.
98     if (socket_thread)
99     {
100         socket_thread->terminate();
101     }
102
103     switch (mode)
104     {
105     case IRIS_TCP_CLIENT:
106         socketSet.removeSocket(&sock);
107         break;
108
109     case IRIS_SERVICE_SERVER:
110         socketSet.removeSocket(service_socket);
111         // remove service_socket TODO safer memory management
112         delete service_socket;
113         break;
114     }
115
116     iris::sleepMs(sleepOnDestructionMs);
117 }
118
132 void connectCommandLine(const std::vector<std::string>& commandLine_, const std::string&
programName)
133 {
134     std::vector<std::string> commandLine = commandLine_;
135     connectCommandLineKeepOtherArgs(commandLine, programName);
136     if (!commandLine.empty())
137     {
138         throw IrisErrorExceptionString(E_not_connected, "connectCommandLine(): Unknown argument(s):
" + joinString(commandLine, ", ") + ".");
139     }
140 }
141
143 void connectCommandLineKeepOtherArgs(std::vector<std::string>& commandLine, const std::string&
programName)
144 {
145     // Parse client and server args.
146     IrisCommaSeparatedParameters clientArgs;
147     IrisCommaSeparatedParameters serverArgs;
148     std::vector<std::string> modelCommandLine;
149     for (size_t i = 0; i < commandLine.size(); i++)
150     {
151         if ((commandLine[i] == "--") && clientArgs.have("spawn"))
152         {
153             // Stop parsing args at "--".
154             // The model command line follows.
155             modelCommandLine.insert(modelCommandLine.begin(), commandLine.begin() + i + 1,
commandLine.end());
156             commandLine.resize(i);
157             break;
158         }
159
160         // Get key of key[=value] args.
161         std::string key = commandLine[i];
162         size_t pos = key.find('=');
163         if (pos != std::string::npos)
164         {
165             key = key.substr(0, pos);
166         }
167
168         // Set client args.
169         if ((key == "spawn") || (key == "tcp") || (key == "port") || (key == "timeout") || (key ==
"iris-log") || (key == "verbose") || (key == "help"))
170         {
171             clientArgs.set(commandLine[i]);
172             commandLine.erase(commandLine.begin() + i--);
173         }

```

```

174         // Set server args.
175         else if (key == "server_verbose")
176         {
177             serverArgs.set(commandLine[i].substr(7));
178             commandLine.erase(commandLine.begin() + i--);
179         }
180     }
181
182     // Just print help? Overrides everything else.
183     if (clientArgs.have("help"))
184     {
185         std::string help = getConnectCommandLineHelp();
186         replaceString(help, "%prog", programName);
187         throw IrisErrorExceptionString(E_help_message, help);
188     }
189
190     if (clientArgs.have("spawn"))
191     {
192         clientArgs.erase("spawn");
193
194         if (clientArgs.have("tcp"))
195         {
196             throw IrisErrorExceptionString(E_not_connected, "Only one out of \"spawn\" and \"tcp\"
may be specified.");
197         }
198
199         if (modelCommandLine.empty())
200         {
201             throw IrisErrorExceptionString(E_not_connected, "spawn: Missing/empty model command
line. Expected format: spawn -- isim_system -C foo=bar. Try 'help'.");
202         }
203
204         // Spawn child process and connect to it using UNIX domain socket.
205         spawnAndConnect(modelCommandLine, serverArgs.getParameterSpec(),
clientArgs.getParameterSpec());
206     }
207     else
208     {
209         // Connect via TCP. This is also the default if neither spawn not tcp are specified.
connect() needs an explicit "tcp" so set it here if not set.
210         if (!clientArgs.have("tcp"))
211         {
212             clientArgs.set("tcp");
213         }
214
215         if (!serverArgs.getMap().empty())
216         {
217             throw IrisErrorExceptionString(E_not_connected, "Server args cannot be set for
connections via \"tcp\". Specify server args on the model command line when starting the model.");
218         }
219
220         connect(clientArgs.getParameterSpec());
221     }
222 }
223
224 static std::string getConnectCommandLineHelp()
225 {
226     return
227         "Iris connection options:\n"
228         "  Spawn a model child process and connect to it using UNIX domain sockets:\n"
229         "    %prog [OPTIONS] spawn [timeout=TIMEOUT_IN_MS] [iris-log[=N]] [verbose[=0..3]]
[server_verbose[=0..3]] -- MODEL [MODEL_OPTIONS...]\n"
230         "\n"
231         "  Connect to an already running model process using TCP:\n"
232         "    %prog [OPTIONS] [tcp[=HOST]] [port=PORT] [timeout=TIMEOUT_IN_MS] [iris-log[=N]]
[verbose[=0..3]]\n"
233         "\n"
234         "  The arguments have the following semantics:\n"
235         "    spawn: Spawn a model child process and connect to it using UNIX domain sockets. The
model command line follows after '--'. \n"
236         "    tcp=HOST: (tcp only) Use TCP to connect to model process. Set hostname. Default is
localhost. \n"
237         "    port=N: (tcp only) Set Iris server port. Default is 0 if HOST is localhost, else 7100.
0 means scan ports 7100..7109. \n"
238         "    timeout=N: Set connection timeout to N ms. Default is 100 ms for \"tcp\" if PORT is 0,
else 1000 ms. Set this to 60000 ms when starting under gdb with \"set follow-fork-mode child\". \n"
239         "    iris-log[=N]: Log Iris functions calls (1=pretty, 2=JSON, 3=JSON-multiline, +8=U64JSON,
+16=time, +32=reltime). \n"
240         "    verbose=N: Set verbose level of IrisClient (0..3). \n"
241         "    server_verbose: (spawn only) Set verbose level of Iris server (0..3). (For \"tcp\" set
server verbose level on model command line when starting the model.) \n"
242         "    help: Print this connection option help message. \n"
243         "\n"
244         "  Example: Spawn model and connect to it:\n"
245         "    %prog spawn -- isim_system -C bp.secure_memory=false -a cluster0.cpu0=hello.axf \n"
246         "  Example: Same but also log Iris function calls and increase connection timeout to 60s
(useful when debugging model under gdb with \"set follow-fork-mode child\"): \n"

```

```

249         "    %prog spawn iris-log timeout=60000 -- isim_system -C bp.secure_memory=false -a
cluster0.cpu0=hello.axf\n"
250     "    Example: Connect to first model process found while scanning ports 7100..7109 on
localhost:\n"
251     "    %prog\n"
252     "    Example: Connect to model process on host 10.10.10.10 and port 7101:\n"
253     "    %prog tcp=10.10.10.10 port=7101\n"
254     "\n"
255     ;
256 }
257
262 void spawnAndConnect(const std::vector<std::string>& modelCommandLine, const std::string&
additionalServerArgs = std::string(), const std::string& additionalClientArgs = std::string())
263 {
264 #ifdef _WIN32
265     (void)modelCommandLine;
266     (void)additionalServerArgs;
267     (void)additionalClientArgs;
268     if (modelCommandLine.size() < 1000000) // Hack: Disable spurious "unreachable code" warning in
code calling spawnAndConnect() on Windows while we have not implemented this.
269     {
270         throw IrisErrorExceptionString(E_not_connected, "socketpair() connections not yet supported
on Windows");
271     }
272 #else
273     // Increase verbose level? (connect() below does this, but is too late)
274     IrisCommaSeparatedParameters clientArgs(additionalClientArgs, "1");
275     setVerbose(unsigned(clientArgs.getUint("verbose", 0)), /*increaseOnly=*/true);
276     setIrisMessageLogLevel(unsigned(clientArgs.getUint("iris-log", 0)), /*increaseOnly=*/true);
277     if (verbose)
278     {
279         log.info("IrisClient::spawnAndConnect(modelCommandLine=" + toString(modelCommandLine) + ",
additionalServerArgs=" + toStringToJson(additionalServerArgs) + ", additionalClientArgs=" +
toStringToJson(additionalClientArgs) + ")\n");
280     }
281
282     if (isConnected() || (childPid > 0))
283     {
284         disconnectAndWaitForChildToExit();
285     }
286
287     // Create socket pair.
288     int socketfd[2]; // We arbitrarily choose: 0=parent/client, 1=child/server
289     enum { CLIENT, SERVER };
290     if (socketpair(PF_LOCAL, SOCK_STREAM, 0, socketfd))
291     {
292         throw IrisErrorExceptionString(E_socket_error, "socketpair() failed");
293     }
294
295     lastExitStatus = -1;
296
297     // Fork.
298     childPid = fork();
299     if (childPid == 0)
300     {
301         // Child == server/model.
302         close(socketfd[CLIENT]);
303
304 #if defined(__linux__)
305         // Ask the kernel to kill us with SIGINT on parent thread termination.
306         // NOTE: Cleared on fork, but not on exec.
307         prctl(PR_SET_PDEATHSIG, SIGINT);
308 #endif
309
310         // Prepare args.
311         std::vector<std::string> args = modelCommandLine;
312         args.push_back("--iris-connect");
313         args.push_back("socketfd=" + std::to_string(socketfd[SERVER]) + "," + additionalServerArgs);
314         std::vector<const char*> cargs;
315         for (const std::string& s: args)
316         {
317             cargs.push_back(s.c_str());
318         }
319         cargs.push_back(nullptr);
320
321         // Start model. Replaces the currently running executable. Does not return on success.
322         execve(cargs[0], (char * const *)cargs.data(), environ);
323
324         // execve() only returns on error.
325         close(socketfd[SERVER]);
326         throw IrisErrorExceptionString(E_not_connected, "execve() failed. Error launching model
(command line: " + iris::joinString(args, " ") + ").");
327     }
328     else if (childPid < 0)
329     {
330         close(socketfd[CLIENT]);
331         close(socketfd[SERVER]);

```

```

332         childPid = 0;
333         throw IrisErrorExceptionString(E_not_connected, "fork() failed with errno=" +
std::to_string(errno) + ".");
334     }
335     else
336     {
337         if (verbose)
338         {
339             log.info("IrisClient::spawnAndConnect(): Spawned child process %d.\n", int(childPid));
340         }
341
342         // Parent == client/debugger.
343         close(socketfd[SERVER]);
344
345         try
346         {
347             // Connect to model.
348             connect("socketfd=" + std::to_string(socketfd[CLIENT]) + "," + additionalClientArgs);
349         }
350         catch (...)
351         {
352             // connect() already closed the socket on error.
353
354             // Issue SIGINT and then SIGKILL to terminate child.
355             disconnectAndWaitForChildToExit(0);
356             throw;
357         }
358     }
359 #endif
360 }
361
362 bool disconnectAndWaitForChildToExit(double timeoutInMs = 5000, double timeoutInMsAfterSigInt =
5000, double timeoutInMsAfterSigKill = 5000)
363 {
364     if (verbose)
365     {
366         log.info("IrisClient::disconnectAndWaitForChildToExit(timeoutInMs=%.0f,
timeoutInMsAfterSigInt=%.0f, timeoutInMsAfterSigKill=%.0f)\n", timeoutInMs, timeoutInMsAfterSigInt,
timeoutInMsAfterSigKill);
367     }
368
369     // Disconnect.
370     IrisErrorCode error = disconnect();
371     if (error)
372     {
373         throw IrisErrorExceptionString(E_not_connected, "disconnect() failed.");
374     }
375
376 #ifdef _WIN32
377     (void)timeoutInMs;
378     (void)timeoutInMsAfterSigInt;
379     (void)timeoutInMsAfterSigKill;
380     throw IrisErrorExceptionString(E_not_implemented, "socketpair() connections not yet supported on
Windows.");
381 #else
382     if (childPid == 0)
383     {
384         return true;
385     }
386
387     if (!floatEqual(timeoutInMs, 0.0))
388     {
389         // Wait for child process to exit for timeoutInMs.
390         if (waitpidWithTimeout(childPid, &lastExitStatus, 0, timeoutInMs))
391         {
392             childPid = 0;
393             return true;
394         }
395     }
396
397     if (!floatEqual(timeoutInMsAfterSigInt, 0.0))
398     {
399         // Send SIGINT and wait for timeoutInMsAfterSigInt.
400         if (verbose)
401         {
402             log.info("IrisClient::disconnectAndWaitForChildToExit(): Sending SIGINT to child %d.\n",
int(childPid));
403         }
404         if (kill(childPid, SIGINT) < 0)
405         {
406             throw IrisErrorExceptionString(E_not_connected, "kill(SIGINT) failed with errno=" +
std::to_string(errno) + ".");
407         }
408         if (waitpidWithTimeout(childPid, &lastExitStatus, 0, timeoutInMsAfterSigInt))
409         {
410             childPid = 0;
411             return true;
412         }
413     }
414 }

```



```

425     }
426 }
427
428     if (!floatEqual(timeoutInMsAfterSigKill, 0.0))
429     {
430         // Send SIGKILL and wait for timeoutInMsAfterSigKill.
431         if (verbose)
432         {
433             log.info("IrisClient::disconnectAndWaitForChildToExit(): Sending SIGKILL to child
434 %d.\n", int(childPid));
435             if (kill(childPid, SIGKILL) < 0)
436             {
437                 throw IrisErrorExceptionString(E_not_connected, "kill(SIGKILL) failed with errno=" +
438 std::to_string(errno) + ".");
439             }
440             if (waitpidWithTimeout(childPid, &lastExitStatus, 0, timeoutInMsAfterSigKill))
441             {
442                 childPid = 0;
443                 return true;
444             }
445             // Child did not exit so far.
446             if (verbose)
447             {
448                 log.info("IrisClient::disconnectAndWaitForChildToExit(): Child %d did not exit.\n",
449 int(childPid));
450             }
451             return false;
452 #endif
453     }
454
455 #ifndef _WIN32
456     bool waitpidWithTimeout(pid_t pid, int* status, int options, double timeoutInMs)
457     {
458         if (verbose)
459         {
460             log.info("IrisClient::waitpidWithTimeout(): Waiting %.1f ms for child %d to exit ...\n",
461 timeoutInMs, int(pid));
462         }
463
464         double endTime = getTimeInSec() + timeoutInMs / 1000.0;
465         if (timeoutInMs < 0)
466         {
467             endTime += 1e100;
468         }
469
470         // Wait for child to exit.
471         while (getTimeInSec() < endTime)
472         {
473             pid_t ret = waitpid(pid, status, options | WNOHANG);
474             if (ret == pid)
475             {
476                 if (verbose)
477                 {
478                     log.info("IrisClient::waitpidWithTimeout(): Child %d exited with exit status %d
479 after waiting for %.3fs.\n", int(pid), status ? *status : 0, getTimeInSec() - endTime + (timeoutInMs
480 / 1000.0));
481                 }
482                 return true; // Child exited.
483             }
484             if (ret < 0)
485             {
486                 throw IrisErrorExceptionString(E_not_connected, "waitpid() failed with errno=" +
487 std::to_string(errno) + ".");
488             }
489             if (ret > 0)
490             {
491                 throw IrisErrorExceptionString(E_not_connected, "waitpid() returned unexpected pid=" +
492 std::to_string(pid) + ".");
493             }
494             assert(ret == 0);
495             sleepMs(20);
496         }
497         return false; // Timeout.
498     }
499 #endif
500
501 #ifndef _WIN32
502     pid_t getChildPid() const
503     {
504         return childPid;
505     }
506 #endif
507 #endif

```

```

508
509 int getLastExitStatus() const { return lastExitStatus; }
510
511
512 const std::string connectionHelpStr =
513     "Supported connection types:\n"
514     "tcp[=HOST][,port=PORT][,timeout=T]\n"
515     "    Connect to an Iris TCP server on HOST:PORT.\n"
516     "    The default for HOST is 'localhost' and the default for PORT is 0 if HOST is 'localhost' and
7100 otherwise. If PORT is 0 then a port scan on ports 7100 to 7109 is done.\n"
517     "    T is the connection timeout in ms (defaults to 100 if PORT==0, else 1000).\n"
518     "\n"
519     "socketfd=FD[,timeout=T]\n"
520     "    Use socket file descriptor FD as an established UNIX domain socket connection.\n"
521     "    T is the timeout for the Iris handshake in ms.\n"
522     "\n"
523     "General parameters:\n"
524     "    verbose[=N]: Increase verbose level of IrisClient to level N (0..3).\n"
525     "    iris-log[=N]: Log Iris functions calls (1=pretty, 2=JSON, 3=JSON-multiline, +8=U64JSON,
526     +16=time, +32=reltime).\n";
527
528 void connect(const std::string& connectionSpec)
529 {
530     IrisCommaSeparatedParameters params(connectionSpec, "1");
531
532     // Emit help message?
533     if (params.have("help"))
534     {
535         throw IrisErrorExceptionString(E_help_message, connectionHelpStr);
536     }
537
538     // Increase verbose level?
539     setVerbose(unsigned(params.getUint("verbose", 0)), /*increaseOnly=*/true);
540     setIrisMessageLogLevel(unsigned(params.getUint("iris-log", 0)), /*increaseOnly=*/true);
541     if (verbose)
542     {
543         log.info("IrisClient::connect(connectionSpec=" + quoteStringToJson(connectionSpec) + ")\n");
544     }
545
546     // Validate connection type.
547     if (unsigned(params.have("tcp")) + unsigned(params.have("socketfd")) != 1)
548     {
549         throw IrisErrorExceptionString(E_not_connected, "Exactly one out of \"tcp\", \"socketfd\"
and \"help\" must be specified (got \"" + connectionSpec + "\"). Specify \"help\" to get a list of
all supported connection types.");
550     }
551
552     if (params.have("tcp"))
553     {
554         std::string hostname = params.getStr("tcp");
555         if (hostname == "1")
556         {
557             hostname = "localhost";
558         }
559         uint16_t port = uint16_t(params.getUint("port", hostname == "localhost" ? 0 : 7100));
560         unsigned timeoutInMs = unsigned(params.getUint("timeout", port == 0 ? 100 : 1000));
561         if (params.haveUnusedParameters())
562         {
563             throw IrisErrorExceptionString(E_not_connected, params.getUnusedParametersMessage("Error
in 'tcp' connection parameters: "));
564         }
565         std::string errorResponse;
566         IrisErrorCode status = connect(hostname, port, timeoutInMs, errorResponse);
567         if (status != E_ok)
568         {
569             throw IrisErrorExceptionString(status, errorResponse);
570         }
571     }
572
573     if (params.have("socketfd"))
574     {
575         SocketFd socketfd = SocketFd(params.getUint("socketfd"));
576         unsigned timeoutInMs = unsigned(params.getUint("timeout", 1000));
577         if (params.haveUnusedParameters())
578         {
579             throw IrisErrorExceptionString(E_not_connected, params.getUnusedParametersMessage("Error
in 'socketfd' connection parameters: "));
580         }
581         connectSocketFd(socketfd, timeoutInMs);
582     }
583
584     IrisErrorCode connect(const std::string& hostname, uint16_t port, unsigned timeoutInMs, std::string&
errorResponseOut)
585     {
586         assert(mode == IRIS_TCP_CLIENT);
587
588         if (verbose)

```

```

602         log.info("IrisClient::connect(hostname=%s, port=%u, timeout=%u) enter\n", hostname.c_str(),
port, timeoutInMs);
603
604         // Already connected?
605         IrisErrorCode error = E_ok;
606         if (adapter.isConnected() || sock.isConnected())
607         {
608             error = E_already_connected;
609             goto done;
610         }
611
612         // hostname==localhost and port==0 means port scan.
613         if ((hostname == "localhost") && (port == 0))
614         {
615             const uint16_t startport = 7100;
616             const uint16_t endport = 7109;
617             for (port = startport; port <= endport; port++)
618             {
619                 std::string errorMessage;
620                 if (connect(hostname, port, timeoutInMs, errorResponseOut) == iris::E_ok)
621                     return E_ok;
622             }
623             errorResponseOut = "No Iris TCP server found on ports " + std::to_string(startport) + ".." +
std::to_string(endport) + "\n";
624             error = E_not_connected;
625             goto done;
626         }
627
628         if (!sock.isCreated())
629         {
630             sock.create();
631             sock.setNonBlocking();
632
633             // Unblock a potentially blocked worker thread which so far is waiting indefinitely
634             // on 'no socket'. This thread will block again on the socket we just created.
635             socketSet.stopWaitForEvent();
636         }
637
638         // Connect to server.
639         error = sock.connect(hostname, port, timeoutInMs);
640         if (error != E_ok)
641         {
642             errorResponseOut = "Error connecting to " + hostname + ":" + std::to_string(port);
643             sock.close();
644             goto done;
645         }
646
647         // Initialize client.
648         error = initClient(timeoutInMs, errorResponseOut);
649         if (error == E_ok)
650         {
651             connectionStr = hostname + ":" + std::to_string(port);
652         }
653         else
654         {
655             disconnect();
656         }
657
658         // Return error code (if any).
659     done:
660         if (verbose)
661             log.info("IrisClient::connect() leave (%s)\n", irisErrorCodeCStr(error));
662         return error;
663     }
664
665     void connectSocketFd(SocketFd socketfd, unsigned timeoutInMs = 1000)
666     {
667         assert(mode == IRIS_TCP_CLIENT);
668
669         if (verbose)
670             log.info("IrisClient::connectSocketFd(socketfd=%llu, timeout=%u)\n", (long long)socketfd,
timeoutInMs);
671
672         // Already connected?
673         std::string errorResponse;
674         IrisErrorCode error = E_ok;
675         if (adapter.isConnected() || sock.isConnected())
676         {
677             throw IrisErrorExceptionString(E_already_connected, "Already connected.");
678         }
679
680         sock.setSocketFd(socketfd);
681         sock.setNonBlocking();
682
683         // Unblock a potentially blocked worker thread which so far is waiting indefinitely
684         // on 'no socket'. This thread will block again on the socket we just created.
685         socketSet.stopWaitForEvent();

```

```

689
690     // Initialize client.
691     error = initClient(timeoutInMs, errorResponse);
692     if (error != E_ok)
693     {
694         disconnect();
695         throw IrisErrorExceptionString(error, errorResponse);
696     }
697
698     connectionStr = "(connected via sockfd)";
699 }
700
701 IrisErrorCode disconnect()
702 {
703     if (verbose)
704     {
705         log.info("IrisClient::disconnect()\n");
706     }
707
708     // Tell IrisInstance to stop sending requests to us.
709     // All Iris calls (including the inevitable final
710     // instanceRegistry_unregisterInstance()) will return
711     // E_not_connected from now on.
712     irisInstance.setConnectionInterface(nullptr);
713
714     connectionStr = "(not connected)";
715
716     if (mode != IRIS_TCP_CLIENT)
717     {
718         return E_ok;
719     }
720
721     // We just close the TCP connection. This is a first-class operation which always must be
722     // handled gracefully by the server.
723     // The server needs to do all cleanup automatically.
724     IrisErrorCode errorCode = E_ok;
725     if (adapter.isConnected())
726         errorCode = adapter.closeConnection();
727     if (sock.isConnected())
728     {
729         if (errorCode != E_ok)
730             sock.close();
731         else
732             errorCode = sock.close();
733     }
734
735     // Wake up processing thread since there is no point to wait on a closed socket.
736     socketSet.stopWaitForEvent();
737
738     return errorCode;
739 }
740
741 bool isConnected() const
742 {
743     return adapter.isConnected();
744 }
745
746 IrisInterface* getSendingInterface()
747 {
748     return this;
749 }
750
751 void setInstanceName(const std::string& instName)
752 {
753     if (irisInstance.isRegistered())
754     {
755         throw IrisErrorExceptionString(E_instance_already_registered, "IrisClient::setInstanceName()
756 must be called before connect().");
757     }
758     irisInstanceInstName = instName;
759 }
760
761 IrisInstance& getIrisInstance() { return irisInstance; }
762
763 void setSleepOnDestructionMs(uint64_t sleepOnDestructionMs_)
764 {
765     sleepOnDestructionMs = sleepOnDestructionMs_;
766 }
767
768 // --- IrisProcessEventsInterface implementation ---
769
770 virtual void processEvents() override
771 {
772     if (verbose >= 2)
773         log.info("IrisClient::processEvents() enter\n");
774 }

```

```

801         // in IRIS_SERVICE_SERVER mode, the adapter should work as server and hence call
802         // function processEventsServer()
803         switch (mode)
804         {
805         case IRIS_TCP_CLIENT:
806             adapter.processEventsClient();
807             break;
808         case IRIS_SERVICE_SERVER:
809             adapter.processEventsServer();
810             break;
811         }
812
813         if (verbose >= 2)
814             log.info("IrisClient::processEvents() leave\n");
815     }
816
817     virtual void waitForEvent() override
818     {
819         if (verbose >= 2)
820             log.info("IrisClient::waitForEvent() enter\n");
821         socketSet.waitForEvent(1000);
822         if (verbose >= 2)
823             log.info("IrisClient::waitForEvent() leave\n");
824     }
825
826     virtual void stopWaitForEvent() override
827     {
828         if (verbose)
829             log.info("IrisClient::stopWaitForEvent()\n");
830         socketSet.stopWaitForEvent();
831     }
832
833     void setPreferredSendingFormat(impl::IrisRpcAdapterTcp::Format p)
834     {
835         adapter.setPreferredSendingFormat(p);
836     }
837
838     impl::IrisRpcAdapterTcp::Format getEffectiveSendingFormat() const
839     {
840         return adapter.getEffectiveSendingFormat();
841     }
842
843     void setVerbose(unsigned level, bool increaseOnly = false)
844     {
845         if (increaseOnly && (level <= verbose))
846         {
847             return;
848         }
849
850         verbose = level;
851         if (verbose)
852             log.info("IrisClient: verbose logging enabled (level %d)\n", verbose);
853         if (mode == IRIS_TCP_CLIENT)
854         {
855             sock.setVerbose(verbose);
856         }
857         socketSet.setVerbose(verbose);
858         if (verbose)
859         {
860             log.setIrisMessageLogLevelFlags(IrisLogger::TIMESTAMP);
861         }
862     }
863
864     void setIrisMessageLogLevel(unsigned level, bool increaseOnly = false)
865     {
866         if (increaseOnly && (level <= irisMessageLogLevel))
867         {
868             return;
869         }
870
871         irisMessageLogLevel = level;
872         log.setIrisMessageLogLevel(irisMessageLogLevel);
873     }
874
875     std::string getConnectionStr() const { return connectionStr; }
876
877 private:
878     enum Mode
879     {
880         IRIS_TCP_CLIENT,
881         IRIS_SERVICE_SERVER
882     };
883
884     // Shared code for constructors in client mode.
885     void init(Mode mode_, const std::string& instName)
886     {
887         log.setLogContext("IrisTC");
888     }

```

```

899     mode = mode_;
900
901     // Set instance name of contained IrisInstance.
902     if (instName.empty())
903     {
904         setInstanceName("client.IrisClient");
905     }
906     else
907     {
908         setInstanceName(instName);
909     }
910
911     // Enable verbose logging?
912     setVerbose(static_cast<unsigned>(getEnvU64("IRIS_TCP_CLIENT_VERBOSE")), true);
913     irisMessageLogLevel = unsigned(getEnvU64("IRIS_TCP_CLIENT_LOG_MESSAGES"));
914     log.setIrisMessageLogLevel(irisMessageLogLevel);
915     log.setIrisMessageGetInstNameFunc([&](InstanceId instId) { return getInstName(instId); });
916
917     if (mode == IRIS_TCP_CLIENT)
918     {
919         socketSet.addSocket(&sock);
920     }
921     sendingInterface = adapter.getSendingInterface();
922
923     // Intercept all calls to the global instance since we must modify
924     // instanceRegistry_unregisterInstance() and their responses.
925     instIdToInterface.push_back(&globalInstanceSendingInterface); // This must be index 0 in the
926     // vector (instId 0 == global instance).
927
928     if (mode == IRIS_SERVICE_SERVER)
929     {
930         socket_thread = std::unique_ptr<impl::IrisProcessEventsThread>(new
931         impl::IrisProcessEventsThread(this, "TcpSocket"));
932     }
933
934     IrisErrorCode initClient(unsigned timeoutInMs, std::string& errorResponseOut)
935     {
936         assert(mode == IRIS_TCP_CLIENT);
937
938         // Initialize IrisRpcAdapterTcp.
939         try
940         {
941             adapter.initClient(&sock, &socketSet, &receivingInterface, verbose);
942         }
943         catch (const IrisErrorException& e)
944         {
945             if (e.getMessage().empty())
946             {
947                 throw IrisErrorExceptionString(e.getErrorCode(), "Client: Error connecting to server
948 socket.");
949             }
950             else
951             {
952                 throw;
953             }
954         }
955
956         // Handshake.
957         IrisErrorCode error = adapter.handshakeClient(errorResponseOut, timeoutInMs);
958
959         // Start a thread to process incoming data in the background.
960         socket_thread = std::unique_ptr<impl::IrisProcessEventsThread>(new
961         impl::IrisProcessEventsThread(this, "TcpSocket"));
962
963         // Initialize IrisInstance.
964         irisInstance.setConnectionInterface(this);
965         irisInstance.registerInstance(irisInstanceInstName, iris::IrisInstance::UNIQUEIFY |
966         iris::IrisInstance::THROW_ON_ERROR);
967
968         return error;
969     }
970
971     virtual void irisHandleMessage(const uint64_t* message) override
972     {
973         // Log message?
974         if (irisMessageLogLevel)
975         {
976             log.irisMessage(message);
977         }
978
979         // This calls one of these:
980         // - this->globalInstanceSendingInterface_irisHandleMessage(); (for requests, instId == 0)
981         // - Iris interface of a local instance (if a local instance talks to a local instance)
982         // - sendingInterface (to send message to server using TCP)
983         findInterface(IrisU64JsonReader::getInstId(message))->irisHandleMessage(message);

```

```

984     }
985
986 void globalInstanceSendingInterface_irisHandleMessage(const uint64_t* message)
987 {
988     // This is only ever called for instId == 0.
989     assert(IrisU64JsonReader::getInstId(message) == 0);
990     assert(IrisU64JsonReader::isRequestOrNotification(message));
991
992     // Decode request.
993     IrisU64JsonReader r(message);
994     IrisU64JsonReader::Request req = r.openRequest();
995     std::string method = req.getMethod();
996
997     if (method == "instanceRegistry_registerInstance")
998     {
999         RequestId requestId = req.getRequestId();
1000
1001         // We received an instanceRegistry_registerInstance() request from a local instance:
1002         // - Create a new request id which is unique to this request for this TCP channel. (This is
1003         // not required to be globally unique.)
1004         // - Allocate an ongoingInstanceRegistryCalls slot for this new request id and remember the
1005         // original request id and params.channelId in it.
1006         // - Modify request id of request to the new request id so we can recognize the response
1007         // later.
1008         // - Send modified request.
1009
1010         // Create a new request id which is unique to this request for this TCP channel. (This is
1011         // not required to be globally unique.)
1012         RequestId newRequestId = generateNewRequestIdForRegisterInstanceCall();
1013
1014         // Get channelId.
1015         uint64_t channelId = IRIS_UINT64_MAX;
1016         if (!req.paramOptional(ISTR("channelId"), channelId))
1017         {
1018             // Strange. 'params.channelId' is missing. This should never happen.
1019             log.error(
1020                 "IrisClient::receivingInterface_irisHandleMessage(): "
1021                 "Received instanceRegistry_registerInstance() request without channelId
1022                 parameter:\n%s\n",
1023                 messageToString(message).c_str());
1024             goto send;
1025         }
1026
1027         {
1028             std::lock_guard<std::mutex> lock(ongoingInstanceRegistryCallsMutex);
1029             // Allocate an ongoingInstanceRegistryCalls slot for this new request id and remember
1030             // the
1031             // original request id and params.channelId in it.
1032             ongoingInstanceRegistryCalls[newRequestId] = OngoingInstanceRegistryCallEntry(method,
1033             requestId,
1034             channelId);
1035         }
1036
1037         // Create a modified request that:
1038         // - sets the new request id so we can recognize the response later.
1039         // - removes the channelId parameter (it only has meaning in-process)
1040         IrisU64JsonReader original_message(message);
1041         IrisU64JsonWriter modified_message;
1042
1043         {
1044             IrisU64JsonReader::Request original_req = original_message.openRequest();
1045
1046             IrisU64JsonWriter::Request new_req =
1047             modified_message.openRequest(original_req.getMethod(),
1048             original_req.getInstId());
1049             new_req.setRequestId(newRequestId);
1050
1051             std::string param;
1052             while (original_req.readNextParam(param))
1053             {
1054                 if ((param == "channelId") || (param == "instId"))
1055                 {
1056                     // Skip the params we want to remove (channelId)
1057                     // and skip instId too because that will have already been filled in.
1058                     // skip over the value to the next parameter
1059                     original_message.skip();
1060                 }
1061                 else
1062                 {
1063                     new_req.paramSlow(param);
1064
1065                     // Pass through the original value
1066                     IrisValue value;
1067                     persist(original_message, value);

```

```

1063         persist(modified_message, value);
1064     }
1065 }
1066
1067 // Send modified request.
1068 sendingInterface->irisHandleMessage(modified_message.getMessage());
1069 return;
1070 }
1071 else if (method == "instanceRegistry_unregisterInstance")
1072 {
1073     // We received an instanceRegistry_unregisterInstance() request from a local instance:
1074     // - Allocate an ongoingInstanceRegistryCalls slot for the request id and remember the
1075     instId of the unregistered instance in it.
1076     // - Send request unmodified.
1077
1078     // Get params.aInstId.
1079     InstanceId aInstId = IRIS_UINT64_MAX;
1080     if (!req.paramOptional(ISTR("aInstId"), aInstId))
1081     {
1082         // Strange. 'params.aInstId' is missing. This should never happen.
1083         log.error(
1084             "IrisClient::receivingInterface_irisHandleMessage(): "
1085             " Received instanceRegistry_unregisterInstance() request without aInstId
1086 parameter:\n%s\n",
1087             messageToString(message).c_str());
1088         goto send;
1089     }
1090     if (!req.isNotification())
1091     {
1092         RequestId requestId = req.getRequestId();
1093         if (aInstId == getCallerInstId(requestId))
1094         {
1095             std::lock_guard<std::mutex> lock(ongoingInstanceRegistryCallsMutex);
1096             // There will be a response to this request so we need to remember the interface to
1097             send it to.
1098             // Allocate an ongoingInstanceRegistryCalls slot for the request id and remember
1099             the instId of the unregistered instance in it.
1100             ongoingInstanceRegistryCalls[requestId] = OngoingInstanceRegistryCallEntry(method,
1101             aInstId);
1102             goto send;
1103         }
1104         // There will be no more communication to the instance being unregistered.
1105         // Remove instance from instIdToInterface.
1106         assert(aInstId < InstanceId(instIdToInterface.size()));
1107         // sendingInterface: Forward messages to unknown instIds to the server. The global instance
1108         may have reassigned the same instId to some other instance behind the server which exists.
1109         instIdToInterface[aInstId] = sendingInterface;
1110         // Intended fallthrough to send original request.
1111     }
1112     else if (method == "instanceRegistry_getList")
1113     {
1114         // We received an instanceRegistry_getList() request from a local instance:
1115         // - We want to remember/snoop all returned instance names we get in the response (for
1116         logging).
1117         // - Allocate an ongoingInstanceRegistryCalls slot for the request id in order to recognize
1118         the response.
1119         // - Send request unmodified.
1120         if (!req.isNotification())
1121         {
1122             RequestId requestId = req.getRequestId();
1123             std::lock_guard<std::mutex> lock(ongoingInstanceRegistryCallsMutex);
1124             ongoingInstanceRegistryCalls[requestId] = OngoingInstanceRegistryCallEntry(method);
1125         }
1126         // Intended fallthrough to send original request.
1127     }
1128 }
1129
1130 send:
1131     // Send original message.
1132     sendingInterface->irisHandleMessage(message);
1133 }
1134
1135 void receivingInterface_irisHandleResponse(const uint64_t* message)
1136 {
1137     {
1138         std::lock_guard<std::mutex> lock(ongoingInstanceRegistryCallsMutex);
1139         if (!ongoingInstanceRegistryCalls.empty())
1140         {
1141             // Slow path is only used while a instanceRegistry_registerInstance() or

```



```

instanceRegistry_unregisterInstance()
1145     // call is ongoing. This is usually only the case at startup and shutdown.
1146
1147     // We need to check whether this is the response to either
1148     // instanceRegistry_registerInstance() or
1149     // instanceRegistry_unregisterInstance() or
1150     // any other response.
1151
1152     // Decode response.
1153     IrisU64JsonReader      r(message);
1154     IrisU64JsonReader::Response resp = r.openResponse();
1155     RequestId requestId = resp.getRequestId();
1156
1157     // Check whether this is a response to one of our pending requests.
1158     OngoingInstanceRegistryCallMap::iterator i =
ongoingInstanceRegistryCalls.find(requestId);
1159     if (i == ongoingInstanceRegistryCalls.end())
1160     {
1161         goto send; // None of the pending responses. Handle in the normal way.
1162     }
1163
1164     if (i->second.method == "instanceRegistry_registerInstance")
1165     {
1166         // This is a response to a previous instanceRegistry_registerInstance() call:
1167
1168         IrisInterface* responseIfPtr = channel_registry.getChannel(i->second.channelId);
1169
1170         if (resp.isError())
1171         {
1172             // The call failed, pass on the message.
1173             responseIfPtr->irisHandleMessage(message);
1174         }
1175         else
1176         {
1177             // The call succeeded:
1178             // - add new instId to our local instance registry
1179             // - translate request id back to the original request id
1180             // - send this modified response to the caller
1181             // - erase this entry in ongoingInstanceRegistryCalls
1182
1183             // Add instance to instIdToInterface.
1184             InstanceId newInstId;
1185             if (!resp.getResultReader().openObject().memberOptional(ISTR("instId"),
newInstId))
1186             {
1187                 // Strange. 'result.instId' is missing. This should never happen.
1188                 log.error(
1189                     "IrisClient::receivingInterface_irisHandleResponse(): "
1190                     "Received instanceRegistry_registerInstance() response without
result.instId:\n%s\n",
1191                     messageToString(message).c_str());
1192             }
1193             else
1194             {
1195                 // This is a valid response for instanceRegistry_registerInstance(): Enter
newInstId into instIdToInterface.
1196                 findInterface(newInstId);
1197                 instIdToInterface[newInstId] = responseIfPtr;
1198             }
1199
1200             // Remember instance name.
1201             std::string newInstName;
1202             if (resp.getResultReader().openObject().memberOptional(ISTR("instName"),
newInstName))
1203             {
1204                 setInstName(newInstId, newInstName);
1205             }
1206
1207             // Translate the id back to the id of the original request and use the
responseIfPtr to send the response.
1208             IrisU64JsonWriter modifiedMessageWriter;
1209             modifiedMessageWriter.copyMessageAndModifyId(message, i->second.id);
1210
1211             // Log message?
1212             if (irisMessageLogLevel)
1213             {
1214                 log.irisMessage(modifiedMessageWriter.getMessage());
1215             }
1216
1217             responseIfPtr->irisHandleMessage(modifiedMessageWriter.getMessage());
1218         }
1219
1220         // Remove ongoingInstanceRegistryCalls entry now that we have seen the response.
1221         ongoingInstanceRegistryCalls.erase(i);
1222         return;
1223     }
1224     else if (i->second.method == "instanceRegistry_unregisterInstance")

```

```

1225         {
1226             // This is a response to a previous instanceRegistry_unregisterInstance() call:
1227             // - remove this instId from our local instance registry
1228             // - remove this entry from ongoingInstanceRegistryCalls
1229             // - send response to caller
1230
1231             InstanceId aInstId = i->second.id;
1232
1233             // Remeber the old response interface in case we need it after we override it
1234             IrisInterface* aInst_responseIf = instIdToInterface[aInstId];
1235
1236             // Remove instance from instIdToInterface.
1237             assert(aInstId < InstanceId(instIdToInterface.size()));
1238             // sendingInterface: Forward messages to unknown instIds to the server. The global
instance may have reassigned the same instId to some other instance behind the server which exists.
1239             instIdToInterface[aInstId] = sendingInterface;
1240             setInstName(aInstId, ""); // IrisLogger will generate a default name for unknown
instance ids.
1241
1242             // Remove ongoingInstanceRegistryCalls entry.
1243             ongoingInstanceRegistryCalls.erase(i);
1244
1245             if (aInstId == resp.getInstId())
1246             {
1247                 // An instance unregistered itself so we need to call it directly rather than
1248                 // go through the normal message handler because we just set that to forward
1249                 // messages to this instId to the server.
1250                 aInst_responseIf->irisHandleMessage(message);
1251                 return;
1252             }
1253             // Intended fallthrough to irisHandleMessage(message).
1254         }
1255         else if (i->second.method == "instanceRegistry_getList")
1256         {
1257             // This is a response to a previous instanceRegistry_getList() call:
1258             // - remember all instance names (for logging)
1259             // - send response to caller
1260
1261             // Remove ongoingInstanceRegistryCalls entry.
1262             ongoingInstanceRegistryCalls.erase(i);
1263             try
1264             {
1265                 // Peek into instance list. We do not care whether this is just
1266                 // a subset of all instances or not. We take what we can get.
1267                 std::vector<InstanceInfo> instanceInfoList;
1268                 resp.getResult(instanceInfoList);
1269                 for (const auto& instanceInfo: instanceInfoList)
1270                 {
1271                     setInstName(instanceInfo.instId, instanceInfo.instName);
1272                 }
1273             }
1274             catch(const IrisErrorException&)
1275             {
1276                 // Silently ignore bogus responses. The caller will handle the error.
1277             }
1278             // Intended fallthrough to irisHandleMessage(message).
1279         }
1280     }
1281 }
1282
1283 send:
1284     // Handle response in the normal way.
1285     irisHandleMessage(message);
1286 }
1287
1288 RequestId generateNewRequestIdForRegisterInstanceCall()
1289 {
1290     return nextInstIdForRegisterInstanceCall++;
1291 }
1292
1293 IrisInterface* findInterface(InstanceId instId)
1294 {
1295     if (instId >= IrisMaxTotalInstances)
1296     {
1297         log.error("IrisClient::findInterface(instId=0x%08x): got ridiculously high instId",
1298             int(instId));
1299         return sendingInterface;
1300     }
1301     if (instId >= InstanceId(instIdToInterface.size()))
1302     {
1303         instIdToInterface.resize(instId + 100, sendingInterface);
1304     }
1305     return instIdToInterface[instId];
1306 }
1307
1308 class GlobalInstanceSendingInterface : public IrisInterface
1309 {

```

```

1319     public:
1320         GlobalInstanceSendingInterface(IrisClient* parent_)
1321             : parent(parent_)
1322         {
1323         }
1324
1328         virtual void irisHandleMessage(const uint64_t* message) override
1329         {
1330             if (IrisU64JsonReader::isRequestOrNotification(message))
1331             {
1332                 // Intercept requests to the global instance so we can snoop on
1333                 // calls to instanceRegistry_registerInstance()
1334                 parent->globalInstanceSendingInterface_irisHandleMessage(message);
1335             }
1336             else
1337             {
1338                 // This is called for responses sent from clients to the global instance.
1339                 // Simply forward them as usual. Nothing to intercept.
1340                 parent->sendingInterface->irisHandleMessage(message);
1341             }
1342         }
1343
1344     private:
1345         IrisClient* const parent;
1346 };
1347
1349 class ReceivingInterface : public IrisInterface
1350 {
1351     public:
1352         ReceivingInterface(IrisLogger& log_, IrisClient* parent_)
1353             : parent(parent_)
1354             , log(log_)
1355         {
1356         }
1357
1359         virtual void irisHandleMessage(const uint64_t* message) override
1360         {
1361             InstanceId instId = IrisU64JsonReader::getInstId(message);
1362
1363             if (instId >= InstanceId(instId_to_thread_id.size()))
1364             {
1365                 // We do not have an entry for this instance therefore
1366                 // we have not been asked to marshal requests to a specific
1367                 // thread and should use the default.
1368                 // Todo: Remove once IrisMessageQueue and IrisProcessEventsThread are gone
1369                 setHandlerThread(instId, getDefaultThreadId());
1370             }
1371
1372             // Todo: Refactor once IrisMessageQueue and IrisProcessEventsThread are gone
1373             std::thread::id thread_id = instId_to_thread_id[instId];
1374             if (thread_id == std::this_thread::get_id())
1375             {
1376                 // Message has already been marshalled, forward on
1377                 if (IrisU64JsonReader::isRequestOrNotification(message))
1378                 {
1379                     parent->irisHandleMessage(message);
1380                 }
1381                 else
1382                 {
1383                     parent->receivingInterface_irisHandleResponse(message);
1384                 }
1385             }
1386             else
1387             {
1388                 message_queue.push(message, thread_id);
1389             }
1390         }
1391
1392         void setHandlerThread(InstanceId instId, std::thread::id thread_id)
1393         {
1394             if (instId >= IrisMaxTotalInstances)
1395             {
1396                 log.error(
1397                     "IrisClient::ReceivingInterface::setHandlerThread(instId=0x%08x) : "
1398                     "got ridiculously high instId",
1399                     int(instId));
1400             }
1401             else if (instId >= InstanceId(instId_to_thread_id.size()))
1402             {
1403                 instId_to_thread_id.resize(instId + 100, getDefaultThreadId());
1404             }
1405
1406             instId_to_thread_id[instId] = thread_id;
1407         }
1408
1409         IrisErrorCode processMessagesForCurrentThread(bool waitForAMessage)
1410         {

```

```

1411         if (waitForAMessage)
1412         {
1413             IrisErrorCode code = message_queue.waitForMessageForCurrentThread();
1414             if (code != E_ok)
1415             {
1416                 return code;
1417             }
1418         }
1419         message_queue.processRequestsForCurrentThread();
1420
1421         return E_ok;
1422     }
1423
1424 private:
1425     std::thread::id getDefaultThreadId()
1426     {
1427         return process_events_thread.getThreadId();
1428     }
1429
1430     IrisClient* const parent;
1431
1432     impl::IrisMessageQueue message_queue{this};
1433
1434     std::vector<std::thread::id> instId_to_thread_id;
1435
1436     IrisLogger& log;
1437
1438     impl::IrisProcessEventsThread process_events_thread{&message_queue, "ClientMsgHandler"};
1439 };
1440
1441 public: // IrisConnectionInterface
1442     virtual uint64_t registerIrisInterfaceChannel(IrisInterface* iris_interface, const std::string&
1443     connectionInfo) override
1444     {
1445         return channel_registry.registerChannel(iris_interface, connectionInfo);
1446     }
1447
1448     virtual void unregisterIrisInterfaceChannel(uint64_t channelId) override
1449     {
1450         IrisInterface* if_to_remove = channel_registry.getChannel(channelId);
1451
1452         std::vector<InstanceId> instIds_for_channel;
1453
1454         for (size_t i = 0; i < instIdToInterface.size(); i++)
1455         {
1456             if (instIdToInterface[i] == if_to_remove)
1457             {
1458                 InstanceId instId = InstanceId(i);
1459                 instIds_for_channel.push_back(instId);
1460             }
1461         }
1462
1463         if (instIds_for_channel.size() > 0)
1464         {
1465             // Create an instance to call instanceRegistry_unregisterInstance() with.
1466             IrisInstance instance_killer(this, "framework.IrisClient.instance_killer",
1467             IrisInstance::UNIQUEIFY);
1468             for (InstanceId instId : instIds_for_channel)
1469             {
1470                 instance_killer.irisCall().instanceRegistry_unregisterInstance(instId);
1471             }
1472         }
1473
1474         channel_registry.unregisterChannel(channelId);
1475     }
1476
1477     virtual IrisErrorCode processAsyncMessages(bool waitForAMessage) override
1478     {
1479         return receivingInterface.processMessagesForCurrentThread(waitForAMessage);
1480     }
1481
1482     virtual IrisInterface* getIrisInterface() override
1483     {
1484         return this;
1485     }
1486
1487     void unregisterChannel(uint64_t channelId)
1488     {
1489         channel_registry.unregisterChannel(channelId);
1490     }
1491
1492     // function called by class IrisPlugin
1493     uint64_t registerChannel(IrisC_CommunicationChannel* channel, const std::string& connectionInfo)
1494     {
1495         return channel_registry.registerChannel(channel, connectionInfo);
1496     }
1497
1498 public:

```

```

1504     void loadPlugin(const std::string& plugin_name)
1505     {
1506         assert(mode == IRIS_SERVICE_SERVER);
1507         assert(plugin == nullptr);
1508         plugin = std::unique_ptr<impl::IrisPlugin<IrisClient>>(new impl::IrisPlugin<IrisClient>(this,
1509 plugin_name));
1510     }
1511     void unloadPlugin()
1512     {
1513         assert(mode == IRIS_SERVICE_SERVER);
1514         plugin = nullptr;
1515     }
1516
1517     void initServiceServer(impl::IrisTcpSocket* socket_)
1518     {
1519         assert(mode == IRIS_SERVICE_SERVER);
1520         service_socket = socket_;
1521         socketSet.addSocket(service_socket);
1522         adapter.initServiceServer(service_socket, &socketSet, &receivingInterface, verbose);
1523     }
1524
1525 private:
1526     std::string getInstName(InstanceId instId)
1527     {
1528         // IrisLogger will generate a default name for unknown instances (empty string).
1529         return instId < instIdToInstName.size() ? instIdToInstName[instId] : std::string();
1530     }
1531
1532     void setInstName(InstanceId instId, const std::string& instName)
1533     {
1534         // Ignore ridiculously high instIds (programming errors).
1535         if (instId >= IrisMaxTotalInstances)
1536         {
1537             return;
1538         }
1539
1540         if (instId >= instIdToInstName.size())
1541         {
1542             instIdToInstName.resize(instId + 1, "");
1543         }
1544
1545         instIdToInstName[instId] = instName;
1546     }
1547
1548     // --- Private data. ---
1549
1550     IrisLogger log;
1551
1552     IrisInstance irisInstance;
1553
1554     std::string irisInstanceInstName;
1555
1556     GlobalInstanceSendingInterface globalInstanceSendingInterface{this};
1557
1558     ReceivingInterface receivingInterface{log, this};
1559
1560     impl::IrisTcpSocket sock{log, 0};
1561
1562     impl::IrisTcpSocket* service_socket{nullptr};
1563
1564     impl::IrisTcpSocketSet socketSet{log, 0};
1565
1566     std::vector<IrisInterface*> instIdToInterface;
1567
1568     std::vector<std::string> instIdToInstName;
1569
1570     impl::IrisChannelRegistry channel_registry{log};
1571
1572     IrisInterface* sendingInterface{nullptr};
1573
1574     uint32_t nextInstIdForRegisterInstanceCall{0};
1575
1576     struct OngoingInstanceRegistryCallEntry
1577     {
1578         OngoingInstanceRegistryCallEntry()
1579         {
1580         }
1581
1582         OngoingInstanceRegistryCallEntry(const std::string& method_, uint64_t id_ = IRIS_UINT64_MAX,
1583                                         uint64_t channelId_ = IRIS_UINT64_MAX)
1584             : method(method_)
1585             , id(id_)
1586             , channelId(channelId_)
1587         {
1588         }
1589     };
1590
1591

```

```

1613         std::string method; // instanceRegistry_registerInstance,
instanceRegistry_unregisterInstance or instanceRegistry_getList().
1614         uint64_t id{IRIS_UINT64_MAX}; // For instanceRegistry_registerInstance(): Original
request id. For instanceRegistry_unregisterInstance(): params.aInstId.
1615         uint64_t channelId{IRIS_UINT64_MAX}; // For instanceRegistry_registerInstance() only:
params.channelId.
1616     };
1617
1621     typedef std::map<uint64_t, OngoingInstanceRegistryCallEntry> OngoingInstanceRegistryCallMap;
1622
1623     OngoingInstanceRegistryCallMap ongoingInstanceRegistryCalls;
1624
1626     std::mutex ongoingInstanceRegistryCallsMutex;
1627
1629     unsigned verbose{0};
1630
1632     unsigned irisMessageLogLevel{0};
1633
1635     impl::IrisRpcAdapterTcp adapter{log};
1636
1638     std::unique_ptr<impl::IrisProcessEventsThread> socket_thread{nullptr};
1639
1641     Mode mode;
1642
1644     std::string component_name;
1645
1647     std::unique_ptr<impl::IrisPlugin<IrisClient>> plugin{nullptr};
1648
1650     std::string connectionStr{"(not connected)"};
1651
1654     uint64_t sleepOnDestructionMs{};
1655
1656 #ifndef _WIN32
1658     pid_t childPid{};
1659 #endif
1660
1662     int lastExitStatus{-1};
1663 };
1664
1665 NAMESPACE_IRIS_END
1666
1667 #endif // #ifndef ARM_INCLUDE_IrisClient_h

```

9.7 IrisCommandLineParser.h File Reference

Generic command line parser.

```

#include <stdint>
#include <map>
#include <string>
#include <vector>
#include <functional>
#include <exception>
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisErrorException.h"

```

Classes

- class [iris::IrisCommandLineParser](#)
- struct [iris::IrisCommandLineParser::Option](#)
Option container.

9.7.1 Detailed Description

Generic command line parser.

Copyright

Copyright (C) 2020-2023 Arm Limited. All rights reserved.

9.8 IrisCommandLineParser.h

[Go to the documentation of this file.](#)

```

1
2 #ifndef ARM_INCLUDE_IrisCommandLineParser_h
3 #define ARM_INCLUDE_IrisCommandLineParser_h
4
5 #include <stdint>
6 #include <map>
7 #include <string>
8 #include <vector>
9 #include <functional>
10 #include <exception>
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisErrorException.h"
14
15 NAMESPACE_IRIS_START
16
17 #if 0
18 #include <iostream>
19 #include "iris/IrisCommandLineParser.h"
20
21 int main(int argc, const char* argv[])
22 {
23     // Declare command line options.
24     iris::IrisCommandLineParser options("mytool", "Usage: mytool [OPTIONS]\n", "0.0.1");
25     options.addOption('v', "verbose", "Be more verbose (may be specified multiple times)."); // Switch
26     option.
27     options.addOption(0, "port", "Specify local server port.", "PORT", "7999"); // Option with argument,
28     without a short option.
29
30     // Parse command line.
31     options.parseCommandLine(argc, argv);
32
33     // Use options.
34     if (options.getSwitch("verbose"))
35     {
36         std::cout << "Verbose level: " << options.getSwitch("verbose") << "\n";
37     }
38     std::cout << "Port: " << options.getInt("port") << "\n";
39     return 0;
40 }
41 #endif
42
43 class IrisCommandLineParser
44 {
45 public:
46     static const bool KeepDashDash = true;
47
48     struct Option
49     {
50         // Public interface:
51         Option& setList(char sep = ',') { listSeparator = sep; return *this; }
52
53     private:
54         // Meta info:
55         char shortOption{};
56         std::string longOption;
57         std::string help;
58         std::string formalArgumentName;
59         std::string defaultValue;
60         char listSeparator{};
61         bool hasFormalArgument() const { return !formalArgumentName.empty(); }
62
63         // Actual values from command line:
64         std::string value;
65         bool isSpecified{};
66         void setValue(const std::string& v);
67     };
68 };

```

```

114         void unsetValue();
115
116         friend class IrisCommandLineParser;
117     };
118
119     IrisCommandLineParser(const std::string& programName, const std::string& usageHeader, const
std::string& versionStr, bool keepDashDash = false);
120
121     Option& addOption(char shortOption, const std::string& longOption, const std::string& help, const
std::string& formalArgumentName = std::string(), const std::string& defaultValue = std::string());
122
123     Option& addOption(char shortOption, const std::string& longOption, const std::string& help, const
std::string& formalArgumentName, int64_t defaultValue)
124     {
125         return addOption(shortOption, longOption, help, formalArgumentName,
std::to_string(defaultValue));
126     }
127
128     int parseCommandLine(int argc, const char** argv);
129     int parseCommandLine(int argc, char** argv) { return parseCommandLine(argc, const_cast<const
char**>(argv)); }
130
131     void noNonOptionArguments();
132
133     void pleaseSpecifyOneOf(const std::vector<std::string>& options, const std::vector<std::string>&
formalNonOptionArguments = std::vector<std::string>());
134
135     std::string getStr(const std::string& longOption) const;
136
137     int64_t getInt(const std::string& longOption) const;
138
139     uint64_t getUInt(const std::string& longOption) const;
140
141     double getDbl(const std::string& longOption) const;
142
143     uint64_t getSwitch(const std::string& longOption) const;
144
145     bool operator()(const std::string& longOption) const { return getSwitch(longOption) > 0; }
146
147     std::vector<std::string> getList(const std::string& longOption) const;
148
149     std::map<std::string, std::string> getMap(const std::string& longOption) const;
150
151     bool isSpecified(const std::string& longOption) const;
152
153     const std::vector<std::string>& getNonOptionArguments() const { return nonOptionArguments; }
154
155     std::vector<std::string>& getNonOptionArguments() { return nonOptionArguments; }
156
157     void clear();
158
159     int printMessage(const std::string& message, int error = 0, bool exit = false) const;
160
161     int printError(const std::string& message) const;
162
163     int printErrorAndExit(const std::string& message) const;
164
165     int printErrorAndExit(const std::exception& e) const;
166
167     void setMessageFunc(const std::function<int(const std::string& message, int error, bool exit)>&
messageFunc);
168
169     static int defaultMessageFunc(const std::string& message, int error, bool exit);
170
171     std::string getHelpMessage() const;
172
173     void setValue(const std::string& longOption, const std::string& value, bool append = false);
174
175     void unsetValue(const std::string& longOption);
176
177     void setProgramName(const std::string& programName_, bool append = false);
178
179     std::string getProgramName() const { return programName; }
180
181 private:
182     Option& getOption(const std::string& longOption);
183
184     const Option& getOption(const std::string& longOption) const;
185
186     std::string programName;
187
188     std::string usageHeader;
189
190     std::string versionStr;
191
192     bool keepDashDash;

```



```

298     std::vector<std::string> optionList;
299
302     std::map<std::string, Option> options;
303
305     std::vector<std::string> nonOptionArguments;
306
308     std::function<int(const std::string& message, int error, bool exit)> messageFunc;
309 };
310
311 NAMESPACE_IRIS_END
312
313 #endif // ARM_INCLUDE_IrisCommandLineParser_h

```

9.9 IrisElfDwarfArm.h File Reference

Constants for the register.canonicalRnScheme "ElfDwarf" for architecture Arm.

```
#include "iris/detail/IrisInterface.h"
```

```
#include "iris/detail/IrisCommon.h"
```

Enumerations

- enum **ElfDwarfArm** : uint64_t {
 - ARM_R0** = 0x2800000000 , **ARM_R1** = 0x2800000001 , **ARM_R2** = 0x2800000002 , **ARM_R3** = 0x2800000003 ,
 - ARM_R4** = 0x2800000004 , **ARM_R5** = 0x2800000005 , **ARM_R6** = 0x2800000006 , **ARM_R7** = 0x2800000007 ,
 - ARM_R8** = 0x2800000008 , **ARM_R9** = 0x2800000009 , **ARM_R10** = 0x280000000a , **ARM_R11** = 0x280000000b ,
 - ARM_R12** = 0x280000000c , **ARM_R13** = 0x280000000d , **ARM_R14** = 0x280000000e , **ARM_R15** = 0x280000000f ,
 - ARM_SPSR** = 0x2800000080 , **ARM_SPSR_fiq** = 0x2800000081 , **ARM_SPSR_irq** = 0x2800000082 ,
 - ARM_SPSR_abt** = 0x2800000083 ,
 - ARM_SPSR_und** = 0x2800000084 , **ARM_SPSR_svc** = 0x2800000085 , **ARM_R8_fiq** = 0x2800000097 ,
 - ARM_R9_fiq** = 0x2800000098 ,
 - ARM_R10_fiq** = 0x2800000099 , **ARM_R11_fiq** = 0x280000009a , **ARM_R12_fiq** = 0x280000009b ,
 - ARM_R13_fiq** = 0x280000009c ,
 - ARM_R14_fiq** = 0x280000009d , **ARM_R13_irq** = 0x280000009e , **ARM_R14_irq** = 0x280000009f , **ARM_R13_abt** = 0x28000000a0 ,
 - ARM_R14_abt** = 0x28000000a1 , **ARM_R13_und** = 0x28000000a2 , **ARM_R14_und** = 0x28000000a3 ,
 - ARM_R13_svc** = 0x28000000a4 ,
 - ARM_R14_svc** = 0x28000000a5 , **ARM_D0** = 0x2800000100 , **ARM_D1** = 0x2800000101 , **ARM_D2** = 0x2800000102 ,
 - ARM_D3** = 0x2800000103 , **ARM_D4** = 0x2800000104 , **ARM_D5** = 0x2800000105 , **ARM_D6** = 0x2800000106 ,
 - ARM_D7** = 0x2800000107 , **ARM_D8** = 0x2800000108 , **ARM_D9** = 0x2800000109 , **ARM_D10** = 0x280000010a ,
 - ARM_D11** = 0x280000010b , **ARM_D12** = 0x280000010c , **ARM_D13** = 0x280000010d , **ARM_D14** = 0x280000010e ,
 - ARM_D15** = 0x280000010f , **ARM_D16** = 0x2800000110 , **ARM_D17** = 0x2800000111 , **ARM_D18** = 0x2800000112 ,
 - ARM_D19** = 0x2800000113 , **ARM_D20** = 0x2800000114 , **ARM_D21** = 0x2800000115 , **ARM_D22** = 0x2800000116 ,
 - ARM_D23** = 0x2800000117 , **ARM_D24** = 0x2800000118 , **ARM_D25** = 0x2800000119 , **ARM_D26** = 0x280000011a ,
 - ARM_D27** = 0x280000011b , **ARM_D28** = 0x280000011c , **ARM_D29** = 0x280000011d , **ARM_D30** = 0x280000011e ,
 - ARM_D31** = 0x280000011f , **AARCH64_X0** = 0xb700000000 , **AARCH64_X1** = 0xb700000001 ,
 - AARCH64_X2** = 0xb700000002 ,
 - AARCH64_X3** = 0xb700000003 , **AARCH64_X4** = 0xb700000004 , **AARCH64_X5** = 0xb700000005 ,
 - AARCH64_X6** = 0xb700000006 ,

```

AARCH64_X7 = 0xb700000007 , AARCH64_X8 = 0xb700000008 , AARCH64_X9 = 0xb700000009 ,
AARCH64_X10 = 0xb70000000a ,
AARCH64_X11 = 0xb70000000b , AARCH64_X12 = 0xb70000000c , AARCH64_X13 = 0xb70000000d ,
AARCH64_X14 = 0xb70000000e ,
AARCH64_X15 = 0xb70000000f , AARCH64_X16 = 0xb700000010 , AARCH64_X17 = 0xb700000011 ,
AARCH64_X18 = 0xb700000012 ,
AARCH64_X19 = 0xb700000013 , AARCH64_X20 = 0xb700000014 , AARCH64_X21 = 0xb700000015 ,
AARCH64_X22 = 0xb700000016 ,
AARCH64_X23 = 0xb700000017 , AARCH64_X24 = 0xb700000018 , AARCH64_X25 = 0xb700000019 ,
AARCH64_X26 = 0xb70000001a ,
AARCH64_X27 = 0xb70000001b , AARCH64_X28 = 0xb70000001c , AARCH64_X29 = 0xb70000001d ,
AARCH64_X30 = 0xb70000001e ,
AARCH64_SP = 0xb70000001f , AARCH64_ELR = 0xb700000021 , AARCH64_V0 = 0xb700000040 ,
AARCH64_V1 = 0xb700000041 ,
AARCH64_V2 = 0xb700000042 , AARCH64_V3 = 0xb700000043 , AARCH64_V4 = 0xb700000044 ,
AARCH64_V5 = 0xb700000045 ,
AARCH64_V6 = 0xb700000046 , AARCH64_V7 = 0xb700000047 , AARCH64_V8 = 0xb700000048 ,
AARCH64_V9 = 0xb700000049 ,
AARCH64_V10 = 0xb70000004a , AARCH64_V11 = 0xb70000004b , AARCH64_V12 = 0xb70000004c ,
AARCH64_V13 = 0xb70000004d ,
AARCH64_V14 = 0xb70000004e , AARCH64_V15 = 0xb70000004f , AARCH64_V16 = 0xb700000050 ,
AARCH64_V17 = 0xb700000051 ,
AARCH64_V18 = 0xb700000052 , AARCH64_V19 = 0xb700000053 , AARCH64_V20 = 0xb700000054 ,
AARCH64_V21 = 0xb700000055 ,
AARCH64_V22 = 0xb700000056 , AARCH64_V23 = 0xb700000057 , AARCH64_V24 = 0xb700000058 ,
AARCH64_V25 = 0xb700000059 ,
AARCH64_V26 = 0xb70000005a , AARCH64_V27 = 0xb70000005b , AARCH64_V28 = 0xb70000005c ,
AARCH64_V29 = 0xb70000005d ,
AARCH64_V30 = 0xb70000005e , AARCH64_V31 = 0xb70000005f }

```

9.9.1 Detailed Description

Constants for the register.canonicalRnScheme "ElfDwarf" for architecture Arm.

Date

Copyright ARM Limited 2019. All Rights Reserved.

9.10 IrisElfDwarfArm.h

[Go to the documentation of this file.](#)

```

1
2 #ifndef ARM_INCLUDE_IrisElfDwarfArm_h
3 #define ARM_INCLUDE_IrisElfDwarfArm_h
4
5 #include "iris/detail/IrisInterface.h" // uint64_t
6 #include "iris/detail/IrisCommon.h" // namespace iris
7
8 NAMESPACE_IRIS_START
9
10 namespace ElfDwarf
11 {
12     enum ElfDwarfArm: uint64_t
13     {
14         // Constant      canonicalRn      Register      Architecture ELF-Arch DwarfReg
15         // =====
16         ARM_R0          = 0x2800000000, // R0          EM_ARM          40          0
17         ARM_R1          = 0x2800000001, // R1          EM_ARM          40          1
18         ARM_R2          = 0x2800000002, // R2          EM_ARM          40          2
19         ARM_R3          = 0x2800000003, // R3          EM_ARM          40          3
20         ARM_R4          = 0x2800000004, // R4          EM_ARM          40          4
21         ARM_R5          = 0x2800000005, // R5          EM_ARM          40          5
22         ARM_R6          = 0x2800000006, // R6          EM_ARM          40          6
23         ARM_R7          = 0x2800000007, // R7          EM_ARM          40          7
24         ARM_R8          = 0x2800000008, // R8          EM_ARM          40          8
25         ARM_R9          = 0x2800000009, // R9          EM_ARM          40          9
26         ARM_R10         = 0x280000000a, // R10         EM_ARM          40          10

```

34	ARM_R11	= 0x280000000b, // R11	EM_ARM	40	11
35	ARM_R12	= 0x280000000c, // R12	EM_ARM	40	12
36	ARM_R13	= 0x280000000d, // R13	EM_ARM	40	13
37	ARM_R14	= 0x280000000e, // R14	EM_ARM	40	14
38	ARM_R15	= 0x280000000f, // R15	EM_ARM	40	15
39	ARM_SPSR	= 0x2800000080, // SPSR	EM_ARM	40	128
40	ARM_SPSR_fiq	= 0x2800000081, // SPSR_fiq	EM_ARM	40	129
41	ARM_SPSR_irq	= 0x2800000082, // SPSR_irq	EM_ARM	40	130
42	ARM_SPSR_abt	= 0x2800000083, // SPSR_abt	EM_ARM	40	131
43	ARM_SPSR_und	= 0x2800000084, // SPSR_und	EM_ARM	40	132
44	ARM_SPSR_svc	= 0x2800000085, // SPSR_svc	EM_ARM	40	133
45	ARM_R8_fiq	= 0x2800000097, // R8_fiq	EM_ARM	40	151
46	ARM_R9_fiq	= 0x2800000098, // R9_fiq	EM_ARM	40	152
47	ARM_R10_fiq	= 0x2800000099, // R10_fiq	EM_ARM	40	153
48	ARM_R11_fiq	= 0x280000009a, // R11_fiq	EM_ARM	40	154
49	ARM_R12_fiq	= 0x280000009b, // R12_fiq	EM_ARM	40	155
50	ARM_R13_fiq	= 0x280000009c, // R13_fiq	EM_ARM	40	156
51	ARM_R14_fiq	= 0x280000009d, // R14_fiq	EM_ARM	40	157
52	ARM_R13_irq	= 0x280000009e, // R13_irq	EM_ARM	40	158
53	ARM_R14_irq	= 0x280000009f, // R14_irq	EM_ARM	40	159
54	ARM_R13_abt	= 0x28000000a0, // R13_abt	EM_ARM	40	160
55	ARM_R14_abt	= 0x28000000a1, // R14_abt	EM_ARM	40	161
56	ARM_R13_und	= 0x28000000a2, // R13_und	EM_ARM	40	162
57	ARM_R14_und	= 0x28000000a3, // R14_und	EM_ARM	40	163
58	ARM_R13_svc	= 0x28000000a4, // R13_svc	EM_ARM	40	164
59	ARM_R14_svc	= 0x28000000a5, // R14_svc	EM_ARM	40	165
60	ARM_D0	= 0x2800000100, // D0	EM_ARM	40	256
61	ARM_D1	= 0x2800000101, // D1	EM_ARM	40	257
62	ARM_D2	= 0x2800000102, // D2	EM_ARM	40	258
63	ARM_D3	= 0x2800000103, // D3	EM_ARM	40	259
64	ARM_D4	= 0x2800000104, // D4	EM_ARM	40	260
65	ARM_D5	= 0x2800000105, // D5	EM_ARM	40	261
66	ARM_D6	= 0x2800000106, // D6	EM_ARM	40	262
67	ARM_D7	= 0x2800000107, // D7	EM_ARM	40	263
68	ARM_D8	= 0x2800000108, // D8	EM_ARM	40	264
69	ARM_D9	= 0x2800000109, // D9	EM_ARM	40	265
70	ARM_D10	= 0x280000010a, // D10	EM_ARM	40	266
71	ARM_D11	= 0x280000010b, // D11	EM_ARM	40	267
72	ARM_D12	= 0x280000010c, // D12	EM_ARM	40	268
73	ARM_D13	= 0x280000010d, // D13	EM_ARM	40	269
74	ARM_D14	= 0x280000010e, // D14	EM_ARM	40	270
75	ARM_D15	= 0x280000010f, // D15	EM_ARM	40	271
76	ARM_D16	= 0x2800000110, // D16	EM_ARM	40	272
77	ARM_D17	= 0x2800000111, // D17	EM_ARM	40	273
78	ARM_D18	= 0x2800000112, // D18	EM_ARM	40	274
79	ARM_D19	= 0x2800000113, // D19	EM_ARM	40	275
80	ARM_D20	= 0x2800000114, // D20	EM_ARM	40	276
81	ARM_D21	= 0x2800000115, // D21	EM_ARM	40	277
82	ARM_D22	= 0x2800000116, // D22	EM_ARM	40	278
83	ARM_D23	= 0x2800000117, // D23	EM_ARM	40	279
84	ARM_D24	= 0x2800000118, // D24	EM_ARM	40	280
85	ARM_D25	= 0x2800000119, // D25	EM_ARM	40	281
86	ARM_D26	= 0x280000011a, // D26	EM_ARM	40	282
87	ARM_D27	= 0x280000011b, // D27	EM_ARM	40	283
88	ARM_D28	= 0x280000011c, // D28	EM_ARM	40	284
89	ARM_D29	= 0x280000011d, // D29	EM_ARM	40	285
90	ARM_D30	= 0x280000011e, // D30	EM_ARM	40	286
91	ARM_D31	= 0x280000011f, // D31	EM_ARM	40	287
92	AARCH64_X0	= 0xb700000000, // X0	EM_AARCH64	183	0
93	AARCH64_X1	= 0xb700000001, // X1	EM_AARCH64	183	1
94	AARCH64_X2	= 0xb700000002, // X2	EM_AARCH64	183	2
95	AARCH64_X3	= 0xb700000003, // X3	EM_AARCH64	183	3
96	AARCH64_X4	= 0xb700000004, // X4	EM_AARCH64	183	4
97	AARCH64_X5	= 0xb700000005, // X5	EM_AARCH64	183	5
98	AARCH64_X6	= 0xb700000006, // X6	EM_AARCH64	183	6
99	AARCH64_X7	= 0xb700000007, // X7	EM_AARCH64	183	7
100	AARCH64_X8	= 0xb700000008, // X8	EM_AARCH64	183	8
101	AARCH64_X9	= 0xb700000009, // X9	EM_AARCH64	183	9
102	AARCH64_X10	= 0xb70000000a, // X10	EM_AARCH64	183	10
103	AARCH64_X11	= 0xb70000000b, // X11	EM_AARCH64	183	11
104	AARCH64_X12	= 0xb70000000c, // X12	EM_AARCH64	183	12
105	AARCH64_X13	= 0xb70000000d, // X13	EM_AARCH64	183	13
106	AARCH64_X14	= 0xb70000000e, // X14	EM_AARCH64	183	14
107	AARCH64_X15	= 0xb70000000f, // X15	EM_AARCH64	183	15
108	AARCH64_X16	= 0xb700000010, // X16	EM_AARCH64	183	16
109	AARCH64_X17	= 0xb700000011, // X17	EM_AARCH64	183	17
110	AARCH64_X18	= 0xb700000012, // X18	EM_AARCH64	183	18
111	AARCH64_X19	= 0xb700000013, // X19	EM_AARCH64	183	19
112	AARCH64_X20	= 0xb700000014, // X20	EM_AARCH64	183	20
113	AARCH64_X21	= 0xb700000015, // X21	EM_AARCH64	183	21
114	AARCH64_X22	= 0xb700000016, // X22	EM_AARCH64	183	22
115	AARCH64_X23	= 0xb700000017, // X23	EM_AARCH64	183	23
116	AARCH64_X24	= 0xb700000018, // X24	EM_AARCH64	183	24
117	AARCH64_X25	= 0xb700000019, // X25	EM_AARCH64	183	25
118	AARCH64_X26	= 0xb70000001a, // X26	EM_AARCH64	183	26
119	AARCH64_X27	= 0xb70000001b, // X27	EM_AARCH64	183	27
120	AARCH64_X28	= 0xb70000001c, // X28	EM_AARCH64	183	28

```

121  AARCH64_X29 = 0xb70000001d, // X29      EM_AARCH64      183      29
122  AARCH64_X30 = 0xb70000001e, // X30      EM_AARCH64      183      30
123  AARCH64_SP  = 0xb70000001f, // SP       EM_AARCH64      183      31
124  AARCH64_ELR = 0xb700000021, // ELR      EM_AARCH64      183      33
125  AARCH64_V0  = 0xb700000040, // V0       EM_AARCH64      183      64
126  AARCH64_V1  = 0xb700000041, // V1       EM_AARCH64      183      65
127  AARCH64_V2  = 0xb700000042, // V2       EM_AARCH64      183      66
128  AARCH64_V3  = 0xb700000043, // V3       EM_AARCH64      183      67
129  AARCH64_V4  = 0xb700000044, // V4       EM_AARCH64      183      68
130  AARCH64_V5  = 0xb700000045, // V5       EM_AARCH64      183      69
131  AARCH64_V6  = 0xb700000046, // V6       EM_AARCH64      183      70
132  AARCH64_V7  = 0xb700000047, // V7       EM_AARCH64      183      71
133  AARCH64_V8  = 0xb700000048, // V8       EM_AARCH64      183      72
134  AARCH64_V9  = 0xb700000049, // V9       EM_AARCH64      183      73
135  AARCH64_V10 = 0xb70000004a, // V10      EM_AARCH64      183      74
136  AARCH64_V11 = 0xb70000004b, // V11      EM_AARCH64      183      75
137  AARCH64_V12 = 0xb70000004c, // V12      EM_AARCH64      183      76
138  AARCH64_V13 = 0xb70000004d, // V13      EM_AARCH64      183      77
139  AARCH64_V14 = 0xb70000004e, // V14      EM_AARCH64      183      78
140  AARCH64_V15 = 0xb70000004f, // V15      EM_AARCH64      183      79
141  AARCH64_V16 = 0xb700000050, // V16      EM_AARCH64      183      80
142  AARCH64_V17 = 0xb700000051, // V17      EM_AARCH64      183      81
143  AARCH64_V18 = 0xb700000052, // V18      EM_AARCH64      183      82
144  AARCH64_V19 = 0xb700000053, // V19      EM_AARCH64      183      83
145  AARCH64_V20 = 0xb700000054, // V20      EM_AARCH64      183      84
146  AARCH64_V21 = 0xb700000055, // V21      EM_AARCH64      183      85
147  AARCH64_V22 = 0xb700000056, // V22      EM_AARCH64      183      86
148  AARCH64_V23 = 0xb700000057, // V23      EM_AARCH64      183      87
149  AARCH64_V24 = 0xb700000058, // V24      EM_AARCH64      183      88
150  AARCH64_V25 = 0xb700000059, // V25      EM_AARCH64      183      89
151  AARCH64_V26 = 0xb70000005a, // V26      EM_AARCH64      183      90
152  AARCH64_V27 = 0xb70000005b, // V27      EM_AARCH64      183      91
153  AARCH64_V28 = 0xb70000005c, // V28      EM_AARCH64      183      92
154  AARCH64_V29 = 0xb70000005d, // V29      EM_AARCH64      183      93
155  AARCH64_V30 = 0xb70000005e, // V30      EM_AARCH64      183      94
156  AARCH64_V31 = 0xb70000005f, // V31      EM_AARCH64      183      95
157 }; // enum ElfDwarfArm
158
159 } // namespace ElfDwarf
160
161 NAMESPACE_IRIS_END
162
163 #endif // ARM_INCLUDE_IrisElfDwarfArm_h
164

```

9.11 IrisEventEmitter.h File Reference

A utility class for emitting Iris events.

```
#include "iris/detail/IrisEventEmitterBase.h"
```

Classes

- class [iris::IrisEventEmitter< ARGS >](#)
A helper class for generating Iris events.

9.11.1 Detailed Description

A utility class for emitting Iris events.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

9.12 IrisEventEmitter.h

[Go to the documentation of this file.](#)

```

1
8 #ifndef ARM_INCLUDE_IrisEventEmitter_h
9 #define ARM_INCLUDE_IrisEventEmitter_h
10
11 #include "iris/detail/IrisEventEmitterBase.h"
12
13 NAMESPACE_IRIS_START

```

```

14
35 template <typename... ARGS>
36 class IrisEventEmitter : public IrisEventEmitterBase
37 {
38 public:
40     IrisEventEmitter()
41         : IrisEventEmitterBase(sizeof...(ARGS))
42     {
43     }
44
45     void operator() (ARGS... args)
46     {
47         emitEvent(args...);
48     }
49 };
50
51 namespace iris
52 {
53     #ifdef ARM_INCLUDE_IrisEventEmitter_h
54     #endif

```

9.13 IrisGlobalInstance.h File Reference

Central instance which lives in the simulation engine and distributes all Iris messages.

```

#include "iris/IrisInstance.h"
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisFunctionDecoder.h"
#include "iris/detail/IrisInterface.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include "iris/detail/IrisReceivedRequest.h"
#include "iris/impl/IrisChannelRegistry.h"
#include "iris/impl/IrisPlugin.h"
#include "iris/impl/IrisServiceClient.h"
#include "iris/impl/IrisTcpServer.h"
#include <atomic>
#include <list>
#include <map>
#include <memory>
#include <mutex>
#include <string>
#include <thread>
#include <unordered_map>
#include <vector>

```

Classes

- class [iris::IrisGlobalInstance](#)

9.13.1 Detailed Description

Central instance which lives in the simulation engine and distributes all Iris messages.

Date

Copyright ARM Limited 2014-2023 All Rights Reserved.

The IrisGlobalInstance lives in the simulation engine. It contains all central data structures like the instance registry. It is responsible for distributing Iris messages to all in-process instances and to the IrisTcpServer.

9.14 IrisGlobalInstance.h

[Go to the documentation of this file.](#)

```

1
10 #ifndef ARM_INCLUDE_IrisGlobalInstance_h

```

```

11 #define ARM_INCLUDE_IrisGlobalInstance_h
12
13 #include "iris/IrisInstance.h"
14 #include "iris/detail/IrisCommon.h"
15 #include "iris/detail/IrisFunctionDecoder.h"
16 #include "iris/detail/IrisInterface.h"
17 #include "iris/detail/IrisLogger.h"
18 #include "iris/detail/IrisObjects.h"
19 #include "iris/detail/IrisReceivedRequest.h"
20
21 #include "iris/impl/IrisChannelRegistry.h"
22 #include "iris/impl/IrisPlugin.h"
23 #include "iris/impl/IrisServiceClient.h"
24 #include "iris/impl/IrisTcpServer.h"
25
26 #include <atomic>
27 #include <list>
28 #include <map>
29 #include <memory>
30 #include <mutex>
31 #include <string>
32 #include <thread>
33 #include <unordered_map>
34 #include <vector>
35
36 namespace IRIS_START
37 {
38     class IrisGlobalInstance : public IrisInterface
39     , public IrisConnectionInterface
40     {
41     public:
42         IrisGlobalInstance();
43
44         ~IrisGlobalInstance();
45
46         uint64_t registerChannel(IrisC_CommunicationChannel* channel, const std::string& connectionInfo);
47
48         void unregisterChannel(uint64_t channelId);
49
50         IrisInstance& getIrisInstance() { return irisInstance; }
51
52     public: // IrisConnectionInterface
53         virtual uint64_t registerIrisInterfaceChannel(IrisInterface* iris_interface, const std::string&
54             connectionInfo) override;
55
56         virtual void unregisterIrisInterfaceChannel(uint64_t channelId) override
57         {
58             unregisterChannel(channelId);
59         }
60
61         virtual IrisErrorCode processAsyncMessages(bool waitForAMessage) override
62         {
63             return irisProxyInterface.load()->processAsyncMessagesInProxy(waitForAMessage);
64         }
65
66         virtual IrisInterface* getIrisInterface() override
67         {
68             return this;
69         }
70
71         virtual void setIrisProxyInterface(IrisProxyInterface* irisProxyInterface_) override
72         {
73             if (logMessages)
74             {
75                 log.info("setIrisProxyInterface(irisProxyInterface=%p)\n", (void*)irisProxyInterface_);
76             }
77             irisProxyInterface = irisProxyInterface_ ? irisProxyInterface_ : &defaultIrisProxyInterface;
78         }
79
80     public:
81         // IrisInterface implementation.
82
83         virtual void irisHandleMessage(const uint64_t* message) override;
84
85         // Set log level for logging messages.
86         void setLogLevel(unsigned level);
87
88         // Emit log message.
89         void emitLogMessage(const std::string& message, const std::string& severityLevel);
90
91         void setLogMessageFunction(std::function<IrisErrorCode(const std::string&, const std::string&)>
92             func)
93         {
94             logMessageFunction = func;
95         }
96
97     private:

```

```

117 // --- Functions implemented locally in the global instance (registered in the functionDecoder). ---
118
120 void impl_instanceRegistry_registerInstance(IrisReceivedRequest& request);
121
123 void impl_instanceRegistry_unregisterInstance(IrisReceivedRequest& request);
124
126 void impl_instanceRegistry_getList(IrisReceivedRequest& request);
127
129 void impl_instanceRegistry_getInstanceInfoByInstId(IrisReceivedRequest& request);
130
132 void impl_instanceRegistry_getInstanceInfoByName(IrisReceivedRequest& request);
133
135 void impl_perInstanceExecution_setStateAll(IrisReceivedRequest& request);
136
138 void impl_perInstanceExecution_getStateAll(IrisReceivedRequest& request);
139
141 void impl_tcpServer_start(IrisReceivedRequest& request);
142
144 void impl_tcpServer_stop(IrisReceivedRequest& request);
145
147 void impl_tcpServer_getPort(IrisReceivedRequest& request);
148
150 void impl_plugin_load(IrisReceivedRequest& request);
151
153 void impl_service_connect(IrisReceivedRequest& request);
154
156 void impl_service_disconnect(IrisReceivedRequest& request);
157
159 void impl_logger_logMessage(IrisReceivedRequest& request);
160
161 // --- Private helpers ---
162
164 struct InstanceRegistryEntry
165 {
166     // instId: The index in instanceRegistry is the instId.
167     std::string instName;
168     uint64_t channelId{IRIS_UINT64_MAX}; // If this is IRIS_UINT64_MAX this means this entry
169     // is unused.
170     IrisInterface* iris_interface{nullptr};
171     std::string connectionInfo;
172
173     bool empty() const
174     {
175         return channelId == IRIS_UINT64_MAX;
176     }
177
178     void clear()
179     {
180         instName = "";
181         channelId = IRIS_UINT64_MAX;
182         iris_interface = nullptr;
183         connectionInfo = "";
184
185         assert(empty());
186     }
187 };
188
190 InstanceId registerInstance(std::string& instName,
191                             uint64_t channel_id,
192                             bool uniquify,
193                             IrisInterface* iris_interface);
194
196 void unregisterInstanceAndGenerateEvent(InstanceRegistryEntry* entry,
197                                         InstanceId aInstId,
198                                         uint64_t time,
199                                         std::list<IrisRequest>& deferred_event_requests);
200
202 const InstanceRegistryEntry* findInstanceRegistryEntry(InstanceId instId) const
203 {
204     if (instId >= InstanceId(instanceRegistry.size()))
205         return nullptr;
206
207     if (instanceRegistry[instId].empty())
208         return nullptr;
209
210     return &instanceRegistry[instId];
211 }
212
216 InstanceId addNewInstance(const std::string& instName,
217                           uint64_t channelId,
218                           IrisInterface* iris_interface);
219
220 // Stop the Iris Server (if running)
221 void stopServer();
222
223 // stop the Iris Client (if running)
224 void stopClient();

```

```

225
226     void loadPlugin(const std::string& plugin_path);
227
228     IrisErrorCode createEventStream(EventStream*&, const EventSourceInfo&, const
std::vector<std::string>&);
230
231     uint64_t getTimeForEvents();
232
233     std::string getInstName(InstanceId instId) const;
234
235     void initGlobalEventSources();
236
237     // Register functions for global instance
238     void registerGlobalFunctions();
239
240     // --- Private data ---
241
242     class Instance : public IrisInstance
243     {
244     public:
245         Instance()
246             : IrisInstance()
247         {
248             thisInstanceInfo.instName = "framework.GlobalInstance";
249             thisInstanceInfo.instId = IrisInstIdGlobalInstance;
250             setProperty("instName", getInstanceName());
251             setProperty("instId", getInstId());
252             // NOTE: This instance does not think it is registered.
253             //       This means it won't unregister itself when it is destroyed but that doesn't matter.
254             //       We will be cleaning up all that state anyway.
255         }
256
257         IrisInstanceEvent event_handler;
258     } irisInstance;
259
260     IrisEventRegistry instance_registry_changed_event_registry;
261
262     IrisEventRegistry shutdown_enter_event_registry;
263
264     IrisEventRegistry shutdown_leave_event_registry;
265
266     IrisEventRegistry log_message_event_registry;
267
268     std::vector<InstanceRegistryEntry> instanceRegistry;
269
270     //
271     std::mutex instance_registry_mutex;
272
273     std::vector<InstanceId> freeInstIds;
274
275     typedef std::map<std::string, uint64_t> InstanceRegistryNameToIdMap;
276
277     InstanceRegistryNameToIdMap instanceRegistryNameToId;
278
279     unsigned logMessages;
280
281     IrisLogger log;
282
283     // TCP server. This won't start listening until startServer() is called.
284     impl::IrisTcpServer* tcp_server;
285
286     impl::IrisServiceClient* service_client;
287
288     // Create and manage communication channels
289     impl::IrisChannelRegistry channel_registry;
290
291     // --- Load and manage plugins ---
292     using Plugin = impl::IrisPlugin<IrisGlobalInstance>;
293     std::unordered_map<std::string, std::unique_ptr<Plugin>> plugins;
294
295     std::mutex plugins_mutex;
296
297     std::mutex log_mutex;
298
299     class DefaultIrisProxyInterface : public IrisProxyInterface
300     {
301     public:
302         virtual void irisHandleMessageInProxy(IrisInterface* irisInterface, InstanceId instId,
const uint64_t* message) override;
303         virtual IrisErrorCode processAsyncMessagesInProxy(bool waitForAMessage) override;
304     } defaultIrisProxyInterface;
305
306     std::atomic<IrisProxyInterface*> irisProxyInterface{&defaultIrisProxyInterface};
307
308     std::function<IrisErrorCode(const std::string&, const std::string&> logMessageFunction;
309 };
310
311

```



```

353 NAMESPACE_IRIS_END
354
355 #endif // #ifndef ARM_INCLUDE_IrisGlobalInstance_h

```

9.15 IrisInstance.h File Reference

Boilerplate code for an Iris instance, including clients and components.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisCppAdapter.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisFunctionDecoder.h"
#include "iris/detail/IrisObjects.h"
#include "iris/detail/IrisReceivedRequest.h"
#include "iris/IrisInstanceEvent.h"
#include <cassert>
#include <mutex>
#include <functional>
#include "iris/IrisInstanceBuilder.h"

```

Classes

- class [iris::IrisInstance](#)

Macros

- #define **irisRegisterEventBufferCallback**(instancePtr, instanceType, functionName, description) register↵
EventBufferCallback<instanceType, &instanceType::impl_##functionName>(instancePtr, #functionName,
description, #instanceType)
Register an event buffer callback function using an EventBufferCallbackDelegate.
- #define **irisRegisterEventCallback**(instancePtr, instanceType, functionName, description) registerEvent↵
Callback<instanceType, &instanceType::impl_##functionName>(instancePtr, #functionName, description,
#instanceType)
*Register an event callback function using an EventCallbackDelegate Note: Use enableEvent() instead of
[irisRegisterEventCallback\(\)](#).*
- #define **irisRegisterFunction**(instancePtr, instanceType, functionName, functionInfoJson) register↵
Function(instancePtr, #functionName, &instanceType::impl_##functionName, functionInfoJson, #instance↵
Type)
*Register an Iris function implementation. The function can be implemented in this class or in any other class. The
helper macro is here to avoid repeating the function name. The 'impl_' prefix limits namespace pollution.*

Typedefs

- typedef IrisDelegate< const EventBufferCallbackData & > **iris::EventBufferCallbackDelegate**
- typedef IrisDelegate< uint64_t, const IrisValueMap &, uint64_t, uint64_t, bool, std::string & >
[iris::EventCallbackDelegate](#)
Event callback delegate (deprecated)

9.15.1 Detailed Description

Boilerplate code for an Iris instance, including clients and components.

Copyright

Copyright (C) 2015-2023 Arm Limited. All rights reserved.

The IrisInstance class provides infrastructure that is:

- Necessary for all Iris instances.

- Useful for Iris components.
- Useful for Iris clients.

Note

Using this class to implement a correct Iris interface is optional. This class does not form an interface between instances. It just forms an interface between itself and the code of an instance.

This class is useful for, and used by, both components and clients.

9.15.2 Typedef Documentation

9.15.2.1 EventCallbackDelegate

```
typedef IrisDelegate<uint64_t, const IrisValueMap&, uint64_t, uint64_t, bool, std::string&>
```

[iris::EventCallbackDelegate](#)

Event callback delegate (deprecated)

Note: Use `enableEvent()` instead of `irisRegisterEventCallback()`.

Used to register a function that can receive event callbacks.

```
iris::IrisErrorCode ec_FOO(EventStreamId esId, const iris::IrisValueMap &fields, uint64_t time,
                          InstanceId sInstId, bool syncEc, std::string &errorMessageOut)
```

Example:

```
class MyEventCallback
{
public:
    iris::IrisErrorCode impl_ec_FOO(EventStreamId esId, const iris::IrisValueMap &fields, uint64_t time,
                                    InstanceId sInstId, bool syncEc, std::string &errorMessageOut)
    {
        ...
        return E_ok;
    }
};

MyEventCallback* my_event_callback_ptr;
iris_instance->irisRegisterEventCallback(my_event_callback_ptr, MyEventCallback, ec_FOO, "Handle event
FOO");
```

9.16 IrisInstance.h

[Go to the documentation of this file.](#)

```
1
19 #ifndef ARM_INCLUDE_IrisInstance_h
20 #define ARM_INCLUDE_IrisInstance_h
21
22 #include "iris/detail/IrisCommon.h"
23 #include "iris/detail/IrisCppAdapter.h"
24 #include "iris/detail/IrisDelegate.h"
25 #include "iris/detail/IrisFunctionDecoder.h"
26 #include "iris/detail/IrisObjects.h"
27 #include "iris/detail/IrisReceivedRequest.h"
28 #include "iris/IrisInstanceEvent.h"
29
30 #include <cassert>
31 #include <mutex>
32 #include <functional>
33
34 namespace IRIS_START
35
36 typedef IrisDelegate<uint64_t, const IrisValueMap&, uint64_t, uint64_t, bool, std::string&>
    EventCallbackDelegate;
37 typedef IrisDelegate<const EventBufferCallbackData&> EventBufferCallbackDelegate;
38
39 class IrisInstantiationContext;
40 class IrisInstanceBuilder;
41
42 class IrisInstance
43 {
44 public:
45     // --- Construction and destruction. ---
46
47 #define irisRegisterFunction(instancePtr, instanceType, functionName, functionInfoJson)
48     registerFunction(instancePtr, #functionName, &instanceType::impl_##functionName, functionInfoJson,
49                     #instanceType)
```

```

84
88 #define irisRegisterEventCallback(instancePtr, instanceType, functionName, description)
    registerEventCallback<instanceType, &instanceType::impl_##functionName>(instancePtr, #functionName,
    description, #instanceType)
89
91 #define irisRegisterEventBufferCallback(instancePtr, instanceType, functionName, description)
    registerEventBufferCallback<instanceType, &instanceType::impl_##functionName>(instancePtr,
    #functionName, description, #instanceType)
92
100     static const uint64_t UNIQUIFY = (1 << 0);
101
103     static const uint64_t THROW_ON_ERROR = (1 << 1);
104
106     static const uint64_t DEFAULT_FLAGS = THROW_ON_ERROR;
107
109     static const bool SYNCHRONOUS = true;
110
113     IrisInstance(IrisConnectionInterface* connection_interface = nullptr,
114                 const std::string& instName = std::string(),
115                 uint64_t flags = DEFAULT_FLAGS);
116
117     IrisInstance(IrisInstantiationContext* context);
118
119     ~IrisInstance();
120
121     void setConnectionInterface(IrisConnectionInterface* connection_interface);
122
123     void processAsyncRequests();
124
125     IrisInterface* getRemoteIrisInterface()
126     {
127         return remoteIrisInterface;
128     }
129
130     void setThrowOnError(bool throw_on_error)
131     {
132         default_cppAdapter = throw_on_error ? &throw_cppAdapter : &nothrow_cppAdapter;
133     }
134
135     IrisErrorCode registerInstance(const std::string& instName, uint64_t flags = DEFAULT_FLAGS);
136
137     IrisErrorCode unregisterInstance();
138
139     template <class T>
140     void setProperty(const std::string& propertyName, const T& propertyValue)
141     {
142         propertyMap[propertyName].set(propertyValue);
143     }
144
145     const PropertyMap& getPropertyMap() const
146     {
147         return propertyMap;
148     }
149
150     // --- Interface for components. Provide functionality to clients. ---
151
152     template <class T>
153     void registerFunction(T* instance, const std::string& name, void
154     (T::*memberFunctionPtr)(IrisReceivedRequest&), const std::string& functionInfoJson, const
155     std::string& instanceTypeStr)
156     {
157         functionDecoder.registerFunction(instance, name, memberFunctionPtr, functionInfoJson,
158     instanceTypeStr);
159     }
160
161     void unregisterFunction(const std::string& name)
162     {
163         functionDecoder.unregisterFunction(name);
164     }
165
166     template <class T>
167     void registerEventCallback(T* instance, const std::string& name, const std::string& description,
168     void (T::*memberFunctionPtr)(IrisReceivedRequest&),
169     const std::string& instanceTypeStr)
170     {
171         std::string funcInfoJson = "{description:'" + description +
172         "',"
173         "args:{
174         " instId:{type:'NumberU64', description:'Target instance id.'},"
175         " esId:{type:'NumberU64', description:'Event stream id.'},"
176         " fields:{type:'Object', description:'Object which contains the names and values of event
177     source fields.'},"
178         " time:{type:'NumberU64', description:'Simulation time timestamp of the event.'},"
179         " sInstId:{type:'NumberU64', description:'Source instId: Instance which generated and sent
180     this event.'},"
181         " syncEc:{type:'Boolean', description:'Synchronous callback behaviour.', optional:true},"
182         "},"
183         "};"
184     }

```

```

296         "retval:{type:'Null'}}";
297         functionDecoder.registerFunction(instance, name, memberFunctionPtr, funcInfoJson,
instanceTypeStr);
298     }
299
300     void registerEventCallback(EventCallbackDelegate delegate, const std::string& name,
301                               const std::string& description, const std::string& dlgInstanceTypeStr)
302     {
303         eventCallbacks[name] = ECD(delegate);
304         registerEventCallback(this, name, description, &IrisInstance::impl_eventCallback,
dlgInstanceTypeStr);
305     }
306
307     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t, const AttributeValueMap&, uint64_t,
uint64_t, bool, std::string&)>
308     void registerEventCallback(T* instance, const std::string& name, const std::string& description,
309                               const std::string& dlgInstanceTypeStr)
310     {
311         registerEventCallback(EventCallbackDelegate::make<T, METHOD>(instance),
312                               name, description, dlgInstanceTypeStr);
313     }
314
315     template <class T>
316     void registerEventBufferCallback(T* instance, const std::string& name, const std::string&
description,
317                                     void (T::*memberFunctionPtr)(IrisReceivedRequest&),
318                                     const std::string& instanceTypeStr)
319     {
320         std::string funcInfoJson = "{description:'" + description + "',"
321         "args:{"
322         "  instId:{type:'NumberU64', description:'Target instance id.'},"
323         "  sInstId:{type:'NumberU64', description:'Source instId: Instance which generated and sent"
324         "this event buffer data.'},"
325         "  evBufId:{type:'NumberU64', description:'Event buffer id.'},"
326         "  events:{type:'EventData[]', description:'Array of EventData objects which represent the"
327         "individual events in chronological order.'}"
328         "},"
329         "retval:{type:'Null'}}";
330         functionDecoder.registerFunction(instance, name, memberFunctionPtr, funcInfoJson,
instanceTypeStr);
331     }
332
333     void registerEventBufferCallback(EventBufferCallbackDelegate delegate, const std::string& name,
334                                     const std::string& description, const std::string&
dlgInstanceTypeStr)
335     {
336         eventBufferCallbacks[name] = EBCD(delegate);
337         registerEventBufferCallback(this, name, description, &IrisInstance::impl_eventBufferCallback,
dlgInstanceTypeStr);
338     }
339
340     template <typename T, IrisErrorCode (T::*METHOD)(const EventBufferCallbackData& data)>
341     void registerEventBufferCallback(T* instance, const std::string& name, const std::string&
description,
342                                     const std::string& dlgInstanceTypeStr)
343     {
344         registerEventBufferCallback(EventBufferCallbackDelegate::make<T, METHOD>(instance),
345                                     name, description, dlgInstanceTypeStr);
346     }
347
348     void unregisterEventCallback(const std::string& name);
349
350     void unregisterEventBufferCallback(const std::string& name);
351
352     using EventCallbackFunction = std::function<IrisErrorCode(EventStreamId, const IrisValueMap&,
uint64_t, InstanceId, bool, std::string&)>;
353
354     void setCallback_IRIS_SIMULATION_TIME_EVENT(EventCallbackFunction f);
355
356     void setCallback_IRIS_SHUTDOWN_LEAVE(EventCallbackFunction f);
357
358     void addCallback_IRIS_INSTANCE_REGISTRY_CHANGED(EventCallbackFunction f);
359
360     void sendResponse(const uint64_t* response)
361     {
362         remoteIrisInterface->irisHandleMessage(response);
363     }
364
365     // --- Interface for clients. Access to other components. ---
366
367     IrisCppAdapter& irisCall() { return *default_cppAdapter; }
368
369     IrisCppAdapter& irisCallNoThrow() { return nothrow_cppAdapter; }
370
371     IrisCppAdapter& irisCallThrow() { return throw_cppAdapter; }
372
373     bool sendRequest(IrisRequest& req)

```

```

484     {
485         return irisCall().callAndPerhapsWaitForResponse(req);
486     }
487
488     // --- Misc functionality. ---
489
490     IrisInterface* getLocalIrisInterface() { return functionDecoder.getIrisInterface(); }
491
492     InstanceId getInstId() const { return thisInstanceInfo.instId; }
493
494     void setInstId(InstanceId instId) { thisInstanceInfo.instId = instId;
495     cppAdapter_request_manager.setInstId(instId); }
496
497     const std::string& getInstanceName() const { return thisInstanceInfo.instName; }
498
499     bool isRegistered() const { return cppAdapter_request_manager.isRegistered(); }
500
501     IrisInstanceBuilder* getBuilder();
502
503     bool isAdapterInitialized() const { return is_adapter_initialized; }
504
505     void setAdapterInitialized() { is_adapter_initialized = true; }
506
507     void setEventHandler(IrisInstanceEvent* handler);
508
509     void notifyStateChanged();
510
511     template<class T>
512     void publishCppInterface(const std::string& interfaceName, T *pointer, const std::string&
513     jsonDescription)
514     {
515         // Ignore null pointers: instance_getCppInterface...() promises to always return non-null
516         // pointers.
517         // (If there is no interface, do not publish it.)
518         if (pointer == nullptr)
519             return;
520
521         std::string functionInfoJson =
522             "{"
523             "  \"description\": \"\" + jsonDescription + \"\n\"
524             "  \"If this function is present it always returns a non-null pointer.\n\"
525             "  \"The caller of this function must make sure that the caller and callee use the same C++
526             interface class layout and run in the same process. \"
527             "  \"This effectively means that they both must be compiled using the same compiler using the
528             same header files. \"
529             "  \"The returned pointer is only meaningful if caller and callee run in the same process.\n\"
530             "  \"The meta-information provided alongside the returned pointer in CppInterfacePointer can
531             (and should) be used to do minimal compatibility checking between caller and callee, see
532             'CppInterfacePointer::isCompatibleWith()' in 'IrisObjects.h'.\", \"
533             "    \"args\": {
534             "      \"instId\": {
535             "        \"description\": \"Opaque number uniquely identifying the target instance.\",
536             "        \"type\": \"NumberU64\"
537             "      }
538             "    },
539             "    \"errors\": [
540             "      \"E_unknown_instance_id\"
541             "    ],
542             "    \"retval\": {
543             "      \"description\": \"Pointer to the requested C++ interface (and associated
544             meta-information) of this instance. Use 'CppInterfacePointer::isCompatibleWith()' to do a minimal
545             compatibility check before using the pointer.\",
546             "      \"type\": \"CppInterfacePointer\"
547             "    }
548             "  };
549
550         registerFunction(this, "instance_getCppInterface" + interfaceName,
551         &IrisInstance::impl_instance_getCppInterface, functionInfoJson, "IrisInstance");
552         cppInterfaceRegistry[interfaceName].set(pointer);
553     }
554
555     void unpublishCppInterface(const std::string& interfaceName)
556     {
557         unregisterFunction("instance_getCppInterface" + interfaceName);
558         cppInterfaceRegistry.erase(interfaceName);
559     }
560
561     // --- Blocking simulation time functions ---
562
563     void simulationTimeRun();
564
565     void simulationTimeStop();
566
567     void simulationTimeRunUntilStop(double timeoutInSeconds = 0.0);
568
569     bool simulationTimeWaitForStop(double timeoutInSeconds = 0.0);
570
571     bool simulationTimeIsRunning();

```

```

664
667 void simulationTimeDisableEvents();
678
685 void setPendingSyncStepResponse(RequestId requestId);
686
692 bool setSyncStepEventBufferId(EventBufferId evBufId);
693
704 void eventBufferDestroyed(EventBufferId evBufId);
705
713 bool isValidEvBufId(EventBufferId evBufId) const;
714
758 std::vector<EventStreamInfo> findEventSourcesAndFields(const std::string& spec, InstanceId
defaultInstId = IRIS_UINT64_MAX);
759 void findEventSourcesAndFields(const std::string& spec, std::vector<EventStreamInfo>&
eventStreamInfosOut, InstanceId defaultInstId = IRIS_UINT64_MAX);
760
761
808 void enableEvent(const std::string& eventSpec, std::function<void (const EventStreamInfo&
eventStreamInfo, IrisReceivedRequest& request)> callback, bool syncEc = false);
809
822 void enableEvent(const std::string& eventSpec, std::function<void ()> callback, bool syncEc =
false);
823
842 void disableEvent(const std::string& eventSpec);
843
851 std::vector<InstanceInfo> findInstanceInfos(const std::string& instancePathFilter = "all");
852
859 std::vector<EventSourceInfo> findEventSources(const std::string& instancePathFilter = "all");
860
865 const std::vector<EventSourceInfo>& getEventSourceInfosOfAllInstances();
866
874 void destroyAllEventStreams();
875
883 const InstanceInfo& getInstanceInfo(InstanceId instId);
884
901 InstanceInfo getInstanceInfo(const std::string& instancePathFilter);
902
913 const std::vector<InstanceInfo>& getInstanceList();
914
924 std::string getInstanceName(InstanceId instId);
925
935 InstanceId getInstanceId(const std::string& instName);
936
946 ResourceId getResourceId(InstanceId instId, const std::string& resourceSpec);
947
971 uint64_t resourceRead(InstanceId instId, const std::string& resourceSpec);
972
980 uint64_t resourceReadCrn(InstanceId instId, uint64_t canonicalRegisterNumber)
981 {
982     return resourceRead(instId, "crn:" + std::to_string(canonicalRegisterNumber));
983 }
984
994 std::string resourceReadStr(InstanceId instId, const std::string& resourceSpec);
995
1003 void resourceWrite(InstanceId instId, const std::string& resourceSpec, uint64_t value);
1004
1010 void resourceWriteCrn(InstanceId instId, uint64_t canonicalRegisterNumber, uint64_t value)
1011 {
1012     resourceWrite(instId, "crn:" + std::to_string(canonicalRegisterNumber), value);
1013 }
1014
1023 void resourceWriteStr(InstanceId instId, const std::string& resourceSpec, const std::string&
value);
1024
1028 const std::vector<ResourceGroupInfo>& getResourceGroups(InstanceId instId);
1029
1033 const ResourceInfo& getResourceInfo(InstanceId instId, ResourceId resourceId);
1034
1038 const ResourceInfo& getResourceInfo(InstanceId instId, const std::string& resourceSpec);
1039
1043 const std::vector<ResourceInfo>& getResourceInfos(InstanceId instId);
1044
1048 MemorySpaceId getMemorySpaceId(InstanceId instId, uint64_t canonicalMsn);
1049
1056 MemorySpaceId getMemorySpaceId(InstanceId instId, const std::string& name);
1057
1061 const MemorySpaceInfo& getMemorySpaceInfo(InstanceId instId, uint64_t canonicalMsn);
1062
1069 const MemorySpaceInfo& getMemorySpaceInfo(InstanceId instId, const std::string& name);
1070
1074 const std::vector<MemorySpaceInfo>& getMemorySpaceInfos(InstanceId instId);
1075
1079 void clearCachedMetaInfo();
1080
1081 private:
1082 void init(IrisConnectionInterface* connection_interface_ = nullptr,
1083           const std::string& instName = std::string(),

```

```

1084         uint64_t             flags             = DEFAULT_FLAGS);
1085
1086     struct InstanceMetaInfo
1087     {
1088         std::map<std::string,ResourceId> resourceSpecToResourceIdAll;
1089
1090         std::map<std::string,ResourceId> resourceSpecToResourceIdUsed;
1091
1092         std::vector<ResourceGroupInfo> groupInfos;
1093
1094         std::vector<ResourceInfo> resourceInfos;
1095
1096         std::map<ResourceId,uint64_t> resourceIdToIndex;
1097
1098         std::vector<MemorySpaceInfo> memorySpaceInfos;
1099
1100         std::vector<EventSourceInfo> eventSourceInfos;
1101         bool eventSourceInfosValid{};
1102     };
1103
1104     InstanceMetaInfo& getInstanceMetaInfo(InstanceId instId);
1105
1106     IrisInstance::InstanceMetaInfo& getResourceMetaInfo(InstanceId instId);
1107
1108     IrisInstance::InstanceMetaInfo& getMemoryMetaInfo(InstanceId instId);
1109
1110     IrisInstance::InstanceMetaInfo& getEventSourceMetaInfo(InstanceId instId);
1111
1112     void expandWildcardsInEventStreamInfos(std::vector<EventStreamInfo>& eventStreamInfosInOut,
1113     InstanceId defaultInstId);
1114
1115     void enableSimulationTimeEvents();
1116
1117     void enableShutdownLeaveEvents();
1118
1119     void enableInstanceRegistryChangedEvent();
1120
1121     void simulationTimeWaitForRunning();
1122
1123     void simulationTimeClearGotRunning();
1124
1125     std::string lookupInstanceNameLocal(InstanceId instId);
1126
1127     void inFlightReceivedRequestsPush(IrisReceivedRequest *request)
1128     {
1129         assert(request);
1130         request->setNextInFlightReceivedRequest(inFlightReceivedRequestsHead);
1131         inFlightReceivedRequestsHead = request;
1132     }
1133
1134     IrisReceivedRequest *inFlightReceivedRequestsPop()
1135     {
1136         IrisReceivedRequest *r = inFlightReceivedRequestsHead;
1137         if (r)
1138         {
1139             inFlightReceivedRequestsHead = r->getNextInFlightReceivedRequest();
1140             r->setNextInFlightReceivedRequest(nullptr);
1141         }
1142         return r;
1143     }
1144
1145     // --- Iris function implementations ---
1146     void impl_instance_getProperties(IrisReceivedRequest& request);
1147
1148     void impl_instance_ping(IrisReceivedRequest& request);
1149
1150     void impl_instance_ping2(IrisReceivedRequest& request);
1151
1152     void impl_instance_getCppInterface(IrisReceivedRequest& request);
1153
1154     void impl_eventCallback(IrisReceivedRequest& request);
1155
1156     void impl_eventBufferCallback(IrisReceivedRequest& request);
1157
1158     void impl_enableEventCallback(IrisReceivedRequest &request);
1159
1160     IrisErrorCode impl_ec_IrisInstance_IRIS_SIMULATION_TIME_EVENT(EventStreamId esId, const
1161     IrisValueMap& fields, uint64_t time,
1162
1163     InstanceId sInstId, bool syncEc,
1164     std::string& errorMessageOut);
1165
1166     IrisErrorCode impl_ec_IrisInstance_IRIS_SHUTDOWN_LEAVE(EventStreamId esId, const IrisValueMap&
1167     fields, uint64_t time,
1168
1169     InstanceId sInstId, bool syncEc,
1170     std::string& errorMessageOut);
1171
1172     IrisErrorCode impl_ec_IrisInstance_IRIS_INSTANCE_REGISTRY_CHANGED(EventStreamId esId, const

```

```

IrisValueMap& fields, uint64_t time,
1219                                     InstanceId sInstId, bool syncEc,
std::string& errorMessageOut);
1220
1221 // --- Iris specific data and state ---
1222
1224 IrisFunctionDecoder functionDecoder{log, this};
1225
1227 IrisCppAdapter::RequestManager cppAdapter_request_manager{log};
1228
1230 IrisCppAdapter throw_cppAdapter{&cppAdapter_request_manager, /*throw_on_error=*/true};
1231
1233 IrisCppAdapter nothrow_cppAdapter{&cppAdapter_request_manager, /*throw_on_error=*/false};
1234
1236 IrisCppAdapter* default_cppAdapter{&throw_cppAdapter};
1237
1241 IrisConnectionInterface* connection_interface{nullptr};
1242
1245 IrisInterface* remoteIrisInterface{nullptr};
1246
1247 protected:
1249     InstanceInfo thisInstanceInfo{};
1250
1251 private:
1253     bool instance_getProperties_called{false};
1254
1255     bool registered{false};
1256
1259     IrisReceivedRequest* inFlightReceivedRequestsHead{};
1260
1271     bool is_adapter_initialized{false};
1272
1273     uint64_t channelId{IRIS_UINT64_MAX};
1274
1276     IrisLogger log;
1277
1278 // --- Instance specific data and state ---
1279
1281     PropertyMap propertyMap{};
1282
1284     struct ECD
1285     {
1286         // Work around symbol length limits in Visual Studio (warning C4503)
1287         EventCallbackDelegate dlg;
1288         ECD() {}
1289         ECD(EventCallbackDelegate dlg_)
1290             : dlg(dlg_)
1291         {
1292         }
1293     };
1294     typedef std::map<std::string, ECD> EventCallbackMap;
1295     EventCallbackMap eventCallbacks{};
1296
1298     struct EBCD
1299     {
1300         // Work around symbol length limits in Visual Studio (warning C4503)
1301         EventBufferCallbackDelegate dlg;
1302         EBCD() {}
1303         EBCD(EventBufferCallbackDelegate dlg_)
1304             : dlg(dlg_)
1305         {
1306         }
1307     };
1308     typedef std::map<std::string, EBCD> EventBufferCallbackMap;
1309     EventBufferCallbackMap eventBufferCallbacks{};
1310
1312     struct EnableEventCallbackInfo
1313     {
1314         EnableEventCallbackInfo() = default;
1315         EnableEventCallbackInfo(const EventStreamInfo& eventStreamInfo_, std::function<void (const
1316 EventStreamInfo& eventStreamInfo, IrisReceivedRequest& request)> callback_):
1317             eventStreamInfo(eventStreamInfo_),
1318             callback(callback_)
1319         {
1320         }
1321         EventStreamInfo eventStreamInfo;
1322         std::function<void (const EventStreamInfo& eventStreamInfo, IrisReceivedRequest& request)>
1323 callback;
1324     };
1325     typedef std::map<std::string, EnableEventCallbackInfo> EnableEventCallbackMap;
1326     EnableEventCallbackMap enableEventCallbacks;
1327
1328     IrisInstanceBuilder* builder{nullptr};
1329
1331     IrisInstanceEvent *irisInstanceEvent{};
1332

```



```

1336     typedef std::map<std::string, CppInterfacePointer> CppInterfaceRegistryMap;
1337     CppInterfaceRegistryMap cppInterfaceRegistry{};
1338
1340     bool simulationTimeIsRunning_{};
1341
1343     bool simulationTimeGotRunningTrue{};
1344
1346     bool simulationTimeGotRunningFalse{};
1347
1349     std::mutex simulationTimeIsRunningMutex;
1350
1352     std::condition_variable simulationTimeIsRunningChanged;
1353
1355     EventStreamId simulationTimeEsId = IRIS_UINT64_MAX;
1356
1358     EventStreamId shutdownLeaveEsId = IRIS_UINT64_MAX;
1359
1361     EventStreamId instanceRegistryChangedEsId = IRIS_UINT64_MAX;
1362
1364     EventCallbackFunction simulationTimeCallbackFunction;
1365
1367     EventCallbackFunction shutdownLeaveCallbackFunction;
1368
1369     // List of callback functions for IRIS_INSTANCE_REGISTRY_CHANGED.
1370     std::vector<EventCallbackFunction> instanceRegistryChangedFunctions;
1371
1373     struct PendingSyncStepResponse
1374     {
1376         void setRequestId(RequestId requestId_)
1377         {
1378             requestId = requestId_;
1379         }
1380
1382         void setEventBufferId(EventBufferId evBufId_)
1383         {
1384             evBufId = evBufId_;
1385         }
1386
1388         bool isPending() const
1389         {
1390             return requestId != IRIS_UINT64_MAX;
1391         }
1392
1394         void clear()
1395         {
1396             requestId = IRIS_UINT64_MAX;
1397         }
1398
1400         void eventBufferDestroyed(EventBufferId evBufId_)
1401         {
1402             if (evBufId_ == evBufId)
1403             {
1404                 clear();
1405                 evBufId = IRIS_UINT64_MAX;
1406             }
1407         }
1408
1411         RequestId requestId{IRIS_UINT64_MAX};
1412
1414         EventBufferId evBufId{IRIS_UINT64_MAX};
1415     };
1416
1418     PendingSyncStepResponse pendingSyncStepResponse;
1419
1421
1423     std::vector<InstanceInfo> instanceInfos;
1424
1427     std::vector<uint64_t> instIdToIndex;
1428
1430     std::map<InstanceId, InstanceMetaInfo> instIdToMetaInfo;
1431
1433     std::vector<EventSourceInfo> eventSourceInfosOfAllInstances;
1434 };
1435
1436
1437 NAMESPACE_IRIS_END
1438
1439 #endif // #ifndef ARM_INCLUDE_IrisInstance_h
1440
1441 // Convenience #include.
1442 // (IrisInstanceBuilder needs the complete type of IrisInstance.)
1443 #include "iris/IrisInstanceBuilder.h"
1444

```

9.17 IrisInstanceBreakpoint.h File Reference

Breakpoint add-on to IrisInstance.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include <cstdio>
```

Classes

- struct [iris::BreakpointHitInfo](#)
- class [iris::IrisInstanceBreakpoint](#)
Breakpoint add-on for [IrisInstance](#).

Typedefs

- typedef IrisDelegate< const BreakpointInfo & > [iris::BreakpointDeleteDelegate](#)
Delete the breakpoint corresponding to the given information.
- typedef IrisDelegate< BreakpointInfo & > [iris::BreakpointSetDelegate](#)
Set a breakpoint corresponding to the given information.
- typedef IrisDelegate< const BreakpointHitInfo & > [iris::HandleBreakpointHitDelegate](#)
Handle a breakpoint hit corresponding to the given information.

9.17.1 Detailed Description

Breakpoint add-on to IrisInstance.

Copyright

Copyright (C) 2016-2020 Arm Limited. All rights reserved.

The IrisInstanceBreakpoint class:

- Implements all breakpoint-related Iris functions.
- Maintains and provides breakpoint information, for example type, address, and rscl.
- Converts between Iris breakpoint functions (breakpoint*()) and various C++ access functions.

9.17.2 Typedef Documentation

9.17.2.1 BreakpointDeleteDelegate

```
typedef IrisDelegate<const BreakpointInfo&> iris::BreakpointDeleteDelegate
```

Delete the breakpoint corresponding to the given information.

```
IrisErrorCode deleteBpt(const BreakpointInfo &bptInfo)
```

The breakpoint is guaranteed to exist and to be valid.

Error: Return E_* error code if it failed to delete the breakpoint.

9.17.2.2 BreakpointSetDelegate

```
typedef IrisDelegate<BreakpointInfo&> iris::BreakpointSetDelegate
```

Set a breakpoint corresponding to the given information.

```
IrisErrorCode setBpt(BreakpointInfo &bptInfo)
```

The breakpoint information members are guaranteed to be valid. The BreakpointInfo is non-const as the metadata might need to be modified. For example, in some cases it might be useful to align the address and fix the size of a data breakpoint. It should never modify the bptId, which is uniquely set by this add-on.

Error: Return E_* error code if it failed to set the breakpoint.

9.17.2.3 HandleBreakpointHitDelegate

typedef IrisDelegate<const BreakpointHitInfo&> [iris::HandleBreakpointHitDelegate](#)

Handle a breakpoint hit corresponding to the given information.

IrisErrorCode handleBreakpointHit(const BreakpointHitInfo &bptInfo)

The breakpoint is guaranteed to exist and to be valid.

Error: Return E_* error code if there is some error in handling the breakpoint.

9.18 IrisInstanceBreakpoint.h

[Go to the documentation of this file.](#)

```

1
12 #ifndef ARM_INCLUDE_IrisInstanceBreakpoint_h
13 #define ARM_INCLUDE_IrisInstanceBreakpoint_h
14
15 #include "iris/detail/IrisCommon.h"
16 #include "iris/detail/IrisDelegate.h"
17 #include "iris/detail/IrisLogger.h"
18 #include "iris/detail/IrisObjects.h"
19
20 #include <cstdio>
21
22 NAMESPACE_IRIS_START
23
24 class IrisInstance;
25 class IrisInstanceEvent;
26 class IrisEventRegistry;
27 class IrisReceivedRequest;
28
29 class EventStream;
30 struct EventSourceInfo;
31
32 struct BreakpointHitInfo
33 {
34     //Required for all breakpoint types
35     const BreakpointInfo& bptInfo;
36
37     //Register and memory breakpoint
38     const std::vector<uint64_t>& accessData;
39     bool isReadAccess;
40 };
41
42 typedef IrisDelegate<BreakpointInfo&> BreakpointSetDelegate;
43
44 typedef IrisDelegate<const BreakpointInfo&> BreakpointDeleteDelegate;
45
46 typedef IrisDelegate<const BreakpointHitInfo&> HandleBreakpointHitDelegate;
47
48 class IrisInstanceBreakpoint
49 {
50 public:
51     // --- Construction and destruction. ---
52     IrisInstanceBreakpoint(IrisInstance* irisInstance = nullptr);
53     ~IrisInstanceBreakpoint();
54
55     void attachTo(IrisInstance* irisInstance);
56
57     void setBreakpointSetDelegate(BreakpointSetDelegate delegate);
58
59     void setBreakpointDeleteDelegate(BreakpointDeleteDelegate delegate);
60
61     void setHandleBreakpointHitDelegate(HandleBreakpointHitDelegate delegate);
62
63     void setEventHandler(IrisInstanceEvent* handler);
64
65     void notifyBreakpointHit(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId);
66
67     void notifyBreakpointHitData(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId,
68                                 uint64_t accessAddr, uint64_t accessSize,
69                                 const std::string& accessRw, const std::vector<uint64_t>& data);
70
71     void notifyBreakpointHitRegister(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId,
72                                     const std::string& accessRw, const std::vector<uint64_t>& data);
73
74     const BreakpointInfo* getBreakpointInfo(BreakpointId bptId) const;
75
76     void addCondition(const std::string& name, const std::string& type, const std::string& description,
77                     const std::vector<std::string> bpt_types = std::vector<std::string>());
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218

```

```

225     void handleBreakpointHit(const BreakpointHitInfo& bptHitInfo);
226
227 private:
228     void impl_breakpoint_set(IrisReceivedRequest& request);
229
230     void impl_breakpoint_delete(IrisReceivedRequest& request);
231
232     void impl_breakpoint_getList(IrisReceivedRequest& request);
233
234     void impl_breakpoint_getAdditionalConditions(IrisReceivedRequest& request);
235
236     bool validateInterceptionParameters(IrisReceivedRequest& request, const InterceptionParams&
interceptionParams);
237
238     bool beginBreakpointHit(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId);
239
240     IrisErrorCode createEventStream(EventStream*&, const EventSourceInfo&, const
std::vector<std::string>&);
241
242     IrisErrorCode deleteBreakpoint(BreakpointId bpt);
243
244     void register_ec_IRIS_INSTANCE_REGISTRY_CHANGED();
245     IrisErrorCode ec_IRIS_INSTANCE_REGISTRY_CHANGED(EventStreamId esId, const IrisValueMap& fields,
uint64_t time,
246
247
248
249
250
251
252
253
254
255     IrisInstance* irisInstance;
256
257     IrisEventRegistry* breakpoint_hit_registry;
258
259     std::vector<BreakpointInfo> bptInfos;
260
261     std::vector<uint64_t> freeBptIds;
262
263     std::map<uint64_t, BreakpointAction> bptActions;
264
265     std::vector<BreakpointConditionInfo> additional_conditions;
266
267     BreakpointSetDelegate bptSetDelegate;
268
269     BreakpointDeleteDelegate bptDeleteDelegate;
270
271     HandleBreakpointHitDelegate handleBreakpointHitDelegate;
272
273     IrisLogger log;
274
275     bool instance_registry_changed_registered{};
276
277 };
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

9.19 IrisInstanceBuilder.h File Reference

A high level interface to build up functionality on an IrisInstance.

```

#include "iris/IrisEventEmitter.h"
#include "iris/IrisInstance.h"
#include "iris/IrisInstanceBreakpoint.h"
#include "iris/IrisInstanceDebuggableState.h"
#include "iris/IrisInstanceDisassembler.h"
#include "iris/IrisInstanceEvent.h"
#include "iris/IrisInstanceImage.h"
#include "iris/IrisInstanceMemory.h"
#include "iris/IrisInstancePerInstanceExecution.h"
#include "iris/IrisInstanceResource.h"
#include "iris/IrisInstanceSemihosting.h"
#include "iris/IrisInstanceCheckpoint.h"
#include "iris/IrisInstanceStep.h"
#include "iris/IrisInstanceTable.h"
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisElfDwarf.h"
#include <cassert>

```

Classes

- class [iris::IrisInstanceBuilder::AddressTranslationBuilder](#)
Used to set metadata for an address translation.
- class [iris::IrisInstanceBuilder::EventSourceBuilder](#)
Used to set metadata on an EventSource.
- class [iris::IrisInstanceBuilder::FieldBuilder](#)
Used to set metadata on a register field resource.
- class [iris::IrisInstanceBuilder](#)
Builder interface to populate an [IrisInstance](#) with registers, memory etc.
- class [iris::IrisInstanceBuilder::MemorySpaceBuilder](#)
Used to set metadata for a memory space.
- class [iris::IrisInstanceBuilder::ParameterBuilder](#)
Used to set metadata on a parameter.
- class [iris::IrisInstanceBuilder::RegisterBuilder](#)
Used to set metadata on a register resource.
- class [iris::IrisInstanceBuilder::SemihostingManager](#)
semihosting_apis [IrisInstanceBuilder](#) semihosting APIs
- class [iris::IrisInstanceBuilder::TableBuilder](#)
Used to set metadata for a table.
- class [iris::IrisInstanceBuilder::TableColumnBuilder](#)
Used to set metadata for a table column.

9.19.1 Detailed Description

A high level interface to build up functionality on an [IrisInstance](#).

Copyright

Copyright (C) 2016-2019 Arm Limited. All rights reserved.

9.20 IrisInstanceBuilder.h

[Go to the documentation of this file.](#)

```

1
2 #ifndef ARM_INCLUDE_IrisInstanceBuilder_h
3 #define ARM_INCLUDE_IrisInstanceBuilder_h
4
5 #include "iris/IrisEventEmitter.h"
6 #include "iris/IrisInstance.h"
7 #include "iris/IrisInstanceBreakpoint.h"
8 #include "iris/IrisInstanceDebuggableState.h"
9 #include "iris/IrisInstanceDisassembler.h"
10 #include "iris/IrisInstanceEvent.h"
11 #include "iris/IrisInstanceImage.h"
12 #include "iris/IrisInstanceMemory.h"
13 #include "iris/IrisInstancePerInstanceExecution.h"
14 #include "iris/IrisInstanceResource.h"
15 #include "iris/IrisInstanceSemihosting.h"
16 #include "iris/IrisInstanceCheckpoint.h"
17 #include "iris/IrisInstanceStep.h"
18 #include "iris/IrisInstanceTable.h"
19 #include "iris/detail/IrisCommon.h"
20 #include "iris/detail/IrisElfDwarf.h"
21
22 #include <cassert>
23
24 namespace IRIS_START
25 {
26     class IrisRegisterEventEmitterBase;
27
28     class IrisInstanceBuilder
29     {

```

```

41 private:
42     template <typename T, T* (IrisInstanceBuilder::*INIT_METHOD) ()>
43     class LazyAddOn
44     {
45     private:
46         IrisInstanceBuilder* parent;
47         T* add_on;
48     public:
49         LazyAddOn(IrisInstanceBuilder* parent_)
50             : parent(parent_)
51             , add_on(nullptr)
52         {
53         }
54
55         ~LazyAddOn()
56         {
57             delete add_on;
58         }
59
60         T* operator->()
61         {
62             if (add_on == nullptr)
63             {
64                 init();
65             }
66
67             return add_on;
68         }
69
70         operator T*()
71         {
72             if (add_on == nullptr)
73             {
74                 init();
75             }
76
77             return add_on;
78         }
79
80         T* getPtr()
81         {
82             return add_on;
83         }
84
85         void init()
86         {
87             assert(add_on == nullptr);
88             add_on = (parent->*INIT_METHOD)();
89         }
90     };
91
92     IrisInstance* iris_instance;
93 #define INTERNAL_LAZY(addon) \
94     addon* init##addon(); \
95     LazyAddOn<addon, &IrisInstanceBuilder::init##addon> \
96     INTERNAL_LAZY(IrisInstanceResource) \
97     inst_resource; \
98     INTERNAL_LAZY(IrisInstanceEvent) \
99     inst_event; \
100     INTERNAL_LAZY(IrisInstanceBreakpoint) \
101     inst_breakpoint; \
102     INTERNAL_LAZY(IrisInstanceMemory) \
103     inst_memory; \
104     INTERNAL_LAZY(IrisInstanceImage) \
105     inst_image; \
106     INTERNAL_LAZY(IrisInstanceImage_Callback) \
107     inst_image_cb; \
108     INTERNAL_LAZY(IrisInstanceStep) \
109     inst_step; \
110     INTERNAL_LAZY(IrisInstancePerInstanceExecution) \
111     inst_per_inst_exec; \
112     INTERNAL_LAZY(IrisInstanceTable) \
113     inst_table; \
114     INTERNAL_LAZY(IrisInstanceDisassembler) \
115     inst_disass; \
116     INTERNAL_LAZY(IrisInstanceDebuggableState) \
117     inst_dbg_state; \
118     INTERNAL_LAZY(IrisInstanceSemihosting) \
119     inst_semihost; \
120     INTERNAL_LAZY(IrisInstanceCheckpoint) \
121     inst_checkpoint;
122 #undef INTERNAL_LAZY
123
124 ResourceReadDelegate default_reg_read_delegate;
125 ResourceWriteDelegate default_reg_write_delegate;

```

```

136
139     bool canonicalRnSchemeIsAlreadySet{};
141
143
144     struct RegisterEventInfo
145     {
146         IrisInstanceEvent::EventSourceInfoAndDelegate event_info;
147
148         typedef std::vector<uint64_t> RscIdList;
149         RscIdList rscId_list;
150         IrisRegisterEventEmitterBase* event_emitter;
151
152         RegisterEventInfo()
153             : event_emitter(nullptr)
154         {
155         }
156     };
157
158     std::vector<RegisterEventInfo> register_read_event_info_list;
159     std::vector<RegisterEventInfo> register_update_event_info_list;
160
161     RegisterEventInfo* active_register_read_event_info{};
162     RegisterEventInfo* active_register_update_event_info{};
163
164     RegisterEventInfo* find_register_event(const std::vector<RegisterEventInfo*>&
165     register_event_info_list,
166                                           const std::string& name);
167
168     RegisterEventInfo* initRegisterReadEventInfo(const std::string& name);
169     RegisterEventInfo* initRegisterUpdateEventInfo(const std::string& name);
170
171     void finalizeRegisterEvent(RegisterEventInfo* event_info, bool is_read);
172     std::string associateRegisterWithTraceEvents(ResourceId rscId);
173
174     IrisErrorCode setBreakpoint(BreakpointInfo& info);
175     IrisErrorCode deleteBreakpoint(const BreakpointInfo& info);
176
177     struct RegisterEventEmitterPair
178     {
179         IrisRegisterEventEmitterBase* read;
180         IrisRegisterEventEmitterBase* update;
181
182         RegisterEventEmitterPair()
183             : read(nullptr)
184             , update(nullptr)
185         {
186         }
187     };
188
189     typedef std::map<uint64_t, RegisterEventEmitterPair> RscIdEventEmitterMap;
190     RscIdEventEmitterMap register_event_emitter_map;
191
192     BreakpointSetDelegate user_setBreakpoint;
193     BreakpointDeleteDelegate user_deleteBreakpoint;
194
195 public:
196     IrisInstanceBuilder(IrisInstance* iris_instance);
197
198     /* No destructor: IrisInstanceBuilder objects live as long as the instance
199     * they belong to. Do not key anything to the destructor.
200     */
201
202 #define INTERNAL_RESOURCE_BUILDER_MIXIN(TYPE)
203
204     TYPE& setName(const std::string& name)
205     {
206         info->resourceInfo.name = name;
207         return *this;
208     }
209
210     TYPE& setCname(const std::string& cname)
211     {
212         info->resourceInfo.cname = cname;
213         return *this;

```

```

234     }
235
236     \
237     \
238     TYPE& setDescription(const std::string& description)
239     {
240         info->resourceInfo.description = description;
241         return *this;
242     }
243
244     \
245     /* [[deprecated("Inconsistently named function. Use setDescription() instead.")]] */
246     TYPE& setDescr(const std::string& description)
247     {
248         return setDescription(description);
249     }
250
251     \
252     TYPE& setFormat(const std::string& format)
253     {
254         info->resourceInfo.format = format;
255         return *this;
256     }
257
258     \
259     TYPE& setBitWidth(uint64_t bitWidth)
260     {
261         info->resourceInfo.bitWidth = bitWidth;
262         return *this;
263     }
264
265     \
266     TYPE& setType(const std::string& type)
267     {
268         info->resourceInfo.type = type;
269         return *this;
270     }
271
272     \
273     TYPE& setRwMode(const std::string& rwMode)
274     {
275         info->resourceInfo.rwMode = rwMode;
276         return *this;
277     }
278
279     \
280     TYPE& setSubRscId(uint64_t subRscId)
281     {
282         info->resourceInfo.subRscId = subRscId;

```



```

287         return *this;
288     }
289
290     \
291     \
292     \
293     \
294     \
295     TYPE& addEnum(const std::string& symbol, const IrisValue& value, const std::string& description =
std::string())
296     {
297         info->resourceInfo.enums.push_back(EnumElementInfo(value, symbol, description));
298         return *this;
299     }
300
301     \
302     \
303     \
304     \
305     TYPE& addStringEnum(const std::string& stringValue, const std::string& description = std::string())
306     {
307         info->resourceInfo.enums.push_back(EnumElementInfo(IrisValue(stringValue), std::string(),
description));
308         return *this;
309     }
310
311     \
312     \
313     TYPE& setTag(const std::string& tag)
314     {
315         info->resourceInfo.tags[tag] = IrisValue(true);
316         return *this;
317     }
318
319     \
320     \
321     \
322     TYPE& setTag(const std::string& tag, const IrisValue& value)
323     {
324         info->resourceInfo.tags[tag] = value;
325         return *this;
326     }
327
328     \
329     \
330     \
331     \
332     TYPE& setReadDelegate(ResourceReadDelegate readDelegate)
333     {
334         info->readDelegate = readDelegate;
335         return *this;
336     }
337
338     \
339     \
340     \
341     \
342     TYPE& setWriteDelegate(ResourceWriteDelegate writeDelegate)
343     {

```

```

344     info->writeDelegate = writeDelegate;
345     return *this;
346 }
347
348
349
350
351 \
352
353
354
355 template <typename T, IrisErrorCode (T::*METHOD)(const ResourceInfo&, ResourceReadResult&)>
356 TYPE& setReadDelegate(T* instance)
357 {
358     return setReadDelegate(ResourceReadDelegate::make<T, METHOD>(instance));
359 }
360
361
362
363
364
365
366 template <IrisErrorCode (*FUNC)(const ResourceInfo&, ResourceReadResult&)>
367 TYPE& setReadDelegate()
368 {
369     return setReadDelegate(ResourceReadDelegate::make<FUNC>());
370 }
371
372
373
374
375 \
376
377
378
379 template <typename T, IrisErrorCode (T::*METHOD)(const ResourceInfo&, const ResourceWriteValue&)>
380 TYPE& setWriteDelegate(T* instance)
381 {
382     return setWriteDelegate(ResourceWriteDelegate::make<T, METHOD>(instance));
383 }
384
385
386
387
388
389
390 template <IrisErrorCode (*FUNC)(const ResourceInfo&, const ResourceWriteValue&)>
391 TYPE& setWriteDelegate()
392 {
393     return setWriteDelegate(ResourceWriteDelegate::make<FUNC>());
394 }
395
396
397
398
399
400 TYPE& setParentRscId(ResourceId parentRscId)
401 {
402     info->resourceInfo.parentRscId = parentRscId;
403     return *this;

```

```

404     }
405         \
406         \
407         ResourceId getRscId() const
408     {
409         \
410         return info->resourceInfo.rscId;
411     }
412         \
413         \
414         \
415         TYPE& getRscId(ResourceId &rscIdOut)
416     {
417         \
418         rscIdOut = info->resourceInfo.rscId;
419         return *this;
420     }
421 #define INTERNAL_REGISTER_BUILDER_MIXIN(TYPE)
422     \
423         \
424         \
425         TYPE& setLsbOffset(uint64_t lsbOffset)
426     {
427         \
428         info->resourceInfo.registerInfo.lsbOffset = lsbOffset;
429         return *this;
430     }
431         \
432         \
433         \
434         TYPE& setCanonicalRn(uint64_t canonicalRn_)
435     {
436         \
437         info->resourceInfo.registerInfo.canonicalRn = canonicalRn_;
438         info->resourceInfo.registerInfo.hasCanonicalRn = true;
439         return *this;
440     }
441         \
442         \
443         \
444         TYPE& setCanonicalRnElfDwarf(uint16_t architecture, uint16_t dwarfRegNum)
445     {
446         \
447         if (!instance_builder->canonicalRnSchemeIsAlreadySet) /* Only set property if not already set.
448         */\
449         {
450             \
451             if (getWithDefault(instance_builder->iris_instance->getPropertyMap(),
452             "register.canonicalRnScheme", "").getAsString().empty()) \
453             {
454                 \
455                 instance_builder->setPropertyCanonicalRnScheme("ElfDwarf");
456             }
457             \
458             instance_builder->canonicalRnSchemeIsAlreadySet = true;
459         }
460         \
461         return setCanonicalRn(makeCanonicalRnElfDwarf(architecture, dwarfRegNum));
462     }
463         \
464         \
465         \
466         \
467         \
468         \
469         \
470         \
471         \
472         \
473         \
474         \
475         \
476         \
477         \
478         \
479         \
480         \
481         \
482         \
483         \
484         \
485         \
486         \
487         \
488         \
489         \
490         \
491         \
492         \
493         \
494         \
495         \
496         \
497         \
498         \
499         \
500         \
501         \
502         \
503         \
504         \
505         \
506         \
507         \
508         \
509         \
510         \
511         \
512         \
513         \
514         \
515         \
516         \
517         \
518         \
519         \
520         \
521         \
522         \
523         \
524         \
525         \
526         \
527         \
528         \
529         \
530         \
531         \
532         \
533         \
534         \
535         \
536         \
537         \
538         \
539         \
540         \
541         \
542         \
543         \
544         \
545         \
546         \
547         \
548         \
549         \
550         \
551         \
552         \
553         \
554         \
555         \
556         \
557         \
558         \
559         \
560         \
561         \
562         \
563         \
564         \
565         \
566         \
567         \
568         \
569         \
570         \
571         \
572         \
573         \
574         \
575         \
576         \
577         \
578         \
579         \
580         \
581         \
582         \
583         \
584         \
585         \
586         \
587         \
588         \
589         \
590         \
591         \
592         \
593         \
594         \
595         \
596         \
597         \
598         \
599         \
600         \
601         \
602         \
603         \
604         \
605         \
606         \
607         \
608         \
609         \
610         \
611         \
612         \
613         \
614         \
615         \
616         \
617         \
618         \
619         \
620         \
621         \
622         \
623         \
624         \
625         \
626         \
627         \
628         \
629         \
630         \
631         \
632         \
633         \
634         \
635         \
636         \
637         \
638         \
639         \
640         \
641         \
642         \
643         \
644         \
645         \
646         \
647         \
648         \
649         \
650         \
651         \
652         \
653         \
654         \
655         \
656         \
657         \
658         \
659         \
660         \
661         \
662         \
663         \
664         \
665         \
666         \
667         \
668         \
669         \
670         \
671         \
672         \
673         \
674         \
675         \
676         \
677         \
678         \
679         \
680         \
681         \
682         \
683         \
684         \
685         \
686         \
687         \
688         \
689         \
690         \
691         \
692         \
693         \
694         \
695         \
696         \
697         \
698         \
699         \
700         \
701         \
702         \
703         \
704         \
705         \
706         \
707         \
708         \
709         \
710         \
711         \
712         \
713         \
714         \
715         \
716         \
717         \
718         \
719         \
720         \
721         \
722         \
723         \
724         \
725         \
726         \
727         \
728         \
729         \
730         \
731         \
732         \
733         \
734         \
735         \
736         \
737         \
738         \
739         \
740         \
741         \
742         \
743         \
744         \
745         \
746         \
747         \
748         \
749         \
750         \
751         \
752         \
753         \
754         \
755         \
756         \
757         \
758         \
759         \
760         \
761         \
762         \
763         \
764         \
765         \
766         \
767         \
768         \
769         \
770         \
771         \
772         \
773         \
774         \
775         \
776         \
777         \
778         \
779         \
780         \
781         \
782         \
783         \
784         \
785         \
786         \
787         \
788         \
789         \
790         \
791         \
792         \
793         \
794         \
795         \
796         \
797         \
798         \
799         \
800         \
801         \
802         \
803         \
804         \
805         \
806         \
807         \
808         \
809         \
810         \
811         \
812         \
813         \
814         \
815         \
816         \
817         \
818         \
819         \
820         \
821         \
822         \
823         \
824         \
825         \
826         \
827         \
828         \
829         \
830         \
831         \
832         \
833         \
834         \
835         \
836         \
837         \
838         \
839         \
840         \
841         \
842         \
843         \
844         \
845         \
846         \
847         \
848         \
849         \
850         \
851         \
852         \
853         \
854         \
855         \
856         \
857         \
858         \
859         \
860         \
861         \
862         \
863         \
864         \
865         \
866         \
867         \
868         \
869         \
870         \
871         \
872         \
873         \
874         \
875         \
876         \
877         \
878         \
879         \
880         \
881         \
882         \
883         \
884         \
885         \
886         \
887         \
888         \
889         \
890         \
891         \
892         \
893         \
894         \
895         \
896         \
897         \
898         \
899         \
900         \
901         \
902         \
903         \
904         \
905         \
906         \
907         \
908         \
909         \
910         \
911         \
912         \
913         \
914         \
915         \
916         \
917         \
918         \
919         \
920         \
921         \
922         \
923         \
924         \
925         \
926         \
927         \
928         \
929         \
930         \
931         \
932         \
933         \
934         \
935         \
936         \
937         \
938         \
939         \
940         \
941         \
942         \
943         \
944         \
945         \
946         \
947         \
948         \
949         \
950         \
951         \
952         \
953         \
954         \
955         \
956         \
957         \
958         \
959         \
960         \
961         \
962         \
963         \
964         \
965         \
966         \
967         \
968         \
969         \
970         \
971         \
972         \
973         \
974         \
975         \
976         \
977         \
978         \
979         \
980         \
981         \
982         \
983         \
984         \
985         \
986         \
987         \
988         \
989         \
990         \
991         \
992         \
993         \
994         \
995         \
996         \
997         \
998         \
999         \
1000        \

```

```

458         \
459         \
460     \
461     TYPE& setWriteMask(uint64_t value)
462     {
463         info->resourceInfo.setVector(info->resourceInfo.registerInfo.writeMask, value);
464         return *this;
465     }
466     \
467     \
468     \
469     \
470     \
471     \
472     \
473     \
474     template<typename Container>
475     TYPE& setWriteMaskFromContainer(const Container& container)
476     {
477         info->resourceInfo.setVectorFromContainer(info->resourceInfo.registerInfo.writeMask, container);
478         return *this;
479     }
480     \
481     \
482     \
483     \
484     \
485     \
486     template<typename T>
487     TYPE& setWriteMask(std::initializer_list<T>&& t)
488     {
489         setWriteMaskFromContainer(std::forward<std::initializer_list<T>>(t));
490         return *this;
491     }
492     \
493     \
494     \
495     \
496     \
497     TYPE& setResetData(uint64_t value)
498     {
499         info->resourceInfo.setVector(info->resourceInfo.registerInfo.resetData, value);
500         return *this;
501     }
502     \
503     \
504     \
505     \
506     \
507     \
508     \
509     \
510     template<typename Container>
511     TYPE& setResetDataFromContainer(const Container& container)
512     {
513         info->resourceInfo.setVectorFromContainer(info->resourceInfo.registerInfo.resetData, container);
514         return *this;
515     }
516     \

```

```

517         \
518     \
519         \
520         \
521     \
522     template<typename T>
523     TYPE& setResetData(std::initializer_list<T>&& t)
524     {
525         \
526         setResetDataFromContainer(std::forward<std::initializer_list<T>>(t));
527         \
528         \
529         \
530     \
531     \
532     TYPE& setResetString(const std::string& resetString)
533     {
534         \
535         info->resourceInfo.registerInfo.resetString = resetString;
536         \
537         \
538         \
539     \
540     TYPE& setAddressOffset(uint64_t addressOffset)
541     {
542         \
543         info->resourceInfo.registerInfo.addressOffset = addressOffset;
544         \
545         info->resourceInfo.registerInfo.hasAddressOffset = true;
546         \
547         \
548         \
549     TYPE& setBreakpointSupportInfo(const std::string& supported)
550     {
551         \
552         info->resourceInfo.registerInfo.breakpointSupport = supported;
553         \
554         \
555     \
556     #define INTERNAL_PARAMETER_BUILDER_MIXIN(TYPE)
557     \
558     \
559     \
560     \
561     TYPE& setDefaultData(uint64_t value)
562     {
563         \
564         info->resourceInfo.setVector(info->resourceInfo.parameterInfo.defaultData, value);
565         \
566         \
567         \
568         \
569         \
570         \
571         \
572         \
573     \
574     template<typename Container>
575     TYPE& setDefaultDataFromContainer(const Container& container)

```

```

576     {
577         info->resourceInfo.setVectorFromContainer(info->resourceInfo.parameterInfo.defaultData,
578         container); \
579         return *this;
580     }
581
582     \
583     \
584     \
585     \
586     template<typename T>
587     TYPE& setDefaultData(std::initializer_list<T>&& t)
588     {
589         setDefaultDataFromContainer(std::forward<std::initializer_list<T>>(t));
590         return *this;
591     }
592
593     \
594     \
595     \
596     TYPE& setDefaultString(const std::string& defaultString)
597     {
598         info->resourceInfo.parameterInfo.defaultString = defaultString;
599         return *this;
600     }
601
602     \
603     \
604     \
605     TYPE& setInitOnly(bool initOnly = true)
606     {
607         info->resourceInfo.parameterInfo.initOnly = initOnly;
608         /* Implicitly set read-only to make clear that parameter cannot be modified at run-time. */
609         info->resourceInfo.rwMode = initOnly ? "r" : std::string(); /* =rw */
610         return *this;
611     }
612
613     \
614     /* but can still be accessed by resource_getResourceInfo() for clients that know the
615     */ \
616     /* resource name. */
617     \
618     TYPE& setHidden(bool hidden = true)
619     {
620         info->resourceInfo.isHidden = hidden;
621         return *this;
622     }
623
624     \
625     \
626     \
627     TYPE& setMax(uint64_t value)
628     {
629         info->resourceInfo.setVector(info->resourceInfo.parameterInfo.max, value);

```

```

630         return *this;
631     }
632     \
633     \
634     \
635     \
636     \
637     \
638     \
639     \
640     template<typename Container>
641     TYPE& setMaxFromContainer(const Container& container)
642     {
643         info->resourceInfo.setVectorFromContainer(info->resourceInfo.parameterInfo.max, container);
644         return *this;
645     }
646     \
647     \
648     \
649     \
650     \
651     \
652     template<typename T>
653     TYPE& setMax(std::initializer_list<T>&& t)
654     {
655         setMaxFromContainer(std::forward<std::initializer_list<T>>(t));
656         return *this;
657     }
658     \
659     \
660     \
661     \
662     \
663     TYPE& setMin(uint64_t value)
664     {
665         info->resourceInfo.setVector(info->resourceInfo.parameterInfo.min, value);
666         return *this;
667     }
668     \
669     \
670     \
671     \
672     \
673     \
674     \
675     \
676     template<typename Container>
677     TYPE& setMinFromContainer(const Container& container)
678     {
679         info->resourceInfo.setVectorFromContainer(info->resourceInfo.parameterInfo.min, container);
680         return *this;
681     }
682     \
683     \
684     \
685     \
686     \
687     \
688     template<typename T>
689     TYPE& setMin(std::initializer_list<T>&& t)

```

```

690     {
691         \
692         \
693     }
694     \
695
696     class ParameterBuilder
697     {
698     private:
699         IrisInstanceResource::ResourceInfoAndAccess* info;
700
701     public:
702         ParameterBuilder(IrisInstanceResource::ResourceInfoAndAccess& info_)
703             : info(&info_)
704         {
705             info->resourceInfo.isParameter = true;
706         }
707
708         ParameterBuilder()
709             : info(nullptr)
710         {
711         }
712
713         INTERNAL_RESOURCE_BUILDER_MIXIN(ParameterBuilder)
714         INTERNAL_PARAMETER_BUILDER_MIXIN(ParameterBuilder)
715     };
716
717     class FieldBuilder;
718
719     class RegisterBuilder
720     {
721     private:
722         IrisInstanceResource::ResourceInfoAndAccess* info{};
723         IrisInstanceResource* inst_resource{};
724         IrisInstanceBuilder* instance_builder{};
725
726     public:
727         RegisterBuilder(IrisInstanceResource::ResourceInfoAndAccess& info_, IrisInstanceResource*
728 inst_resource_, IrisInstanceBuilder* instance_builder_)
729             : info(&info_)
730             , inst_resource(inst_resource_)
731             , instance_builder(instance_builder_)
732         {
733             info->resourceInfo.isRegister = true;
734         }
735
736         RegisterBuilder()
737         {
738         }
739
740         INTERNAL_RESOURCE_BUILDER_MIXIN(RegisterBuilder)
741         INTERNAL_REGISTER_BUILDER_MIXIN(RegisterBuilder)
742
743         FieldBuilder addField(const std::string& name, uint64_t lsbOffset, uint64_t bitWidth, const
744 std::string& description);
745
746         FieldBuilder addLogicalField(const std::string& name, uint64_t bitWidth, const std::string&
747 description);
748     };
749
750     class FieldBuilder
751     {
752     protected:
753         IrisInstanceResource::ResourceInfoAndAccess* info{};
754         RegisterBuilder* parent_reg{};
755         IrisInstanceBuilder* instance_builder{};
756
757     public:
758         FieldBuilder(IrisInstanceResource::ResourceInfoAndAccess& info_, RegisterBuilder* parent_reg_,
759 IrisInstanceBuilder* instance_builder_)
760             : info(&info_)
761             , parent_reg(parent_reg_)
762             , instance_builder(instance_builder_)
763         {
764         }
765
766         FieldBuilder()
767         {
768         }
769
770         INTERNAL_RESOURCE_BUILDER_MIXIN(FieldBuilder)
771         INTERNAL_REGISTER_BUILDER_MIXIN(FieldBuilder)
772
773         RegisterBuilder& parent()

```



```

804     {
805         return *parent_reg;
806     }
807
812     FieldBuilder addField(const std::string& name, uint64_t lsbOffset, uint64_t bitWidth, const
std::string& description)
813     {
814         return parent().addField(name, lsbOffset, bitWidth, description);
815     }
816
821     FieldBuilder addLogicalField(const std::string& name, uint64_t bitWidth, const std::string&
description)
822     {
823         return parent().addLogicalField(name, bitWidth, description);
824     }
825 };
826
827 #undef INTERNAL_RESOURCE_BUILDER_MIXIN
828 #undef INTERNAL_REGISTER_BUILDER_MIXIN
829 #undef INTERNAL_PARAMETER_BUILDER_MIXIN
830
861     void setDefaultResourceReadDelegate(ResourceReadDelegate delegate = ResourceReadDelegate())
862     {
863         default_reg_read_delegate = delegate;
864     }
865
893     template <typename T, IrisErrorCode (T::*METHOD)(const ResourceInfo&, ResourceReadResult&)>
894     void setDefaultResourceReadDelegate(T* instance)
895     {
896         setDefaultResourceReadDelegate(ResourceReadDelegate::make<T, METHOD>(instance));
897     }
898
918     template <IrisErrorCode (*FUNC)(const ResourceInfo&, ResourceReadResult&)>
919     void setDefaultResourceReadDelegate()
920     {
921         setDefaultResourceReadDelegate(ResourceReadDelegate::make<FUNC>());
922     }
923
953     void setDefaultResourceWriteDelegate(ResourceWriteDelegate delegate = ResourceWriteDelegate())
954     {
955         default_reg_write_delegate = delegate;
956     }
957
984     template <typename T, IrisErrorCode (T::*METHOD)(const ResourceInfo&, const ResourceWriteValue&)>
985     void setDefaultResourceWriteDelegate(T* instance)
986     {
987         setDefaultResourceWriteDelegate(ResourceWriteDelegate::make<T, METHOD>(instance));
988     }
989
1008     template <IrisErrorCode (*FUNC)(const ResourceInfo&, const ResourceWriteValue&)>
1009     void setDefaultResourceWriteDelegate()
1010     {
1011         setDefaultResourceWriteDelegate(ResourceWriteDelegate::make<*FUNC>());
1012     }
1013
1023     template <typename T, IrisErrorCode (T::*READER)(const ResourceInfo&, ResourceReadResult&),
1024             IrisErrorCode (T::*WRITER)(const ResourceInfo&, const ResourceWriteValue&)>
1025     void setDefaultResourceDelegates(T* instance)
1026     {
1027         setDefaultResourceReadDelegate(ResourceReadDelegate::make<T, READER>(instance));
1028         setDefaultResourceWriteDelegate(ResourceWriteDelegate::make<T, WRITER>(instance));
1029     }
1030
1053     void beginResourceGroup(const std::string& name,
1054                             const std::string& description,
1055                             uint64_t subRscIdStart = IRIS_UINT64_MAX,
1056                             const std::string& cname = std::string());
1057
1080     ParameterBuilder addParameter(const std::string& name, uint64_t bitWidth, const std::string&
description);
1081
1100     ParameterBuilder addStringParameter(const std::string& name, const std::string& description);
1101
1135     RegisterBuilder addRegister(const std::string& name, uint64_t bitWidth, const std::string&
description,
1136                                uint64_t addressOffset = IRIS_UINT64_MAX, uint64_t canonicalRn =
IRIS_UINT64_MAX);
1137
1156     RegisterBuilder addStringRegister(const std::string& name, const std::string& description);
1157
1178     RegisterBuilder addNoValueRegister(const std::string& name, const std::string& description, const
std::string& format);
1179
1198     ParameterBuilder enhanceParameter(ResourceId rscId)
1199     {
1200         return ParameterBuilder(*(inst_resource->getResourceInfo(rscId)));
1201     }

```

```

1202
1224 RegisterBuilder enhanceRegister(ResourceId rscId)
1225 {
1226     return RegisterBuilder(*(inst_resource->getResourceInfo(rscId)), inst_resource, this);
1227 }
1228
1251 void setPropertyCanonicalRnScheme(const std::string& canonicalRnScheme);
1252
1260 void setNextSubRscId(uint64_t nextSubRscId)
1261 {
1262     inst_resource->setNextSubRscId(nextSubRscId);
1263 }
1264
1274 void setTag(ResourceId rscId, const std::string& tag);
1275
1283 const ResourceInfo &getResourceInfo(ResourceId rscId)
1284 {
1285     return inst_resource->getResourceInfo(rscId)->resourceInfo;
1286 }
1287
1288
1302 class EventSourceBuilder
1303 {
1304 private:
1305     IrisInstanceEvent::EventSourceInfoAndDelegate& info;
1306
1307 public:
1308     EventSourceBuilder(IrisInstanceEvent::EventSourceInfoAndDelegate& info_)
1309         : info(info_)
1310     {
1311     }
1312
1318 EventSourceBuilder& setName(const std::string& name)
1319 {
1320     info.info.name = name;
1321     return *this;
1322 }
1323
1329 EventSourceBuilder& setDescription(const std::string& description)
1330 {
1331     info.info.description = description;
1332     return *this;
1333 }
1334
1340 EventSourceBuilder& setFormat(const std::string& format)
1341 {
1342     info.info.format = format;
1343     return *this;
1344 }
1345
1351 EventSourceBuilder& setCounter(bool counter = true)
1352 {
1353     info.info.counter = counter;
1354     return *this;
1355 }
1356
1364 EventSourceBuilder& setHidden(bool hidden = true)
1365 {
1366     info.info.isHidden = hidden;
1367     return *this;
1368 }
1369
1376 EventSourceBuilder& hasSideEffects(bool hasSideEffects_ = true)
1377 {
1378     info.info.hasSideEffects = hasSideEffects_;
1379     return *this;
1380 }
1381
1394 EventSourceBuilder& addField(const std::string& name, const std::string& type, uint64_t
sizeInBytes,
1395                             const std::string& description)
1396 {
1397     info.info.addField(name, type, sizeInBytes, description);
1398     return *this;
1399 }
1400
1411 EventSourceBuilder& addEnumElement(uint64_t value, const std::string& symbol, const
std::string& description = "")
1412 {
1413     if (info.info.fields.size() > 0)
1414     {
1415         info.info.fields.back().addEnumElement(value, symbol, description);
1416         return *this;
1417     }
1418     else
1419     {
1420         throw IrisInternalError("EventSourceInfo has no fields to add an enum element to.");

```

```

1421     }
1422 }
1423
1424 EventSourceBuilder& addEnumElement(const std::string& fieldName, uint64_t value, const
std::string& symbol, const std::string& description = "")
1434 {
1435     EventSourceFieldInfo *field = info.info.getField(fieldName);
1436     if (field == nullptr)
1437     {
1438         throw IrisInternalError("addEnumElement(): Field " + fieldName + " not found");
1439     }
1440     field->addEnumElement(value, symbol, description);
1441     return *this;
1442 }
1443
1444 EventSourceBuilder& removeEnumElement(const std::string& fieldName, uint64_t value)
1451 {
1452     EventSourceFieldInfo *field = info.info.getField(fieldName);
1453     if (field == nullptr)
1454     {
1455         throw IrisInternalError("removeEnumElement(): Field " + fieldName + " not found");
1456     }
1457     field->removeEnumElement(value);
1458     return *this;
1459 }
1460
1461 EventSourceBuilder& renameEnumElement(const std::string& fieldName, uint64_t value, const
std::string& newEnumSymbol)
1471 {
1472     EventSourceFieldInfo *field = info.info.getField(fieldName);
1473     if (field == nullptr)
1474     {
1475         throw IrisInternalError("renameEnumElement(): Field " + fieldName + " not found");
1476     }
1477     field->renameEnumElement(value, newEnumSymbol);
1478     return *this;
1479 }
1480
1481 EventSourceBuilder& setEventStreamCreateDelegate(EventStreamCreateDelegate delegate)
1490 {
1491     info.createEventStream = delegate;
1492     return *this;
1493 }
1494
1495 template <typename T,
1508         IrisErrorCode (T::METHOD) (EventStream*&, const EventSourceInfo&, const
std::vector<std::string>&)>
1509 EventSourceBuilder& setEventStreamCreateDelegate(T* instance)
1510 {
1511     return setEventStreamCreateDelegate(EventStreamCreateDelegate::make<T, METHOD>(instance));
1512 }
1513
1514 template<typename T>
1528 EventSourceBuilder& addOption(const std::string& name, const std::string& type, const T&
defaultValue,
1529                             bool optional, const std::string& description)
1530 {
1531     info.info.addOption(name, type, defaultValue, optional, description);
1532     return *this;
1533 }
1534
1535 };
1536
1537 EventSourceBuilder addEventSource(const std::string& name, bool isHidden = false)
1551 {
1552     return EventSourceBuilder(inst_event->addEventSource(name, isHidden));
1553 }
1554
1555 EventSourceBuilder addEventSource(const std::string& name, IrisEventEmitterBase& event_emitter,
bool isHidden = false)
1567 {
1568     IrisInstanceEvent::EventSourceInfoAndDelegate& info = inst_event->addEventSource(name,
isHidden);
1569
1570     event_emitter.setIrisInstance(iris_instance);
1571     event_emitter.setEvSrcId(info.info.evSrcId);
1572     info.createEventStream = EventStreamCreateDelegate::make<IrisEventEmitterBase,
1573 &IrisEventEmitterBase::createEventStream>(&event_emitter);
1574
1575     return EventSourceBuilder(info);
1576 }
1577
1578 EventSourceBuilder enhanceEventSource(const std::string& name)
1588 {
1589     IrisInstanceEvent::EventSourceInfoAndDelegate& info = inst_event->enhanceEventSource(name);
1590     return EventSourceBuilder(info);
1591 }
1592

```

```

1593
1600     void renameEventSource(const std::string& name, const std::string& newName)
1601     {
1602         inst_event->renameEventSource(name, newName);
1603     }
1604
1610     void deleteEventSource(const std::string& name)
1611     {
1612         inst_event->deleteEventSource(name);
1613     }
1614
1621     bool hasEventSource(const std::string& name)
1622     {
1623         return inst_event->hasEventSource(name);
1624     }
1625
1651     EventSourceBuilder setRegisterReadEvent(const std::string& name, const std::string& description =
std::string());
1652
1678     EventSourceBuilder setRegisterReadEvent(const std::string& name, IrisRegisterEventEmitterBase&
event_emitter);
1679
1686     void finalizeRegisterReadEvent();
1687
1714     EventSourceBuilder setRegisterUpdateEvent(const std::string& name, const std::string& description =
std::string());
1715
1742     EventSourceBuilder setRegisterUpdateEvent(const std::string& name, IrisRegisterEventEmitterBase&
event_emitter);
1743
1750     void finalizeRegisterUpdateEvent();
1751
1758     void resetRegisterReadEvent();
1759
1766     void resetRegisterUpdateEvent();
1767
1799     void setDefaultEsCreateDelegate(EventStreamCreateDelegate delegate)
1800     {
1801         inst_event->setDefaultEsCreateDelegate(delegate);
1802     }
1803
1834     template <typename T, IrisErrorCode (T::*METHOD)(EventStream*&, const EventSourceInfo&, const
std::vector<std::string>&)>
1835     void setDefaultEsCreateDelegate(T* instance)
1836     {
1837         setDefaultEsCreateDelegate(EventStreamCreateDelegate::make<T, METHOD>(instance));
1838     }
1839
1862     template <IrisErrorCode (*FUNC)(EventStream*&, const EventSourceInfo&, const
std::vector<std::string>&)>
1863     void setDefaultEsCreateDelegate()
1864     {
1865         setDefaultEsCreateDelegate(EventStreamCreateDelegate::make<FUNC>());
1866     }
1867
1874     IrisInstanceEvent* getIrisInstanceEvent() { return inst_event; }
1875
1907     void setBreakpointSetDelegate(BreakpointSetDelegate delegate)
1908     {
1909         if (inst_breakpoint.getPtr() == nullptr)
1910         {
1911             // Ensure the underlying IrisInstanceBreakpoint object is initialised too.
1912             inst_breakpoint.init();
1913         }
1914         user_setBreakpoint = delegate;
1915     }
1916
1938     template <typename T, IrisErrorCode (T::*METHOD)(BreakpointInfo&)>
1939     void setBreakpointSetDelegate(T* instance)
1940     {
1941         setBreakpointSetDelegate(BreakpointSetDelegate::make<T, METHOD>(instance));
1942     }
1943
1957     template <IrisErrorCode (*FUNC)(BreakpointInfo&)>
1958     void setBreakpointSetDelegate()
1959     {
1960         setBreakpointSetDelegate(BreakpointSetDelegate::make<FUNC>());
1961     }
1962
1984     void setBreakpointDeleteDelegate(BreakpointDeleteDelegate delegate)
1985     {
1986         if (inst_breakpoint.getPtr() == nullptr)
1987         {
1988             // Ensure the underlying IrisInstanceBreakpoint object is initialised too.
1989             inst_breakpoint.init();
1990         }
1991         user_deleteBreakpoint = delegate;

```

```

1992     }
1993
2015     template <typename T, IrisErrorCode (T::*METHOD)(const BreakpointInfo&)>
2016     void setBreakpointDeleteDelegate(T* instance)
2017     {
2018         setBreakpointDeleteDelegate(BreakpointDeleteDelegate::make<T, METHOD>(instance));
2019     }
2020
2034     template <IrisErrorCode (*FUNC)(const BreakpointInfo&)>
2035     void setBreakpointDeleteDelegate()
2036     {
2037         setBreakpointDeleteDelegate(BreakpointDeleteDelegate::make<FUNC>());
2038     }
2039
2061     void setHandleBreakpointHitDelegate(HandleBreakpointHitDelegate delegate)
2062     {
2063         if (inst_breakpoint.getPtr() == nullptr)
2064         {
2065             // Ensure the underlying IrisInstanceBreakpoint object is initialised too.
2066             inst_breakpoint.init();
2067         }
2068
2069         inst_breakpoint->setHandleBreakpointHitDelegate(delegate);
2070     }
2071
2093     template <typename T, IrisErrorCode (T::*METHOD)(const BreakpointHitInfo&)>
2094     void setHandleBreakpointHitDelegate(T* instance)
2095     {
2096         setHandleBreakpointHitDelegate(HandleBreakpointHitDelegate::make<T, METHOD>(instance));
2097     }
2098
2112     template <IrisErrorCode (*FUNC)(const BreakpointHitInfo&)>
2113     void setHandleBreakpointHitDelegate()
2114     {
2115         setHandleBreakpointHitDelegate(HandleBreakpointHitDelegate::make<FUNC>());
2116     }
2117
2128     void notifyBreakpointHit(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId)
2129     {
2130         inst_breakpoint->notifyBreakpointHit(bptId, time, pc, pcSpaceId);
2131     }
2132
2148     void notifyBreakpointHitData(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId
pcSpaceId,
2149                                uint64_t accessAddr, uint64_t accessSize,
2150                                const std::string& accessRw, const std::vector<uint64_t>& data)
2151     {
2152         inst_breakpoint->notifyBreakpointHitData(bptId, time, pc, pcSpaceId, accessAddr, accessSize,
accessRw, data);
2153     }
2154
2168     void notifyBreakpointHitRegister(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId
pcSpaceId,
2169                                    const std::string& accessRw, const std::vector<uint64_t>& data)
2170     {
2171         inst_breakpoint->notifyBreakpointHitRegister(bptId, time, pc, pcSpaceId, accessRw, data);
2172     }
2173
2181     const BreakpointInfo* getBreakpointInfo(BreakpointId bptId)
2182     {
2183         return inst_breakpoint->getBreakpointInfo(bptId);
2184     }
2185
2187     void addBreakpointCondition(const std::string& name, const std::string& type, const std::string&
description,
2188                                const std::vector<std::string> bpt_types = std::vector<std::string>())
2189     {
2190         inst_breakpoint->addCondition(name, type, description, bpt_types);
2191     }
2192
2206     class MemorySpaceBuilder
2207     {
2208     private:
2209         IrisInstanceMemory::SpaceInfoAndAccess& info;
2210
2211     public:
2212         MemorySpaceBuilder(IrisInstanceMemory::SpaceInfoAndAccess& info_)
2213             : info(info_)
2214         {
2215         }
2216
2223         MemorySpaceBuilder& setName(const std::string& name)
2224         {
2225             info.spaceInfo.name = name;
2226             return *this;
2227         }
2228     }

```

```

2235     MemorySpaceBuilder& setDescription(const std::string& description)
2236     {
2237         info.spaceInfo.description = description;
2238         return *this;
2239     }
2240
2241     MemorySpaceBuilder& setMinAddr(uint64_t minAddr)
2242     {
2243         info.spaceInfo.minAddr = minAddr;
2244         return *this;
2245     }
2246
2247     MemorySpaceBuilder& setMaxAddr(uint64_t maxAddr)
2248     {
2249         info.spaceInfo.maxAddr = maxAddr;
2250         return *this;
2251     }
2252
2253     MemorySpaceBuilder& setCanonicalMsn(uint64_t canonicalMsn)
2254     {
2255         info.spaceInfo.canonicalMsn = canonicalMsn;
2256         return *this;
2257     }
2258
2259     MemorySpaceBuilder& setEndianness(const std::string& endianness)
2260     {
2261         info.spaceInfo.endianness = endianness;
2262         return *this;
2263     }
2264
2265     MemorySpaceBuilder& addAttribute(const std::string& name, AttributeInfo attrib)
2266     {
2267         info.spaceInfo.attrib[name] = attrib;
2268         return *this;
2269     }
2270
2271     MemorySpaceBuilder& setAttributes(const AttributeInfoMap& attribInfoMap)
2272     {
2273         info.spaceInfo.attrib = attribInfoMap;
2274         return *this;
2275     }
2276
2277     MemorySpaceBuilder& setAttributeDefault(const std::string& name, IrisValue value)
2278     {
2279         info.spaceInfo.attribDefaults[name] = value;
2280         return *this;
2281     }
2282
2283     MemorySpaceBuilder& setSupportedByteWidths(uint64_t supportedByteWidths)
2284     {
2285         info.spaceInfo.supportedByteWidths = supportedByteWidths;
2286         return *this;
2287     }
2288
2289     MemorySpaceBuilder& setReadDelegate(MemoryReadDelegate delegate)
2290     {
2291         info.readDelegate = delegate;
2292         return *this;
2293     }
2294
2295     MemorySpaceBuilder& setWriteDelegate(MemoryWriteDelegate delegate)
2296     {
2297         info.writeDelegate = delegate;
2298         return *this;
2299     }
2300
2301     MemorySpaceBuilder& setSidebandDelegate(MemoryGetSidebandInfoDelegate delegate)
2302     {
2303         info.sidebandDelegate = delegate;
2304         return *this;
2305     }
2306
2307     template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, uint64_t,
2308 uint64_t, const AttributeValueMap&, MemoryReadResult&)>
2309     MemorySpaceBuilder& setReadDelegate(T* instance)
2310     {
2311         return setReadDelegate(MemoryReadDelegate::make<T, METHOD>(instance));
2312     }
2313
2314     template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, uint64_t,
2315 uint64_t, const AttributeValueMap&, const uint64_t*, MemoryWriteResult&)>
2316     MemorySpaceBuilder& setWriteDelegate(T* instance)
2317     {
2318         return setWriteDelegate(MemoryWriteDelegate::make<T, METHOD>(instance));
2319     }
2320
2321     template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, const

```

```

IrisValueMap&, const std::vector<std::string>&, IrisValueMap&)>
2445     MemorySpaceBuilder& setSidebandDelegate(T* instance)
2446     {
2447         return setSidebandDelegate(MemoryGetSidebandInfoDelegate::make<T, METHOD>(instance));
2448     }
2449
2450     template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
2460         const AttributeValueMap&, MemoryReadResult&)>
2461     MemorySpaceBuilder& setReadDelegate()
2462     {
2463         return setReadDelegate(MemoryReadDelegate::make<FUNC>());
2464     }
2465
2466     template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
2477         const AttributeValueMap&, const uint64_t*, MemoryWriteResult&)>
2478     MemorySpaceBuilder& setWriteDelegate()
2479     {
2480         return setWriteDelegate(MemoryWriteDelegate::make<FUNC>());
2481     }
2482
2483     template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, const IrisValueMap&,
2494         const std::vector<std::string>&, IrisValueMap&)>
2495     MemorySpaceBuilder& setSidebandDelegate()
2496     {
2497         return setSidebandDelegate(MemoryGetSidebandInfoDelegate::make<FUNC>());
2498     }
2499
2500     MemorySpaceId getSpaceId() const
2501     {
2502         return info.spaceInfo.spaceId;
2503     }
2504 };
2505
2506 class AddressTranslationBuilder
2507 {
2508 private:
2509     IrisInstanceMemory::AddressTranslationInfoAndAccess& info;
2510
2511 public:
2512     AddressTranslationBuilder(IrisInstanceMemory::AddressTranslationInfoAndAccess& info_)
2513         : info(info_)
2514     {
2515     }
2516
2517     AddressTranslationBuilder& setTranslateDelegate(MemoryAddressTranslateDelegate delegate)
2518     {
2519         info.translateDelegate = delegate;
2520         return *this;
2521     }
2522
2523     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t, uint64_t, uint64_t,
2558     MemoryAddressTranslationResult&)>
2559     AddressTranslationBuilder& setTranslateDelegate(T* instance)
2560     {
2561         return setTranslateDelegate(MemoryAddressTranslateDelegate::make<T, METHOD>(instance));
2562     }
2563
2564     template <IrisErrorCode (*FUNC)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult&)>
2574     AddressTranslationBuilder& setTranslateDelegate()
2575     {
2576         return setTranslateDelegate(MemoryAddressTranslateDelegate::make<FUNC>());
2577     }
2578 };
2579
2580 void setPropertyCanonicalMsnScheme(const std::string& canonicalMsnScheme);
2594
2607 void setDefaultMemoryReadDelegate(MemoryReadDelegate delegate = MemoryReadDelegate())
2608 {
2609     inst_memory->setDefaultReadDelegate(delegate);
2610 }
2611
2612     template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, uint64_t,
2664     uint64_t, const AttributeValueMap&, MemoryReadResult&)>
2665     void setDefaultMemoryReadDelegate(T* instance)
2666     {
2667         setDefaultMemoryReadDelegate(MemoryReadDelegate::make<T, METHOD>(instance));
2668     }
2669
2670     template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
2695         const AttributeValueMap&, MemoryReadResult&)>
2696     void setDefaultMemoryReadDelegate()
2697     {
2698         setDefaultMemoryReadDelegate(MemoryReadDelegate::make<FUNC>());
2699     }
2700
2701     void setDefaultMemoryWriteDelegate(MemoryWriteDelegate delegate = MemoryWriteDelegate())
2735     {
2736

```

```

2737     inst_memory->setDefaultWriteDelegate(delegate);
2738 }
2739
2773 template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, uint64_t,
uint64_t, const AttributeValueMap&, const uint64_t*, MemoryWriteResult&)>
2774 void setDefaultMemoryWriteDelegate(T* instance)
2775 {
2776     setDefaultMemoryWriteDelegate(MemoryWriteDelegate::make<T, METHOD>(instance));
2777 }
2778
2804 template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
2805                                const AttributeValueMap&, const uint64_t*, MemoryWriteResult&)>
2806 void setDefaultMemoryWriteDelegate()
2807 {
2808     setDefaultMemoryWriteDelegate(MemoryWriteDelegate::make<FUNC>());
2809 }
2810
2829 MemorySpaceBuilder addMemorySpace(const std::string& name)
2830 {
2831     return MemorySpaceBuilder(inst_memory->addMemorySpace(name));
2832 }
2833
2865 void setDefaultAddressTranslateDelegate(MemoryAddressTranslateDelegate delegate =
MemoryAddressTranslateDelegate())
2866 {
2867     inst_memory->setDefaultTranslateDelegate(delegate);
2868 }
2869
2897 template <typename T, IrisErrorCode (T::*METHOD)(uint64_t, uint64_t, uint64_t,
MemoryAddressTranslationResult&)>
2898 void setDefaultAddressTranslateDelegate(T* instance)
2899 {
2900     setDefaultAddressTranslateDelegate(MemoryAddressTranslateDelegate::make<T, METHOD>(instance));
2901 }
2902
2922 template <IrisErrorCode (*FUNC)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult&)>
2923 void setDefaultAddressTranslateDelegate()
2924 {
2925     setDefaultAddressTranslateDelegate(MemoryAddressTranslateDelegate::make<FUNC>());
2926 }
2927
2944 AddressTranslationBuilder addAddressTranslation(MemorySpaceId inSpaceId, MemorySpaceId outSpaceId,
2945                                                const std::string& description)
2946 {
2947     return AddressTranslationBuilder(inst_memory->addAddressTranslation(inSpaceId, outSpaceId,
description));
2948 }
2949
2982 void setDefaultGetMemorySidebandInfoDelegate(MemoryGetSidebandInfoDelegate delegate)
2983 {
2984     inst_memory->setDefaultGetSidebandInfoDelegate(delegate);
2985 }
2986
3015 template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, const
IrisValueMap&, const std::vector<std::string>&, IrisValueMap&)>
3016 void setDefaultGetMemorySidebandInfoDelegate(T* instance)
3017 {
3018     setDefaultGetMemorySidebandInfoDelegate(MemoryGetSidebandInfoDelegate::make<T,
METHOD>(instance));
3019 }
3020
3041 template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, const IrisValueMap&,
3042                                const std::vector<std::string>&, IrisValueMap&)>
3043 void setDefaultGetMemorySidebandInfoDelegate()
3044 {
3045     setDefaultGetMemorySidebandInfoDelegate(MemoryGetSidebandInfoDelegate::make<FUNC>());
3046 }
3047
3082 void setLoadImageFileDelegate(ImageLoadFileDelegate delegate = ImageLoadFileDelegate())
3083 {
3084     inst_image->setLoadImageFileDelegate(delegate);
3085 }
3086
3107 template <typename T, IrisErrorCode (T::*METHOD)(const std::string&)>
3108 void setLoadImageFileDelegate(T* instance)
3109 {
3110     setLoadImageFileDelegate(ImageLoadFileDelegate::make<T, METHOD>(instance));
3111 }
3112
3125 template <IrisErrorCode (*FUNC)(const std::string&)>
3126 void setLoadImageFileDelegate()
3127 {
3128     setLoadImageFileDelegate(ImageLoadFileDelegate::make<FUNC>());
3129 }
3130
3155 void setLoadImageDataDelegate(ImageLoadDataDelegate delegate = ImageLoadDataDelegate())
3156 {

```



```

3157         inst_image->setLoadImageDataDelegate(delegate);
3158     }
3159
3160     template <typename T, IrisErrorCode (T::*METHOD)(const std::vector<uint8_t>&)>
3161     void setLoadImageDataDelegate(T* instance)
3162     {
3163         setLoadImageDataDelegate(ImageLoadDataDelegate::make<T, METHOD>(instance));
3164     }
3165
3166     template <IrisErrorCode (*FUNC)(const std::vector<uint8_t>&)>
3167     void setLoadImageDataDelegate()
3168     {
3169         setLoadImageDataDelegate(ImageLoadDataDelegate::make<FUNC>());
3170     }
3171
3172     uint64_t openImage(const std::string& filename)
3173     {
3174         return inst_image_cb->openImage(filename);
3175     }
3176
3177     void setRemainingStepSetDelegate(RemainingStepSetDelegate delegate = RemainingStepSetDelegate())
3178     {
3179         inst_step->setRemainingStepSetDelegate(delegate);
3180     }
3181
3182     void setRemainingStepGetDelegate(RemainingStepGetDelegate delegate)
3183     {
3184         inst_step->setRemainingStepGetDelegate(delegate);
3185     }
3186
3187     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t, const std::string&)>
3188     void setRemainingStepSetDelegate(T* instance)
3189     {
3190         setRemainingStepSetDelegate(RemainingStepSetDelegate::make<T, METHOD>(instance));
3191     }
3192
3193     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t&, const std::string&)>
3194     void setRemainingStepGetDelegate(T* instance)
3195     {
3196         setRemainingStepGetDelegate(RemainingStepGetDelegate::make<T, METHOD>(instance));
3197     }
3198
3199     template <IrisErrorCode (*FUNC)(uint64_t, const std::string&)>
3200     void setRemainingStepSetDelegate()
3201     {
3202         setRemainingStepSetDelegate(RemainingStepSetDelegate::make<FUNC>());
3203     }
3204
3205     template <IrisErrorCode (*FUNC)(uint64_t&, const std::string&)>
3206     void setRemainingStepGetDelegate()
3207     {
3208         setRemainingStepGetDelegate(RemainingStepGetDelegate::make<FUNC>());
3209     }
3210
3211     //
3212     void setStepCountGetDelegate(StepCountGetDelegate delegate = StepCountGetDelegate())
3213     {
3214         inst_step->setStepCountGetDelegate(delegate);
3215     }
3216
3217     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t&, const std::string&)>
3218     void setStepCountGetDelegate(T* instance)
3219     {
3220         setStepCountGetDelegate(RemainingStepGetDelegate::make<T, METHOD>(instance));
3221     }
3222
3223     template <IrisErrorCode (*FUNC)(uint64_t&, const std::string&)>
3224     void setStepCountGetDelegate()
3225     {
3226         setStepCountGetDelegate(RemainingStepGetDelegate::make<FUNC>());
3227     }
3228
3229     /*
3230      * @brief exec_apis IrisInstanceBuilder per-instance execution APIs
3231      * @{
3232      */
3233
3234     void setExecutionStateSetDelegate(PerInstanceExecutionStateSetDelegate delegate =
3235     PerInstanceExecutionStateSetDelegate())
3236     {
3237         inst_per_inst_exec->setExecutionStateSetDelegate(delegate);
3238     }
3239
3240     template <typename T, IrisErrorCode (T::*METHOD)(bool)>
3241     void setExecutionStateSetDelegate(T* instance)
3242     {
3243         setExecutionStateSetDelegate(PerInstanceExecutionStateSetDelegate::make<T, METHOD>(instance));
3244     }

```

```

3516     }
3517
3530     template <IrisErrorCode (*FUNC)(bool)>
3531     void setExecutionStateSetDelegate()
3532     {
3533         setExecutionStateSetDelegate(PerInstanceExecutionStateSetDelegate::make<FUNC>());
3534     }
3535
3560     void setExecutionStateGetDelegate(PerInstanceExecutionStateGetDelegate delegate)
3561     {
3562         inst_per_inst_exec->setExecutionStateGetDelegate(delegate);
3563     }
3564
3585     template <typename T, IrisErrorCode (T::*METHOD)(bool&)>
3586     void setExecutionStateGetDelegate(T* instance)
3587     {
3588         setExecutionStateGetDelegate(PerInstanceExecutionStateGetDelegate::make<T, METHOD>(instance));
3589     }
3590
3603     template <IrisErrorCode (*FUNC)(bool&)>
3604     void setExecutionStateGetDelegate()
3605     {
3606         setExecutionStateGetDelegate(PerInstanceExecutionStateGetDelegate::make<FUNC>());
3607     }
3608
3613     /*
3614     * @brief table_apis IrisInstanceBuilder table APIs
3615     * @{
3616     */
3617
3618     class TableColumnBuilder;
3619
3623     class TableBuilder
3624     {
3625     private:
3626         IrisInstanceTable::TableInfoAndAccess& info;
3627
3628     public:
3629         TableBuilder(IrisInstanceTable::TableInfoAndAccess& info_)
3630             : info(info_)
3631         {
3632         }
3633
3639         TableBuilder& setName(const std::string& name)
3640         {
3641             info.tableInfo.name = name;
3642             return *this;
3643         }
3644
3650         TableBuilder& setDescription(const std::string& description)
3651         {
3652             info.tableInfo.description = description;
3653             return *this;
3654         }
3655
3661         TableBuilder& setMinIndex(uint64_t minIndex)
3662         {
3663             info.tableInfo.minIndex = minIndex;
3664             return *this;
3665         }
3666
3672         TableBuilder& setMaxIndex(uint64_t maxIndex)
3673         {
3674             info.tableInfo.maxIndex = maxIndex;
3675             return *this;
3676         }
3677
3683         TableBuilder& setIndexFormatHint(const std::string& hint)
3684         {
3685             info.tableInfo.indexFormatHint = hint;
3686             return *this;
3687         }
3688
3694         TableBuilder& setFormatShort(const std::string& format)
3695         {
3696             info.tableInfo.formatShort = format;
3697             return *this;
3698         }
3699
3705         TableBuilder& setFormatLong(const std::string& format)
3706         {
3707             info.tableInfo.formatLong = format;
3708             return *this;
3709         }
3710
3720         TableBuilder& setReadDelegate(TableReadDelegate delegate)
3721         {

```

```

3722         info.readDelegate = delegate;
3723         return *this;
3724     }
3725
3726     TableBuilder& setWriteDelegate(TableWriteDelegate delegate)
3727     {
3728         info.writeDelegate = delegate;
3729         return *this;
3730     }
3731
3732     template <typename T, IrisErrorCode (T::*METHOD)(const TableInfo&, uint64_t, uint64_t,
3733 TableReadResult&)>
3734     TableBuilder& setReadDelegate(T* instance)
3735     {
3736         return setReadDelegate(TableReadDelegate::make<T, METHOD>(instance));
3737     }
3738
3739     template <typename T, IrisErrorCode (T::*METHOD)(const TableInfo&, const TableRecords&,
3740 TableWriteResult&)>
3741     TableBuilder& setWriteDelegate(T* instance)
3742     {
3743         return setWriteDelegate(TableWriteDelegate::make<T, METHOD>(instance));
3744     }
3745
3746     template <IrisErrorCode (*FUNC)(const TableInfo&, uint64_t, uint64_t, TableReadResult&)>
3747     TableBuilder& setReadDelegate()
3748     {
3749         return setReadDelegate(TableReadDelegate::make<FUNC>());
3750     }
3751
3752     template <IrisErrorCode (*FUNC)(const TableInfo&, const TableRecords&, TableWriteResult&)>
3753     TableBuilder& setWriteDelegate()
3754     {
3755         return setWriteDelegate(TableWriteDelegate::make<FUNC>());
3756     }
3757
3758     TableBuilder& addColumnInfo(const TableColumnInfo& columnInfo)
3759     {
3760         info.tableInfo.columns.push_back(columnInfo);
3761         return *this;
3762     }
3763
3764     TableColumnBuilder addColumn(const std::string& name);
3765 };
3766
3767 class TableColumnBuilder
3768 {
3769 private:
3770     TableBuilder& parent;
3771     TableColumnInfo& info;
3772
3773 public:
3774     TableColumnBuilder(TableBuilder& parent_, TableColumnInfo& info_)
3775         : parent(parent_)
3776         , info(info_)
3777     {
3778     }
3779
3780     TableBuilder& addColumnInfo(const TableColumnInfo& columnInfo)
3781     {
3782         return parent.addColumnInfo(columnInfo);
3783     }
3784
3785     TableColumnBuilder addColumn(const std::string& name) { return parent.addColumn(name); }
3786
3787     TableBuilder& endColumn()
3788     {
3789         return parent;
3790     }
3791
3792     TableColumnBuilder& setName(const std::string& name)
3793     {
3794         info.name = name;
3795         return *this;
3796     }
3797
3798     TableColumnBuilder& setDescription(const std::string& description)
3799     {
3800         info.description = description;
3801         return *this;
3802     }
3803
3804     TableColumnBuilder& setFormat(const std::string& format)
3805     {
3806         info.format = format;
3807         return *this;
3808     }
3809 }

```

```

3926
3933     TableColumnBuilder& setType(const std::string& type)
3934     {
3935         info.type = type;
3936         return *this;
3937     }
3938
3939     TableColumnBuilder& setBitWidth(uint64_t bitWidth)
3940     {
3941         info.bitWidth = bitWidth;
3942         return *this;
3943     }
3944
3945     TableColumnBuilder& setFormatShort(const std::string& format)
3946     {
3947         info.formatShort = format;
3948         return *this;
3949     }
3950
3951     TableColumnBuilder& setFormatLong(const std::string& format)
3952     {
3953         info.formatLong = format;
3954         return *this;
3955     }
3956
3957     TableColumnBuilder& setRwMode(const std::string& rwMode)
3958     {
3959         info.rwMode = rwMode;
3960         return *this;
3961     }
3962
3963     };
3964
3965     TableBuilder addTable(const std::string& name)
3966     {
3967         return TableBuilder(inst_table->addTableInfo(name));
3968     }
3969
3970     void setDefaultTableReadDelegate(TableReadDelegate delegate = TableReadDelegate())
3971     {
3972         inst_table->setDefaultReadDelegate(delegate);
3973     }
3974
3975     void setDefaultTableWriteDelegate(TableWriteDelegate delegate = TableWriteDelegate())
3976     {
3977         inst_table->setDefaultWriteDelegate(delegate);
3978     }
3979
3980     template <typename T, IrisErrorCode (T::*METHOD)(const TableInfo&, uint64_t, uint64_t,
3981     TableReadResult&)>
3982     void setDefaultTableReadDelegate(T* instance)
3983     {
3984         setDefaultTableReadDelegate(TableReadDelegate::make<T, METHOD>(instance));
3985     }
3986
3987     template <typename T, IrisErrorCode (T::*METHOD)(const TableInfo&, const TableRecords&,
3988     TableWriteResult&)>
3989     void setDefaultTableWriteDelegate(T* instance)
3990     {
3991         setDefaultTableWriteDelegate(TableWriteDelegate::make<T, METHOD>(instance));
3992     }
3993
3994     template <IrisErrorCode (*FUNC)(const TableInfo&, uint64_t, uint64_t, TableReadResult&)>
3995     void setDefaultTableReadDelegate()
3996     {
3997         setDefaultTableReadDelegate(TableReadDelegate::make<FUNC>());
3998     }
3999
4000     template <IrisErrorCode (*FUNC)(const TableInfo&, const TableRecords&, TableWriteResult&)>
4001     void setDefaultTableWriteDelegate()
4002     {
4003         setDefaultTableWriteDelegate(TableWriteDelegate::make<FUNC>());
4004     }
4005
4006     void setGetCurrentDisassemblyModeDelegate(GetCurrentDisassemblyModeDelegate delegate)
4007     {
4008         inst_disass->setGetCurrentModeDelegate(delegate);
4009     }
4010
4011     template <typename T, IrisErrorCode (T::*METHOD)(std::string&)>
4012     void setGetCurrentDisassemblyModeDelegate(T* instance)
4013     {
4014         setGetCurrentDisassemblyModeDelegate(GetCurrentDisassemblyModeDelegate::make<T,
4015     METHOD>(instance));
4016     }
4017
4018     void setGetDisassemblyDelegate(GetDisassemblyDelegate delegate)
4019     {
4020

```

```

4224         inst_disass->setGetDisassemblyDelegate(delegate);
4225     }
4226
4227     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t, const std::string&, MemoryReadResult&,
uint64_t, uint64_t, std::vector<DisassemblyLine>&)>
4228     void setGetDisassemblyDelegate(T* instance)
4229     {
4230         setGetDisassemblyDelegate(GetDisassemblyDelegate::make<T, METHOD>(instance));
4231     }
4232
4233     template <IrisErrorCode (*FUNC)(uint64_t, const std::string&, MemoryReadResult&,
uint64_t, uint64_t, std::vector<DisassemblyLine>&)>
4234     void setGetDisassemblyDelegate()
4235     {
4236         setGetDisassemblyDelegate(GetDisassemblyDelegate::make<FUNC>());
4237     }
4238
4239     void setDisassembleOpcodeDelegate(DisassembleOpcodeDelegate delegate)
4240     {
4241         inst_disass->setDisassembleOpcodeDelegate(delegate);
4242     }
4243
4244     template <typename T, IrisErrorCode (T::*METHOD)(const std::vector<uint64_t>&, uint64_t, const
std::string&, DisassembleContext&, DisassemblyLine&)>
4245     void setDisassembleOpcodeDelegate(T* instance)
4246     {
4247         setDisassembleOpcodeDelegate(DisassembleOpcodeDelegate::make<T, METHOD>(instance));
4248     }
4249
4250     template <IrisErrorCode (*FUNC)(const std::vector<uint64_t>&, uint64_t, const std::string&,
DisassembleContext&, DisassemblyLine&)>
4251     void setDisassembleOpcodeDelegate()
4252     {
4253         setDisassembleOpcodeDelegate(DisassembleOpcodeDelegate::make<FUNC>());
4254     }
4255
4256     void addDisassemblyMode(const std::string& name, const std::string& description)
4257     {
4258         inst_disass->addDisassemblyMode(name, description);
4259     }
4260
4261     void setDbgStateSetRequestDelegate(DebuggableStateSetRequestDelegate delegate =
DebuggableStateSetRequestDelegate())
4262     {
4263         inst_dbg_state->setSetRequestDelegate(delegate);
4264     }
4265
4266     template <typename T, IrisErrorCode (T::*METHOD)(bool)>
4267     void setDbgStateSetRequestDelegate(T* instance)
4268     {
4269         setDbgStateSetRequestDelegate(DebuggableStateSetRequestDelegate::make<T, METHOD>(instance));
4270     }
4271
4272     template <IrisErrorCode (*FUNC)(bool)>
4273     void setDbgStateSetRequestDelegate()
4274     {
4275         setDbgStateSetRequestDelegate(DebuggableStateSetRequestDelegate::make<FUNC>());
4276     }
4277
4278     void setDbgStateGetAcknowledgeDelegate(DebuggableStateGetAcknowledgeDelegate delegate =
DebuggableStateGetAcknowledgeDelegate())
4279     {
4280         inst_dbg_state->setGetAcknowledgeDelegate(delegate);
4281     }
4282
4283     template <typename T, IrisErrorCode (T::*METHOD)(bool&)>
4284     void setDbgStateGetAcknowledgeDelegate(T* instance)
4285     {
4286         setDbgStateGetAcknowledgeDelegate(DebuggableStateGetAcknowledgeDelegate::make<T,
METHOD>(instance));
4287     }
4288
4289     template <IrisErrorCode (*FUNC)(bool&)>
4290     void setDbgStateGetAcknowledgeDelegate()
4291     {
4292         setDbgStateGetAcknowledgeDelegate(DebuggableStateGetAcknowledgeDelegate::make<FUNC>());
4293     }
4294
4295     template <typename T, IrisErrorCode (T::*SET_REQUEST)(bool), IrisErrorCode
(T::*GET_ACKNOWLEDGE)(bool&)>
4296     void setDbgStateDelegates(T* instance)
4297     {
4298         setDbgStateSetRequestDelegate<T, SET_REQUEST>(instance);
4299         setDbgStateGetAcknowledgeDelegate<T, GET_ACKNOWLEDGE>(instance);
4300     }
4301
4302     void setCheckpointSaveDelegate(CheckpointSaveDelegate delegate = CheckpointSaveDelegate())

```

```

4456     {
4457         inst_checkpoint->setCheckpointSaveDelegate(delegate);
4458     }
4459
4460     template <typename T, IrisErrorCode (T::*METHOD)(const std::string&)>
4461     void setCheckpointSaveDelegate(T* instance)
4462     {
4463         setCheckpointSaveDelegate(CheckpointSaveDelegate::make<T, METHOD>(instance));
4464     }
4465
4466     void setCheckpointRestoreDelegate(CheckpointRestoreDelegate delegate = CheckpointRestoreDelegate())
4467     {
4468         inst_checkpoint->setCheckpointRestoreDelegate(delegate);
4469     }
4470
4471     template <typename T, IrisErrorCode (T::*METHOD)(const std::string&)>
4472     void setCheckpointRestoreDelegate(T* instance)
4473     {
4474         setCheckpointRestoreDelegate(CheckpointRestoreDelegate::make<T, METHOD>(instance));
4475     }
4476
4477     class SemihostingManager
4478     {
4479     private:
4480         IrisInstanceSemihosting* inst_semihost;
4481
4482     public:
4483         SemihostingManager(IrisInstanceSemihosting* inst_semihost_)
4484             : inst_semihost(inst_semihost_)
4485         {
4486         }
4487
4488         ~SemihostingManager()
4489         {
4490             // Interrupt any requests that are currently blocked
4491             unblock();
4492         }
4493
4494         void enableExtensions()
4495         {
4496             inst_semihost->enableExtensions();
4497         }
4498
4499         std::vector<uint8_t> readData(uint64_t fDes, size_t max_size = 0, uint64_t flags =
4500             semihost::DEFAULT)
4501         {
4502             return inst_semihost->readData(fDes, max_size, flags);
4503         }
4504
4505         /*
4506          * @brief Write data for a given file descriptor
4507          *
4508          * @param fDes      File descriptor to write to. Usually semihost::STDOUT or
4509             semihost::STDERR.
4510          * @param data      Buffer containing the data to write.
4511          * @param size      Size of the data buffer in bytes.
4512          * @return          Returns false if no client is registered for IRIS_SEMIHOSTING_OUTPUT
4513             events.
4514          */
4515         bool writeData(uint64_t fDes, const uint8_t* data, size_t size)
4516         {
4517             return inst_semihost->writeData(fDes, data, size);
4518         }
4519
4520         /*
4521          * @brief Write data for a given file descriptor
4522          *
4523          * @param fDes      File descriptor to write to. Usually semihost::STDOUT or
4524             semihost::STDERR.
4525          * @param data      Buffer containing the data to write.
4526          * @return          Returns false if no client is registered for IRIS_SEMIHOSTING_OUTPUT
4527             events.
4528          */
4529         bool writeData(uint64_t fDes, const std::vector<uint8_t>& data)
4530         {
4531             return writeData(fDes, &data.front(), data.size());
4532         }
4533
4534         std::pair<bool, uint64_t> semihostedCall(uint64_t operation, uint64_t parameter)
4535         {
4536             return inst_semihost->semihostedCall(operation, parameter);
4537         }
4538
4539         /*
4540          * @brief Request premature exit from any blocking requests that are currently blocked.
4541          */
4542         void unblock()

```

```

4582         {
4583             return inst_semihost->unblock();
4584         }
4585     };
4586
4594     SemihostingManager enableSemihostingAndGetManager()
4595     {
4596         inst_semihost.init();
4597         return SemihostingManager(inst_semihost);
4598     }
4599
4603 };
4604
4605 inline IrisInstanceBuilder::TableColumnBuilder IrisInstanceBuilder::TableBuilder::addColumn(const
std::string& name)
4606 {
4607     // Add a new column with default info
4608     info.tableInfo.columns.resize(info.tableInfo.columns.size() + 1);
4609     TableColumnInfo& col = info.tableInfo.columns.back();
4610
4611     col.name = name;
4612
4613     return TableColumnBuilder(*this, col);
4614 }
4615
4616 NAMESPACE_IRIS_END
4617
4618 #endif // ARM_INCLUDE_IrisInstanceBuilder_h

```

9.21 IrisInstanceCheckpoint.h File Reference

Checkpoint add-on to IrisInstance.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"

```

Classes

- class [iris::IrisInstanceCheckpoint](#)
Checkpoint add-on for [IrisInstance](#).

Typedefs

- typedef [IrisDelegate< const std::string & > iris::CheckpointRestoreDelegate](#)
Restore the checkpoint corresponding to the given information.
- typedef [IrisDelegate< const std::string & > iris::CheckpointSaveDelegate](#)
Save a checkpoint corresponding to the given information.

9.21.1 Detailed Description

Checkpoint add-on to IrisInstance.

Date

Copyright ARM Limited 2019 All Rights Reserved.

9.21.2 Typedef Documentation

9.21.2.1 CheckpointRestoreDelegate

```
typedef IrisDelegate<const std::string&> iris::CheckpointRestoreDelegate
```

Restore the checkpoint corresponding to the given information.

```
IrisErrorCode checkpoint_restore(const std::string & checkpoint_dir)
```

Error: Return E_* error code if it failed to restore the checkpoint.

9.21.2.2 CheckpointSaveDelegate

typedef IrisDelegate<const std::string&> [iris::CheckpointSaveDelegate](#)

Save a checkpoint corresponding to the given information.

IrisErrorCode checkpoint_save(const std::string & checkpoint_dir)

Error: Return E_* error code if it failed to save the checkpoint.

9.22 IrisInstanceCheckpoint.h

[Go to the documentation of this file.](#)

```

1
2
3
4
5
6
7 #ifndef ARM_INCLUDE_IrisInstanceCheckpoint_h
8 #define ARM_INCLUDE_IrisInstanceCheckpoint_h
9
10 #include "iris/detail/IrisCommon.h"
11 #include "iris/detail/IrisDelegate.h"
12
13 NAMESPACE_IRIS_START
14
15 class IrisInstance;
16 class IrisReceivedRequest;
17
18 typedef IrisDelegate<const std::string&> CheckpointSaveDelegate;
19
20 typedef IrisDelegate<const std::string&> CheckpointRestoreDelegate;
21
22 class IrisInstanceCheckpoint
23 {
24 public:
25     IrisInstanceCheckpoint(IrisInstance* iris_instance = nullptr);
26
27     void attachTo(IrisInstance* iris_instance_);
28
29     void setCheckpointSaveDelegate(CheckpointSaveDelegate delegate);
30
31     void setCheckpointRestoreDelegate(CheckpointRestoreDelegate delegate);
32 private:
33     void impl_checkpoint_save(IrisReceivedRequest& request);
34
35     void impl_checkpoint_restore(IrisReceivedRequest& request);
36
37     IrisInstance* iris_instance;
38
39     CheckpointSaveDelegate save_delegate;
40
41     CheckpointRestoreDelegate restore_delegate;
42 };
43
44 NAMESPACE_IRIS_END
45 #endif // #ifndef ARM_INCLUDE_IrisInstanceCheckpoint_h

```

9.23 IrisInstanceDebuggableState.h File Reference

IrisInstance add-on to implement debuggableState functions.

```
#include "iris/detail/IrisCommon.h"
```

```
#include "iris/detail/IrisDelegate.h"
```

Classes

- class [iris::IrisInstanceDebuggableState](#)
Debuggable-state add-on for [IrisInstance](#).

Typedefs

- typedef IrisDelegate< bool & > [iris::DebuggableStateGetAcknowledgeDelegate](#)
Interface to stop the simulation time progress.
- typedef IrisDelegate< bool > [iris::DebuggableStateSetRequestDelegate](#)

Delegate to set the debuggable-state-request flag.

9.23.1 Detailed Description

IrisInstance add-on to implement debuggableState functions.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.23.2 Typedef Documentation

9.23.2.1 DebuggableStateGetAcknowledgeDelegate

```
typedef IrisDelegate<bool&> iris::DebuggableStateGetAcknowledgeDelegate
```

Interface to stop the simulation time progress.

```
IrisErrorCode getAcknowledge(bool &acknowledge_out);
```

9.23.2.2 DebuggableStateSetRequestDelegate

```
typedef IrisDelegate<bool> iris::DebuggableStateSetRequestDelegate
```

Delegate to set the debuggable-state-request flag.

```
IrisErrorCode setRequest(bool request);
```

9.24 IrisInstanceDebuggableState.h

[Go to the documentation of this file.](#)

```
1
2
3 #ifndef ARM_INCLUDE_IrisInstanceDebuggableState_h
4 #define ARM_INCLUDE_IrisInstanceDebuggableState_h
5
6 #include "iris/detail/IrisCommon.h"
7 #include "iris/detail/IrisDelegate.h"
8
9 NAMESPACE_IRIS_START
10
11 typedef IrisDelegate<bool> DebuggableStateSetRequestDelegate;
12
13 typedef IrisDelegate<bool&> DebuggableStateGetAcknowledgeDelegate;
14
15 class IrisInstance;
16 class IrisReceivedRequest;
17
18 class IrisInstanceDebuggableState
19 {
20 private:
21     IrisInstance* iris_instance;
22
23     DebuggableStateSetRequestDelegate setRequest;
24     DebuggableStateGetAcknowledgeDelegate getAcknowledge;
25
26 public:
27     IrisInstanceDebuggableState(IrisInstance* iris_instance = nullptr);
28
29     void attachTo(IrisInstance* irisInstance);
30
31     void setSetRequestDelegate(DebuggableStateSetRequestDelegate delegate)
32     {
33         setRequest = delegate;
34     }
35
36     void setGetAcknowledgeDelegate(DebuggableStateGetAcknowledgeDelegate delegate)
37     {
38         getAcknowledge = delegate;
39     }
40
41 private:
42     void impl_debuggableState_setRequest(IrisReceivedRequest& request);
43
44     void impl_debuggableState_getAcknowledge(IrisReceivedRequest& request);
45 };
46
```

```

89
90 NAMESPACE_IRIS_END
91
92 #endif // ARM_INCLUDE_IrisInstanceSimulationTime_h

```

9.25 IrisInstanceDisassembler.h File Reference

Disassembler add-on to IrisInstance.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include <cstdint>

```

Classes

- class [iris::IrisInstanceDisassembler](#)
Disassembler add-on for [IrisInstance](#).

Typedefs

- typedef IrisDelegate< const std::vector< uint64_t > &, uint64_t, const std::string &, DisassembleContext &, DisassemblyLine & > [iris::DisassembleOpcodeDelegate](#)
Get the disassembly for an individual opcode.
- typedef IrisDelegate< std::string & > [iris::GetCurrentDisassemblyModeDelegate](#)
Get the current disassembly mode.
- typedef IrisDelegate< uint64_t, const std::string &, MemoryReadResult &, uint64_t, uint64_t, std::vector< DisassemblyLine > & > [iris::GetDisassemblyDelegate](#)
Get the disassembly of a chunk of memory.

9.25.1 Detailed Description

Disassembler add-on to IrisInstance.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

The IrisInstanceDisassembler class implements all disassembly-related Iris functions.

9.26 IrisInstanceDisassembler.h

[Go to the documentation of this file.](#)

```

1
9 #ifndef ARM_INCLUDE_IrisInstanceDisassembler_h
10 #define ARM_INCLUDE_IrisInstanceDisassembler_h
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisDelegate.h"
14 #include "iris/detail/IrisLogger.h"
15 #include "iris/detail/IrisObjects.h"
16
17 #include <cstdint>
18
19 NAMESPACE_IRIS_START
20
21 class IrisInstance;
22 class IrisReceivedRequest;
23
24 typedef IrisDelegate<std::string&> GetCurrentDisassemblyModeDelegate;
25
26
27 typedef IrisDelegate<uint64_t, const std::string&, MemoryReadResult&,
28                     uint64_t, uint64_t, std::vector<DisassemblyLine>&>
29     GetDisassemblyDelegate;

```

```

54
55 typedef IrisDelegate<const std::vector<uint64_t>&, uint64_t, const std::string&,
56                     DisassembleContext&, DisassemblyLine&>
57     DisassembleOpcodeDelegate;
58
59 /*
60 * @}
61 */
62
63 class IrisInstanceDisassembler
64 {
65 public:
66     IrisInstanceDisassembler(IrisInstance* irisInstance = nullptr);
67
68     void attachTo(IrisInstance* irisInstance);
69
70     void setGetCurrentModeDelegate(GetCurrentDisassemblyModeDelegate delegate)
71     {
72         getCurrentMode = delegate;
73     }
74
75     void setGetDisassemblyDelegate(GetDisassemblyDelegate delegate)
76     {
77         getDisassembly = delegate;
78     }
79
80     void setDisassembleOpcodeDelegate(DisassembleOpcodeDelegate delegate)
81     {
82         disassembleOpcode = delegate;
83     }
84
85     void addDisassemblyMode(const std::string& name, const std::string& description);
86
87 private:
88     void impl_disassembler_getModes(IrisReceivedRequest& request);
89
90     void impl_disassembler_getCurrentMode(IrisReceivedRequest& request);
91
92     void impl_disassembler_getDisassembly(IrisReceivedRequest& request);
93
94     void impl_disassembler_disassembleOpcode(IrisReceivedRequest& request);
95
96     void checkDisassemblyMode(std::string& mode, bool& isValidMode);
97
98     IrisInstance* irisInstance;
99
100     GetCurrentDisassemblyModeDelegate getCurrentMode;
101
102     GetDisassemblyDelegate getDisassembly;
103
104     DisassembleOpcodeDelegate disassembleOpcode;
105
106     std::vector<DisassemblyMode> disassemblyModes;
107     IrisLogger log;
108 };
109
110 namespace IRIS
111 {
112     IRIS_END_NAMESPACE
113 }
114
115 #endif // #ifndef ARM_INCLUDE_IrisInstanceDisassembler_h

```

9.27 IrisInstanceEvent.h File Reference

Event add-on to IrisInstance.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include "iris/detail/IrisRequest.h"
#include <cstdio>
#include <set>

```

Classes

- struct [iris::IrisInstanceEvent::EventSourceInfoAndDelegate](#)
Contains the metadata and delegates for a single EventSource.

- class [iris::EventStream](#)
Base class for event streams.
- class [iris::IrisEventRegistry](#)
Class to register Iris event streams for an event.
- class [iris::IrisEventStream](#)
Event stream class for Iris-specific events.
- class [iris::IrisInstanceEvent](#)
Event add-on for [IrisInstance](#).
- struct [iris::IrisInstanceEvent::ProxyEventInfo](#)
Contains information for a single proxy EventSource.

Typedefs

- typedef `IrisDelegate< EventStream *&, const EventSourceInfo &, const std::vector< std::string > & >`
[iris::EventStreamCreateDelegate](#)
Delegate to create an [EventStream](#).

9.27.1 Detailed Description

Event add-on to [IrisInstance](#).

Copyright

Copyright (C) 2016-2023 Arm Limited. All rights reserved.

The [IrisInstanceEvent](#) class:

- Implements all event-related Iris functions.
- Maintains and provides event source metadata.
- Converts between Iris event functions (`event*()`) and various C++ access functions.

9.27.2 Typedef Documentation

9.27.2.1 EventStreamCreateDelegate

```
typedef IrisDelegate<EventStream*&, const EventSourceInfo&, const std::vector<std::string>&>
iris::EventStreamCreateDelegate
```

Delegate to create an [EventStream](#).

```
IrisErrorCode create(EventStream *&evStream, const EventSourceInfo &srcInfo, const std::vector<std::string>
&fields)
```

Create a new event stream with the specified fields for an event source.

The new event stream is maintained and destroyed in the event add-on.

Error: Return `E_*` error code, for example `E_unknown_event_field`, if the event stream could not be created.

9.28 IrisInstanceEvent.h

[Go to the documentation of this file.](#)

```
1
12 #ifndef ARM_INCLUDE_IrisInstanceEvent_h
13 #define ARM_INCLUDE_IrisInstanceEvent_h
14
15 #include "iris/detail/IrisCommon.h"
16 #include "iris/detail/IrisDelegate.h"
17 #include "iris/detail/IrisLogger.h"
18 #include "iris/detail/IrisObjects.h"
19 #include "iris/detail/IrisRequest.h"
20
21 #include <cstdio>
22 #include <set>
```

```

23
24 NAMESPACE_IRIS_START
25
26 class IrisInstance;
27 class IrisReceivedRequest;
28
29 class EventStream;
30 class IrisEventRegistry;
31
44 typedef IrisDelegate<EventStream*&, const EventSourceInfo&, const std::vector<std::string>&>
    EventStreamCreateDelegate;
45
63 class IrisInstanceEvent
64 {
65 public:
66
67     /* ! What is a proxy event source?
68      - The event source in actual does not belong to this Iris instance, but instead belongs to another
        Iris instance (target).
69      - The event source is registered as a proxy in this Iris instance using Iris interface -
        event_registerProxyEventSource()
70      - This Iris instance acts as a proxy for those registered events.
71      - All the interface calls (for example, eventStream_create) on the proxy event source are forwarded to
        the target instance.
72      - Similarly, all the created event streams in this Iris instance for the proxy event source are
        tagged as proxyForOtherInstance
73      - All the interface calls (for example, eventStream_enable) on such proxy event streams are
        forwarded to the target instance.
74      - Finally, the proxy event source can be deregistered using Iris interface -
        event_unregisterProxyEventSource()
75     */
80 struct ProxyEventInfo
81 {
82     InstanceId targetInstId{}; //target Iris instance Id
83     EventSourceId targetEvSrcId{}; //event source ID in target Iris instance
84     std::vector<EventStreamId> evStreamIds; //list of created event stream IDs
85     //Important note: When we create an event stream, we use the same esID for both - this and target
    Iris instance
86 };
87
91 struct EventSourceInfoAndDelegate
92 {
93     EventSourceInfo info;
94     EventStreamCreateDelegate createEventStream;
95
96     bool isValid{true}; //deleteEventSource() sets isValid to false
97     bool isProxy{false};
98     ProxyEventInfo proxyEventInfo; //contains proper values only if isProxy=true
99 };
100
106 IrisInstanceEvent(IrisInstance* irisInstance = nullptr);
107 ~IrisInstanceEvent();
108
116 void attachTo(IrisInstance* irisInstance);
117
125 void setDefaultEsCreateDelegate(EventStreamCreateDelegate delegate);
126
139 EventSourceInfoAndDelegate& addEventSource(const std::string& name, bool isHidden = false);
140
148 uint64_t addEventSource(const EventSourceInfoAndDelegate& info);
149
158 EventSourceInfoAndDelegate& enhanceEventSource(const std::string& name);
159
168 void renameEventSource(const std::string& name, const std::string& newName);
169
175 void deleteEventSource(const std::string& eventName);
176
183 bool hasEventSource(const std::string& eventName);
184
192 const uint64_t *eventBufferGetSyncStepResponse(EventBufferId evBufId, RequestId requestId);
193
202 void eventBufferClear(EventBufferId evBufId);
203
211 bool isValidEvBufId(EventBufferId evBufId) const;
212
213
222 bool destroyEventStream(EventStreamId esId);
223
234 void destroyAllEventStreams();
235
241 const EventSourceInfo *getEventSourceInfo(EventSourceId evSrcId) const;
242
243 private:
244     // --- Iris function implementations ---
245
246     void impl_event_getEventSources(IrisReceivedRequest& request);

```

```

247
248 void impl_event_getEventSource(IrisReceivedRequest& request);
249
250 void impl_eventStream_create(IrisReceivedRequest& request);
251
252 void impl_eventStream_destroy(IrisReceivedRequest& request);
253
254 void impl_eventStream_destroyAll(IrisReceivedRequest& request);
255
256 void impl_eventStream_enable(IrisReceivedRequest& request);
257
258 void impl_eventStream_disable(IrisReceivedRequest& request);
259
260 void impl_eventStream_getCounter(IrisReceivedRequest& request);
261
262 void impl_eventStream_setTraceRanges(IrisReceivedRequest& request);
263
264 void impl_eventStream_getState(IrisReceivedRequest& request);
265
266 void impl_eventStream_flush(IrisReceivedRequest& request);
267
268 void impl_eventStream_setOptions(IrisReceivedRequest& request);
269
270 void impl_eventStream_action(IrisReceivedRequest& request);
271
272 void impl_eventBuffer_create(IrisReceivedRequest& request);
273
274 void impl_eventBuffer_flush(IrisReceivedRequest& request);
275
276 void impl_eventBuffer_destroy(IrisReceivedRequest& request);
277
278 void impl_ec_eventBuffer(IrisReceivedRequest& request);
279
280 void register_ec_IRIS_INSTANCE_REGISTRY_CHANGED();
281 IrisErrorCode ec_IRIS_INSTANCE_REGISTRY_CHANGED(EventStreamId esId, const IrisValueMap& fields,
uint64_t time,
282                                     InstanceId sInstId, bool syncEc, std::string&
errorMessageOut);
283
284
285
286 void impl_event_registerProxyEventSource(IrisReceivedRequest& request);
287
288 void impl_event_unregisterProxyEventSource(IrisReceivedRequest& request);
289
290 void impl_eventStream_create_proxy(IrisReceivedRequest& request);
291
292 IrisErrorCode impl_eventStream_destroy_target(IrisReceivedRequest& request, EventStream* evStream);
293
294 void impl_eventStream_enable_proxy(IrisReceivedRequest& request, EventStream* evStream);
295
296 void impl_eventStream_disable_proxy(IrisReceivedRequest& request, EventStream* evStream);
297
298 void impl_eventStream_getCounter_proxy(IrisReceivedRequest& request, EventStream* evStream);
299
300 void impl_eventStream_setTraceRanges_proxy(IrisReceivedRequest& request, EventStream* evStream);
301
302 void impl_eventStream_getState_proxy(IrisReceivedRequest& request, EventStream* evStream);
303
304 void impl_eventStream_flush_proxy(IrisReceivedRequest& request, EventStream* evStream);
305
306 void impl_eventStream_setOptions_proxy(IrisReceivedRequest& request, EventStream* evStream);
307
308 void impl_eventStream_action_proxy(IrisReceivedRequest& request, EventStream* evStream);
309
310 ProxyEventInfo& getProxyEventInfo(EventStream* evStream);
311
312 InstanceId getTargetInstId(EventStream* evStream);
313
314
315
316 EventStream* getEventStream(EventStreamId esId);
317
318 struct EventBufferStreamInfo;
319 struct EventBuffer;
320
321
322 const EventBufferStreamInfo* getEventBufferStreamInfo(InstanceId sInstId, EventStreamId esId) const;
323
324
325 EventBuffer* getEventBuffer(EventBufferId evBufId) const;
326
327
328 void eventBufferSend(EventBuffer* eventBuffer, bool flush);
329
330
331 void eventBufferDestroy(EventBufferId evBufId);
332
333
334
335 //Find a free event stream ID where a new EventStream can be added
336 //The returned ID is greater than or equal to 'minEsId'
337 EventStreamId findFreeEventStreamId(EventStreamId minEsId);
338
339
340

```

```

342     IrisInstance* irisInstance;
343
344     std::vector<EventSourceInfoAndDelegate> eventSources;
345
346     std::map<std::string, uint64_t> srcNameToId;
347
348     std::vector<EventStream*> eventStreams;
349
350     std::vector<EventStreamId> freeEsIds;
351
352     EventStreamCreateDelegate defaultEsCreateDelegate;
353
354     IrisLogger log;
355
356     bool instance_registry_changed_registered{};
357
358     struct EventStreamOriginInfo
359     {
360         EventStreamId esId;
361         InstanceId sInstId;
362     };
363
364     struct EventBuffer
365     {
366         EventBuffer(const std::string& mode, uint64_t bufferSize, const std::string& ebcFunc, InstanceId
367         ebcInstId, bool syncEbc, EventBufferId evBufId, IrisInstanceEvent *parent);
368
369         ~EventBuffer();
370
371         void clear();
372
373         const uint64_t* getResponse(RequestId requestId);
374
375         void getRequest(bool flush);
376
377         void addEventData(EventStreamInfoId esInfoId, uint64_t time, const uint64_t *fieldsU64Json);
378
379         void dropOldEvents(uint64_t targetBufferSizeU64);
380
381         std::string mode;
382
383         uint64_t bufferSizeU64{};
384
385         std::string ebcFunc;
386
387         InstanceId ebcInstId{IRIS_UINT64_MAX};
388
389         bool syncEbc{};
390
391         std::vector<EventStreamOriginInfo> eventStreams;
392
393         IrisU64JsonWriter writer;
394
395         uint64_t numEvents{};
396
397         size_t eventDataStartPos{};
398
399         IrisU64JsonWriter responseHeader;
400         size_t responseStartPos{};
401         size_t responseObjectPos{};
402         size_t responseArrayPos{};
403
404         IrisU64JsonWriter requestHeader;
405         size_t requestStartPos{};
406         size_t requestParamsPos{};
407         size_t requestReasonPos{};
408         size_t requestArrayPos{};
409
410         const uint64_t reasonSend = 0x200000646E657304; // == "send"
411         const uint64_t reasonFlush = 0x20006873756C6605; // == "flush"
412
413         IrisInstanceEvent *parent{};
414     };
415     friend struct EventBuffer;
416
417     std::vector<EventBuffer*> eventBuffers;
418
419     std::vector<EventBufferId> freeEventBufferIds;
420
421     struct EventBufferStreamInfo
422     {
423         EventBuffer* eventBuffer{};
424         EventStreamInfoId esInfoId{};
425     };
426
427     std::vector<std::vector<EventBufferStreamInfo>> eventCallbackInfoToEventBufferStreamInfo;

```

```

504     bool inEventStreamCreate{};
505 };
506
512 class EventStream
513 {
514 public:
515     EventStream()
516     {
517     }
518
519     virtual ~EventStream()
520     {
521         // Detach fieldObj from writer contained in internal_req so it does not touch
522         // internal_req after it was deleted.
523         //
524         // Background:
525         // IrisEventRegistry first calls emitEventBegin() on all event streams and one
526         // of the callbacks may lead to the destruction of the destination instance which
527         // will destroy all event streams, including the ones which had emitEventBegin()
528         // called on them without matching emitEventEnd().
529         // While such an event stream is deleted (with this destructor) fieldObj would try
530         // to make the field object consisent, after the writer was deleted. To prevent that,
531         // we detach fieldObj from the writer so fieldObj does nothing on destruction.
532         fieldObj.detach();
533
534         delete internal_req;
535     }
536
537 void selfRelease()
538 {
539     // Disable the event stream if it is still enabled.
540     if (isEnabled())
541     {
542         disable();
543     }
544
545     // The request to destroy this event stream is nested and processed in the delegate to
546     // wait for the response, so it is not multi-threaded and no need to protect the variables.
547     if (!isInEventCallback)
548     {
549         delete this;
550         return;
551     }
552
553     // We are currently in an event callback.
554     // Cancel the wait and release this object later when the callback returns.
555     req->cancel();
556     selfReleaseAfterReturnFromEventCallback = true;
557 }
558
559 virtual IrisErrorCode enable() = 0;
560
561 virtual IrisErrorCode disable() = 0;
562
563 virtual IrisErrorCode getState(IrisValueMap& fields)
564 {
565     (void) fields;
566     return E_not_supported_for_event_source;
567 }
568
569 virtual IrisErrorCode flush(RequestId requestId)
570 {
571     (void) requestId;
572     return E_not_supported_for_event_source;
573 }
574
575 virtual IrisErrorCode setOptions(const AttributeValueMap& options, bool eventStreamCreate,
576 std::string& errorMessageOut)
577 {
578     (void) options;
579     (void) eventStreamCreate;
580     (void) errorMessageOut;
581
582     // Event streams which do not support options happily accept an empty options map.
583     return options.empty() ? E_ok : E_not_supported_for_event_source;
584 }
585
586 virtual IrisErrorCode action(const BreakpointAction& action_)
587 {
588     (void) action_;
589     return E_not_supported_for_event_source;
590 }
591
592 // Temporary: Keep PVModelLib happy. TODO: Remove.
593 virtual IrisErrorCode insertTrigger()
594 {
595     return E_not_supported_for_event_source;
596 }

```



```

674     }
675
676
677     // --- Functions for basic properties ---
678
679     void setProperties(IrisInstance* irisInstance, IrisInstanceEvent* irisInstanceEvent, EventSourceId
680     evSrcId,
681                     InstanceId ecInstId, const std::string& ecFunc, EventStreamId esId,
682                     bool syncEc);
683
684     bool isEnabled() const
685     {
686         return enabled;
687     }
688
689     EventStreamId getEsId() const
690     {
691         return esId;
692     }
693
694     const EventSourceInfo* getEventSourceInfo() const
695     {
696         return irisInstanceEvent ? irisInstanceEvent->getEventSourceInfo(evSrcId) : nullptr;
697     }
698
699     EventSourceId getEventSourceId() const { return evSrcId; }
700
701     InstanceId getEcInstId() const
702     {
703         return ecInstId;
704     }
705
706     // --- Functions for the counter mode ---
707
708     void setCounter(uint64_t startVal, const EventCounterMode& counterMode);
709
710     bool isCounter() const
711     {
712         return counter;
713     }
714
715     void setProxyForOtherInstance()
716     {
717         isProxyForOtherInstance = true;
718     }
719
720     bool IsProxyForOtherInstance() const
721     {
722         return isProxyForOtherInstance;
723     }
724
725     void setProxiedByInstanceId(InstanceId instId)
726     {
727         proxiedByInstanceId = instId;
728     }
729
730     bool IsProxiedByOtherInstance() const
731     {
732         return proxiedByInstanceId != IRIS_UINT64_MAX;
733     }
734
735     InstanceId getProxiedByInstanceId() const
736     {
737         return proxiedByInstanceId;
738     }
739
740     uint64_t getCountVal() const
741     {
742         return curVal;
743     }
744
745     // --- Functions for event stream with ranges
746
747     IrisErrorCode setRanges(const std::string& aspect, const std::vector<uint64_t>& ranges);
748
749     bool checkRangePc(uint64_t pc) const
750     {
751         return ranges.empty() || (aspect != ":pc") || checkRangesHelper(pc, ranges);
752     }
753
754     // --- Functions to emit the event callback ---
755     // Usage (example):
756     //     emitEventBegin(time, pc);    // Start to emit the callback.
757     //     addField(...);             // Add field value.
758     //     addField(...);             // Add field value.
759     //     ...
760     //     emitEventEnd();             // Emit the callback.

```

```

858
866 void emitEventBegin(IrisRequest& req, uint64_t time, uint64_t pc = IRIS_UINT64_MAX);
867
874 void emitEventBegin(uint64_t time, uint64_t pc = IRIS_UINT64_MAX);
875
885 void addField(const IrisU64StringConstant& field, uint64_t value)
886 {
887     addFieldRangeHelper(field, value);
888 }
889
899 void addField(const IrisU64StringConstant& field, int64_t value)
900 {
901     addFieldRangeHelper(field, value);
902 }
903
913 void addField(const IrisU64StringConstant& field, bool value)
914 {
915     addFieldRangeHelper(field, value);
916 }
917
927 template <class T>
928 void addField(const IrisU64StringConstant& field, const T& value)
929 {
930     fieldObj.member(field, value);
931 }
932
942 void addField(const IrisU64StringConstant& field, const uint8_t *data, size_t sizeInBytes)
943 {
944     fieldObj.member(field, data, sizeInBytes);
945 }
946
956 void addFieldSlow(const std::string& field, uint64_t value)
957 {
958     addFieldSlowRangeHelper(field, value);
959 }
960
970 void addFieldSlow(const std::string& field, int64_t value)
971 {
972     addFieldSlowRangeHelper(field, value);
973 }
974
984 void addFieldSlow(const std::string& field, bool value)
985 {
986     addFieldSlowRangeHelper(field, value);
987 }
988
998 template <class T>
999 void addFieldSlow(const std::string& field, const T& value)
1000 {
1001     fieldObj.memberSlow(field, value);
1002 }
1003
1013 void addFieldSlow(const std::string& field, const uint8_t *data, size_t sizeInBytes)
1014 {
1015     fieldObj.memberSlow(field, data, sizeInBytes);
1016 }
1017
1027 void emitEventEnd(bool send = true);
1028
1029 private:
1031
1035 bool counterTrigger();
1036
1038 bool checkRanges() const
1039 {
1040     return !aspectFound || checkRangesHelper(curAspectValue, ranges);
1041 }
1042
1044 static bool checkRangesHelper(uint64_t value, const std::vector<uint64_t>& ranges);
1045
1047 template <typename T>
1048 void addFieldRangeHelper(const IrisU64StringConstant& field, T value)
1049 {
1050     if (!aspect.empty() && aspect == toString(field))
1051     {
1052         aspectFound = true;
1053         curAspectValue = static_cast<uint64_t>(value);
1054     }
1055     fieldObj.member(field, value);
1056 }
1057
1058
1060 template <typename T>
1061 void addFieldSlowRangeHelper(const std::string& field, T value)
1062 {
1063     if (aspect == field)
1064     {

```

```

1065         aspectFound    = true;
1066         curAspectValue = static_cast<uint64_t>(value);
1067     }
1068
1069     fieldObj.memberSlow(field, value);
1070 }
1071
1072 protected:
1073
1074     IrisInstance* irisInstance{};
1075
1076     IrisInstanceEvent* irisInstanceEvent{};
1077
1078     EventSourceId evSrcId{IRIS_UINT64_MAX};
1079
1080     InstanceId ecInstId{IRIS_UINT64_MAX};
1081
1082     std::string ecFunc;
1083
1084     EventStreamId esId{IRIS_UINT64_MAX};
1085
1086     bool syncEc{};
1087
1088     bool enabled{};
1089
1090     IrisRequest* req{};
1091     IrisRequest* internal_req{};
1092     IrisU64JsonWriter::Object fieldObj;
1093
1094     bool counter{};
1095
1096     uint64_t startVal{};
1097     uint64_t curVal{};
1098
1099     EventCounterMode counterMode{};
1100
1101     std::string aspect;
1102     std::vector<uint64_t> ranges;
1103
1104     bool aspectFound{};
1105
1106     uint64_t curAspectValue{};
1107
1108     bool isProxyForOtherInstance{false};
1109
1110     InstanceId proxiedByInstanceId{IRIS_UINT64_MAX};
1111
1112 private:
1113     int isInEventCallback{};
1114
1115     bool selfReleaseAfterReturnFromEventCallback{};
1116 };
1117
1118 class IrisEventStream : public EventStream
1119 {
1120 public:
1121     IrisEventStream(IrisEventRegistry* registry_);
1122
1123     virtual IrisErrorCode enable() IRIS_OVERRIDE;
1124
1125     virtual IrisErrorCode disable() IRIS_OVERRIDE;
1126
1127 private:
1128     IrisEventRegistry* registry;
1129 };
1130
1131 class IrisEventRegistry
1132 {
1133 public:
1134     bool empty() const
1135     {
1136         return esSet.empty();
1137     }
1138
1139     bool registerEventStream(EventStream* evStream);
1140
1141     bool unregisterEventStream(EventStream* evStream);
1142
1143     // --- Functions to emit the callback of all registered event streams ---
1144     // Usage (example):
1145     //     emitEventBegin(time, pc);    // Start to emit the callback.
1146     //     addField(...);              // Add field value.
1147     //     addField(...);              // Add field value.
1148     //     ...
1149     //     emitEventEnd();              // Emit the callback.

```

```

1197
1198     void emitEventBegin(uint64_t time, uint64_t pc = IRIS_UINT64_MAX) const;
1199
1200     template <class T>
1201     void addField(const IrisU64StringConstant& field, const T& value) const
1202     {
1203         for (std::set<EventStream*>::const_iterator i = esSet.begin(), e = esSet.end(); i != e; i++)
1204             (*i)->addField(field, value);
1205     }
1206
1207     template <class T>
1208     void addFieldSlow(const std::string& field, const T& value) const
1209     {
1210         for (std::set<EventStream*>::const_iterator i = esSet.begin(), e = esSet.end(); i != e; i++)
1211             (*i)->addFieldSlow(field, value);
1212     }
1213
1214     template <class T, typename F>
1215     void forEach(F && func) const
1216     {
1217         for (std::set<EventStream*>::const_iterator i = esSet.begin(), e = esSet.end(); i != e; i++)
1218         {
1219             T* t = static_cast<T*>(*i);
1220             func(*t);
1221         }
1222     }
1223
1224     void emitEventEnd() const;
1225
1226     typedef std::set<EventStream*>::const_iterator iterator;
1227
1228     iterator begin() const
1229     {
1230         return esSet.begin();
1231     }
1232
1233     iterator end() const
1234     {
1235         return esSet.end();
1236     }
1237
1238     ~IrisEventRegistry()
1239     {
1240         // Disable any remaining event streams.
1241         // Calling disable() on an EventStream will cause esSet to be modified so we need to loop
1242         without
1243         // using iterators which become invalidated.
1244         while (!esSet.empty())
1245         {
1246             (*esSet.begin())->disable();
1247         }
1248     }
1249
1250 private:
1251     // All registered event streams
1252     std::set<EventStream*> esSet;
1253 };
1254
1255 #endif // #ifndef ARM_INCLUDE_IrisInstanceBreakpoint_h

```

9.29 IrisInstanceFactoryBuilder.h File Reference

A helper class to build instantiation parameter metadata.

```

#include "iris/IrisParameterBuilder.h"
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisObjects.h"
#include <string>
#include <vector>

```

Classes

- class [iris::IrisInstanceFactoryBuilder](#)
A builder class to construct instantiation parameter metadata.

9.29.1 Detailed Description

A helper class to build instantiation parameter metadata.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.30 IrisInstanceFactoryBuilder.h

[Go to the documentation of this file.](#)

```

1
2
3
4
5
6
7 #ifndef ARM_INCLUDE_IrisInstanceFactoryBuilder_h
8 #define ARM_INCLUDE_IrisInstanceFactoryBuilder_h
9
10 #include "iris/IrisParameterBuilder.h"
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisObjects.h"
13
14 #include <string>
15 #include <vector>
16
17 namespace IRIS_START
18 {
19
20     class IrisInstanceFactoryBuilder
21     {
22     private:
23         std::vector<ResourceInfo> parameters;
24         std::vector<ResourceInfo> hidden_parameters;
25         std::string parameter_prefix;
26
27         ResourceInfo& addParameterInternal(const std::string& name, uint64_t bitWidth, const std::string&
28             description,
29             const std::string& type, bool hidden)
30         {
31             std::vector<ResourceInfo>& param_list = hidden ? hidden_parameters : parameters;
32             param_list.resize(parameters.size() + 1);
33             ResourceInfo& info = param_list.back();
34
35             info.name = name;
36             info.bitWidth = bitWidth;
37             info.description = description;
38             info.type = type;
39
40             return info;
41         }
42     public:
43         IrisInstanceFactoryBuilder(const std::string& prefix)
44             : parameter_prefix(prefix)
45         {
46
47         }
48
49         IrisParameterBuilder addParameter(const std::string& name, uint64_t bitWidth, const std::string&
50             description)
51         {
52             return IrisParameterBuilder(addParameterInternal(parameter_prefix + name, bitWidth, description,
53                 "" /*numeric*/, false));
54         }
55
56         IrisParameterBuilder addHiddenParameter(const std::string& name, uint64_t bitWidth, const
57             std::string& description)
58         {
59             return IrisParameterBuilder(addParameterInternal(parameter_prefix + name, bitWidth, description,
60                 "" /*numeric*/, true));
61         }
62
63         IrisParameterBuilder addStringParameter(const std::string& name, const std::string& description)
64         {
65             return IrisParameterBuilder(addParameterInternal(parameter_prefix + name, 0, description,
66                 "string", false));
67         }
68
69         IrisParameterBuilder addHiddenStringParameter(const std::string& name, const std::string&
70             description)
71         {
72             return IrisParameterBuilder(addParameterInternal(parameter_prefix + name, 0, description,
73                 "string", true));
74         }
75
76         IrisParameterBuilder addBoolParameter(const std::string& name, const std::string& description)

```

```

121     {
122         ResourceInfo& info = addParameterInternal(parameter_prefix + name, 1, description, "numeric",
false);
123
124         // Be explicit about the range even though there are only two possible values anyway.
125         info.parameterInfo.min.push_back(0);
126         info.parameterInfo.max.push_back(1);
127
128         // Add enum strings for the values
129         info.enums.push_back(EnumElementInfo(IrisValue(0), "false", ""));
130         info.enums.push_back(EnumElementInfo(IrisValue(1), "true", ""));
131
132         return IrisParameterBuilder(info);
133     }
134     IRIS_DEPRECATED("use addBoolParameter() instead") IrisParameterBuilder addBooleanParameter(const
std::string& name, const std::string& description)
135     {
136         return addBoolParameter(name, description);
137     }
138
139     IrisParameterBuilder addHiddenBoolParameter(const std::string& name, const std::string& description)
140     {
141         ResourceInfo& info = addParameterInternal(parameter_prefix + name, 1, description, "numeric",
true);
142
143         // Be explicit about the range even though there are only two possible values anyway.
144         info.parameterInfo.min.push_back(0);
145         info.parameterInfo.max.push_back(1);
146
147         // Add enum strings for the values
148         info.enums.push_back(EnumElementInfo(IrisValue(0), "false", ""));
149         info.enums.push_back(EnumElementInfo(IrisValue(1), "true", ""));
150
151         return IrisParameterBuilder(info);
152     }
153     IRIS_DEPRECATED("use addHiddenBoolParameter() instead") IrisParameterBuilder
addHiddenBooleanParameter(const std::string& name, const std::string& description)
154     {
155         return addHiddenBoolParameter(name, description);
156     }
157
158     const std::vector<ResourceInfo>& getParameterInfo() const
159     {
160         return parameters;
161     }
162
163     const std::vector<ResourceInfo>& getHiddenParameterInfo() const
164     {
165         return hidden_parameters;
166     }
167 };
168
169 namespace IRIS
170 {
171     #endif // ARM_INCLUDE_IrisInstanceFactoryBuilder_h

```

9.31 IrisInstanceImage.h File Reference

Image-loading add-on to IrisInstance and image-loading callback add-on to the caller.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include <cstdio>

```

Classes

- class [iris::IrisInstanceImage](#)
Image loading add-on for [IrisInstance](#).
- class [iris::IrisInstanceImage_Callback](#)
Image loading add-on for [IrisInstance](#) clients implementing `image_loadDataRead()`.

Typedefs

- typedef IrisDelegate< const std::vector< uint8_t > & > [iris::ImageLoadDataDelegate](#)
Delegate to load an image from the given data.
- typedef IrisDelegate< const std::string & > [iris::ImageLoadFileDelegate](#)
Delegate function to load an image from the given file.

9.31.1 Detailed Description

Image-loading add-on to IrisInstance and image-loading callback add-on to the caller.

Copyright

Copyright (C) 2016-2022 Arm Limited. All rights reserved.

The IrisInstanceImage class:

- Implements all image-loading Iris functions.
- Maintains and provides image metadata, for example path, instanceSideFile, rawAddr.
- Converts between Iris image-loading functions (image_load*()) and various C++ access functions.

9.31.2 Typedef Documentation

9.31.2.1 ImageLoadDataDelegate

```
typedef IrisDelegate<const std::vector<uint8_t>&> iris::ImageLoadDataDelegate
```

Delegate to load an image from the given data.

```
IrisErrorCode loadImage(const std::vector<uint8_t> &data)
```

Typical implementations try to load the data with the supported formats.

Errors:

- If the image format is unknown, E_unknown_image_format is returned.
- If the image format is known but the image could not be loaded, E_image_format_error is returned.

9.31.2.2 ImageLoadFileDelegate

```
typedef IrisDelegate<const std::string&> iris::ImageLoadFileDelegate
```

Delegate function to load an image from the given file.

The path can be absolute or relative to the current working directory.

```
IrisErrorCode loadImage(const std::string &path)
```

Typical implementations try to load the file with the supported formats.

Errors:

- If the file specified by path could not be opened, E_error_opening_file is returned.
- If the file could be opened but could not be read, E_io_error is returned.
- If the image format is unknown, E_unknown_image_format is returned.
- If the image format is known but the image could not be loaded, E_image_format_error is returned.

9.32 IrisInstanceImage.h

[Go to the documentation of this file.](#)

```

1
13 #ifndef ARM_INCLUDE_IrisInstanceImage_h
14 #define ARM_INCLUDE_IrisInstanceImage_h
15
16 #include "iris/detail/IrisCommon.h"
17 #include "iris/detail/IrisDelegate.h"
18 #include "iris/detail/IrisLogger.h"
19 #include "iris/detail/IrisObjects.h"
20
21 #include <cstdio>
22
23 NAMESPACE_IRIS_START
24
25 class IrisInstance;
26 class IrisReceivedRequest;
27
28 typedef IrisDelegate<const std::string&> ImageLoadFileDelegate;
29
30 typedef IrisDelegate<const std::vector<uint8_t>&> ImageLoadDataDelegate;
31
32 class IrisInstanceImage
33 {
34 public:
35     IrisInstanceImage(IrisInstance* irisInstance = 0);
36
37     void attachTo(IrisInstance* irisInstance);
38
39     void setLoadImageFileDelegate(ImageLoadFileDelegate delegate);
40
41     void setLoadImageDataDelegate(ImageLoadDataDelegate delegate);
42
43     static IrisErrorCode readFileData(const std::string& fileName, std::vector<uint8_t>& data);
44
45 private:
46     void loadImageFromData(IrisReceivedRequest& request, const ImageReadResult& imageData);
47
48     void impl_image_loadFile(IrisReceivedRequest& request);
49
50     void impl_image_loadData(IrisReceivedRequest& request);
51
52     void impl_image_loadDataPull(IrisReceivedRequest& request);
53
54     void impl_image_getMetaInfoList(IrisReceivedRequest& request);
55
56     void impl_image_clearMetaInfoList(IrisReceivedRequest& request);
57
58     void writeRawDataToMemory(IrisReceivedRequest& request, const std::vector<uint8_t>& data, uint64_t
rawAddr, MemorySpaceId rawSpaceId);
59
60     IrisErrorCode pullData(InstanceId callerId, uint64_t tag, ImageReadResult& result);
61
62     IrisInstance* irisInstance;
63
64     typedef std::vector<ImageMetaInfo> ImageMetaInfoList;
65     ImageMetaInfoList metaInfos;
66
67     IrisLogger log;
68
69     ImageLoadFileDelegate loadFileDelegate;
70     ImageLoadDataDelegate loadDataDelegate;
71 };
72
73 class IrisInstanceImage_Callback
74 {
75 public:
76     IrisInstanceImage_Callback(IrisInstance* irisInstance = 0);
77
78     ~IrisInstanceImage_Callback();
79
80     void attachTo(IrisInstance* irisInstance);
81
82     uint64_t openImage(const std::string& fileName);
83
84 protected:
85     void impl_image_loadDataRead(IrisReceivedRequest& request);
86
87 private:
88     IrisErrorCode readImageData(uint64_t tag, uint64_t position, uint64_t size, bool end,
ImageReadResult& result);
89
90
91

```



```

210     IrisInstance* irisInstance;
211
212     IrisLogger log;
213
214     typedef std::vector<FILE*> ImageList;
215     ImageList images;
216 };
217
218 NAMESPACE_IRIS_END
219
220 #endif // #ifndef ARM_INCLUDE_IrisInstanceImage_h

```

9.33 IrisInstanceMemory.h File Reference

Memory add-on to IrisInstance.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"

```

Classes

- struct [iris::IrisInstanceMemory::AddressTranslationInfoAndAccess](#)
Contains static address translation information.
- class [iris::IrisInstanceMemory](#)
Memory add-on for [IrisInstance](#).
- struct [iris::IrisInstanceMemory::SpaceInfoAndAccess](#)
Entry in 'spaceInfos'.

Typedefs

- typedef [IrisDelegate< uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult & >](#) [iris::MemoryAddressTranslateDelegate](#)
Delegate to translate an address.
- typedef [IrisDelegate< const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap & >](#) [iris::MemoryGetSidebandInfoDelegate](#)
- typedef [IrisDelegate< const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult & >](#) [iris::MemoryReadDelegate](#)
Delegate to read memory data.
- typedef [IrisDelegate< const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult & >](#) [iris::MemoryWriteDelegate](#)
Delegate to write memory data.

9.33.1 Detailed Description

Memory add-on to IrisInstance.

Copyright

Copyright (C) 2015 Arm Limited. All rights reserved.

The IrisInstanceMemory class:

- Implements all memory-related Iris functions.
- Feeds memory-related properties (memory.*) to `instance_getProperties()` of the associated `IrisInstance`.
- Provides infrastructure that is useful for Iris clients.
- Maintains and provides memory meta information (memory spaces, address translations, sideband information).
- Converts between Iris memory access functions (`memory_read()`) and various C++ access functions.

9.33.2 Typedef Documentation

9.33.2.1 MemoryAddressTranslateDelegate

typedef IrisDelegate<uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult&> [iris::MemoryAddressTranslateDelegate](#)

Delegate to translate an address.

```
IrisErrorCode translate(MemorySpaceId inSpaceId, uint64_t address,
                      MemorySpaceId outSpaceId, MemoryAddressTranslationResult &result)
```

inSpaceId, address, and outSpaceId are guaranteed to be valid.

Typical implementations inspect the inSpaceId and outSpaceId to determine how to translate the address.

Return addresses are appended to result.address, which is a vector<uint64_t>:

- If this array is empty then 'address' is not mapped in 'outSpaceId'.
- If the array contains exactly one element then the mapping is unique.
- If it contains multiple addresses then 'address' is accessible in the same way under all of these addresses in 'outSpaceId'.

Error: Return E_* error code for translation errors.

9.33.2.2 MemoryGetSidebandInfoDelegate

typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, const IrisValueMap&, const std::vector<std::string>&, IrisValueMap&> [iris::MemoryGetSidebandInfoDelegate](#)

@ Delegate to get memory sideband information.

```
IrisErrorCode getSidebandInfo(const MemorySpaceInfo &spaceInfo, uint64_t address,
                             const IrisValueMap &attrib,
                             const std::vector<std::string> &request,
                             IrisValueMap &result)
```

Returns sideband information for a range of addresses in a given memory space.

9.33.2.3 MemoryReadDelegate

typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t, const AttributeValueMap&, MemoryReadResult&> [iris::MemoryReadDelegate](#)

Delegate to read memory data.

```
IrisErrorCode read(const MemorySpaceInfo &spaceInfo, uint64_t address, uint64_t byteWidth,
                  uint64_t count, const AttributeValueMap &attrib, MemoryReadResult &result)
```

spaceInfo, address, byteWidth, and count are guaranteed to be valid.

Typical implementations inspect the spaceId, address, byteWidth, and count to determine which memory elements should be read. Then they append the read elements to result.data, which is a vector<uint64_t>:

- Data elements are read from ascending addresses, packed into uint64_ts such that the lowest address is in the lowest bits.
- Elements of byteWidth >= 2 are read with the endianness of the memory space inside each element, but elements are stored with the lowest bits inside each uint64_t (for byteWidth < 8) and with the lowest bits first in sequences of uint64_t (for byteWidth > 8).

Error: Return E_* error code for read errors. It appends the address that could not be read to result.error.

9.33.2.4 MemoryWriteDelegate

typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t, const AttributeValueMap&, const uint64_t*, MemoryWriteResult&> [iris::MemoryWriteDelegate](#)

Delegate to write memory data.

```
IrisErrorCode write(const MemorySpaceInfo &spaceInfo, uint64_t address, uint64_t byteWidth,
                  uint64_t count, const AttributeValueMap &attrib, const uint64_t *data, MemoryWriteResult
                  &result)
```

See also

MemoryReadDelegate data contains the data elements to be written in the same format as MemoryReadResult.data for reads.

9.34 IrisInstanceMemory.h

[Go to the documentation of this file.](#)

```

1
14 #ifndef ARM_INCLUDE_IrisInstanceMemory_h
15 #define ARM_INCLUDE_IrisInstanceMemory_h
16
17 #include "iris/detail/IrisCommon.h"
18 #include "iris/detail/IrisDelegate.h"
19 #include "iris/detail/IrisLogger.h"
20 #include "iris/detail/IrisObjects.h"
21
22 namespace IRIS_START
23 {
24     class IrisInstance;
25     class IrisReceivedRequest;
26
27     typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
28         const AttributeValueMap&, MemoryReadResult&>
29         MemoryReadDelegate;
30
31     typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
32         const AttributeValueMap&, const uint64_t*, MemoryWriteResult&>
33         MemoryWriteDelegate;
34
35     typedef IrisDelegate<uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult&>
36         MemoryAddressTranslateDelegate;
37
38     typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, const IrisValueMap&,
39         const std::vector<std::string>&, IrisValueMap&>
40         MemoryGetSidebandInfoDelegate;
41
42     class IrisInstanceMemory
43     {
44     public:
45         struct SpaceInfoAndAccess
46         {
47             MemorySpaceInfo          spaceInfo;
48             MemoryReadDelegate        readDelegate;    // May be empty. In this case
49             defaultReadDelegate is used.
50             MemoryWriteDelegate       writeDelegate;   // May be empty. In this case
51             defaultWriteDelegate is used.
52             MemoryGetSidebandInfoDelegate sidebandDelegate; // May be empty. In this case sidebandDelegate
53             is used.
54         };
55
56         struct AddressTranslationInfoAndAccess
57         {
58             AddressTranslationInfoAndAccess (MemorySpaceId inSpaceId, MemorySpaceId outSpaceId, const
59             std::string& description)
60             : translationInfo(inSpaceId, outSpaceId, description)
61             {
62             }
63
64             MemorySupportedAddressTranslationResult translationInfo;
65             MemoryAddressTranslateDelegate          translateDelegate;
66         };
67
68         IrisInstanceMemory(IrisInstance* irisInstance = 0);
69
70         void attachTo(IrisInstance* irisInstance);
71
72         void setDefaultReadDelegate(MemoryReadDelegate delegate = MemoryReadDelegate())
73         {
74             memReadDelegate = delegate;
75         }
76
77         void setDefaultWriteDelegate(MemoryWriteDelegate delegate = MemoryWriteDelegate())
78         {
79             memWriteDelegate = delegate;
80         }
81
82         SpaceInfoAndAccess& addMemorySpace(const std::string& name);
83
84         AddressTranslationInfoAndAccess& addAddressTranslation(MemorySpaceId inSpaceId, MemorySpaceId
85         outSpaceId,
86
87             const std::string& description);
88
89         void setDefaultTranslateDelegate(MemoryAddressTranslateDelegate delegate =
90         MemoryAddressTranslateDelegate())
91         {
92             translateDelegate = delegate;
93         }
94
95         void setDefaultGetSidebandInfoDelegate(MemoryGetSidebandInfoDelegate delegate =
96         MemoryGetSidebandInfoDelegate())

```

```

232     {
233         if (delegate.empty())
234         {
235             delegate = MemoryGetSidebandInfoDelegate::make<IrisInstanceMemory,
236             &IrisInstanceMemory::getDefaultSidebandInfo>(this);
237         }
238         sidebandDelegate = delegate;
239     }
240
241 private:
242
243     void impl_memory_getMemorySpaces(IrisReceivedRequest& request);
244     void impl_memory_read(IrisReceivedRequest& request);
245     void impl_memory_write(IrisReceivedRequest& request);
246     void impl_memory_translateAddress(IrisReceivedRequest& request);
247     void impl_memory_getUsefulAddressTranslations(IrisReceivedRequest& request);
248     void impl_memory_getSidebandInfo(IrisReceivedRequest& request);
249
250     IrisErrorCode getDefaultSidebandInfo(const MemorySpaceInfo& spaceInfo, uint64_t address,
251     const IrisValueMap& attrib,
252     const std::vector<std::string>& request,
253     IrisValueMap& result);
254
255     // --- state ---
256
257     IrisInstance* irisInstance;
258
259     typedef std::vector<SpaceInfoAndAccess> SpaceInfoList;
260     SpaceInfoList spaceInfos;
261
262     typedef std::vector<AddressTranslationInfoAndAccess> SupportedTranslations;
263     SupportedTranslations supportedTranslations;
264
265     MemoryReadDelegate memReadDelegate;
266     MemoryWriteDelegate memWriteDelegate;
267     MemoryAddressTranslateDelegate translateDelegate;
268
269     MemoryGetSidebandInfoDelegate sidebandDelegate;
270
271     IrisLogger log;
272 };
273
274 namespace IRIS_END
275
276 #endif // #ifndef ARM_INCLUDE_IrisInstanceMemory_h

```

9.35 IrisInstancePerInstanceExecution.h File Reference

Per-instance execution control add-on to IrisInstance.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include <cstdio>

```

Classes

- class [iris::IrisInstancePerInstanceExecution](#)
Per-instance execution control add-on for [IrisInstance](#).

Typedefs

- typedef IrisDelegate< bool & > [iris::PerInstanceExecutionStateGetDelegate](#)
Get the execution state.
- typedef IrisDelegate< bool > [iris::PerInstanceExecutionStateSetDelegate](#)
Delegate to set the execution state.

9.35.1 Detailed Description

Per-instance execution control add-on to IrisInstance.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

Implements all per-instance execution control-related Iris functions.

9.35.2 Typedef Documentation

9.35.2.1 PerInstanceExecutionStateGetDelegate

```
typedef IrisDelegate<bool&> iris::PerInstanceExecutionStateGetDelegate
```

Get the execution state.

enabled should be set to true if execution is enabled and false otherwise.

```
IrisErrorCode getState(bool &enabled)
```

Return E_ok on success, otherwise return the error code.

9.35.2.2 PerInstanceExecutionStateSetDelegate

```
typedef IrisDelegate<bool> iris::PerInstanceExecutionStateSetDelegate
```

Delegate to set the execution state.

Enable or disable the execution of instructions (or processing of work items).

```
IrisErrorCode setState(bool enable)
```

Return E_ok on success, otherwise return the error code.

9.36 IrisInstancePerInstanceExecution.h

[Go to the documentation of this file.](#)

```
1
9 #ifndef ARM_INCLUDE_IrisInstancePerInstanceExecution_h
10 #define ARM_INCLUDE_IrisInstancePerInstanceExecution_h
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisDelegate.h"
14 #include "iris/detail/IrisLogger.h"
15 #include "iris/detail/IrisObjects.h"
16
17 #include <cstdio>
18
19 NAMESPACE_IRIS_START
20
21 class IrisInstance;
22 class IrisReceivedRequest;
23
24 typedef IrisDelegate<bool> PerInstanceExecutionStateSetDelegate;
25
26 typedef IrisDelegate<bool&> PerInstanceExecutionStateGetDelegate;
27
28 class IrisInstancePerInstanceExecution
29 {
30 public:
31     IrisInstancePerInstanceExecution(IrisInstance* irisInstance = nullptr);
32
33     void attachTo(IrisInstance* irisInstance);
34
35     void setExecutionStateSetDelegate(PerInstanceExecutionStateSetDelegate delegate);
36
37     void setExecutionStateGetDelegate(PerInstanceExecutionStateGetDelegate delegate);
38
39 private:
40     void impl_perInstanceExecution_setState(IrisReceivedRequest& request);
41
42     void impl_perInstanceExecution_getState(IrisReceivedRequest& request);
43
44     IrisInstance* irisInstance;
45
46     PerInstanceExecutionStateSetDelegate execStateSet;
47     PerInstanceExecutionStateGetDelegate execStateGet;
48 }
```

```

101
103     IrisLogger log;
104 };
105
106 NAMESPACE_IRIS_END
107
108 #endif // #ifndef ARM_INCLUDE_IrisInstancePerInstanceExecution_h

```

9.37 IrisInstanceResource.h File Reference

Resource add-on to IrisInstance.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include <cassert>

```

Classes

- class [iris::IrisInstanceResource](#)
Resource add-on for [IrisInstance](#).
- struct [iris::IrisInstanceResource::ResourceInfoAndAccess](#)
Entry in 'resourceInfos'.
- struct [iris::ResourceWriteValue](#)

Typedefs

- typedef [IrisDelegate](#)< const [ResourceInfo](#) &, [ResourceReadResult](#) & > [iris::ResourceReadDelegate](#)
Delegate to read resources.
- typedef [IrisDelegate](#)< const [ResourceInfo](#) &, const [ResourceWriteValue](#) & > [iris::ResourceWriteDelegate](#)
Delegate to write resources.

Functions

- uint64_t [iris::resourceReadBitField](#) (uint64_t parentValue, const [ResourceInfo](#) &resourceInfo)
- template<class T >
void [iris::resourceWriteBitField](#) (T &parentValue, uint64_t fieldValue, const [ResourceInfo](#) &resourceInfo)

9.37.1 Detailed Description

Resource add-on to IrisInstance.

Copyright

Copyright (C) 2015-2019 Arm Limited. All rights reserved.

The [IrisInstanceResource](#) class:

- Implements all resource-related Iris functions.
- Feeds resource-related properties (resource.*) to [instance_getProperties\(\)](#) of the associated [IrisInstance](#).
- Provides infrastructure that is useful for Iris clients.
- Maintains and provides resource meta information (name, bitwidth).
- Converts between Iris resource-access functions ([resource_read\(\)](#)) and various C++ access functions.

9.37.2 Typedef Documentation

9.37.2.1 ResourceReadDelegate

typedef IrisDelegate<const ResourceInfo&, ResourceReadResult&> iris::ResourceReadDelegate
 Delegate to read resources.

IrisErrorCode read(const ResourceInfo &resourceInfo, ResourceReadResult &result)

resourceInfo.rsclId is guaranteed to be valid.

Typical implementations inspect the rsclId, canonicalRn, addressOffset, or even the name or cname value to determine which resource should be read and then append the read data to result:

- Return data (no undefined bits):
 - Append data to result.data, which is a vector<uint64_t>. Append one uint64_t if resource is <= 64 bits.
 - Append multiple uint64_t for wider resources, least significant uint64_t first.
- Return data with undefined bits:
 - Same as above, but in addition, append a mask which contains 1 bit for all undefined bits to result.↵ undefinedBits (same format and length as result.data) and set all undefined bits to 0 in result.data.

Error: If the resource could not be read, return E_* error code, for example E_error_reading_write_only_resource, E_error_reading_resource, or E_not_implemented, and leave result unchanged.

9.37.2.2 ResourceWriteDelegate

typedef IrisDelegate<const ResourceInfo&, const ResourceWriteValue&> iris::ResourceWriteDelegate
 Delegate to write resources.

IrisErrorCode write(const ResourceInfo &resourceInfo, const ResourceWriteValue &value)

resourceInfo.rsclId is guaranteed to be valid.

Typical implementations inspect the rsclId, canonicalRn, addressOffset, or even the name or cname value to determine which resource should be written.

data contains the data for all resources to be written in the same format as ResourceReadResult.data for reads. The number of elements in the data array is resourceInfo.getDataSizeInU64Chunks(). data is only evaluated for string resources.

9.37.3 Function Documentation

9.37.3.1 resourceReadBitField()

```
uint64_t iris::resourceReadBitField (
    uint64_t parentValue,
    const ResourceInfo & resourceInfo ) [inline]
```

Helper for ResourceReadDelegates to read a bit field of a parent register according to the lsbOffset and bitWidth in resourceInfo. This helps reducing redundancy in the debug interface implementation.

9.37.3.2 resourceWriteBitField()

```
template<class T >
void iris::resourceWriteBitField (
    T & parentValue,
    uint64_t fieldValue,
    const ResourceInfo & resourceInfo ) [inline]
```

Helper for ResourceWriteDelegates to write a bit field of a parent register according to the lsbOffset and bitWidth in resourceInfo. This helps reducing redundancy in the debug interface implementation.

9.38 IrisInstanceResource.h

[Go to the documentation of this file.](#)

```

1
14 #ifndef ARM_INCLUDE_IrisInstanceResource_h
15 #define ARM_INCLUDE_IrisInstanceResource_h
16
17 #include "iris/detail/IrisCommon.h"
18 #include "iris/detail/IrisDelegate.h"
19 #include "iris/detail/IrisLogger.h"
20 #include "iris/detail/IrisObjects.h"
21
22 #include <cassert>
23
24 NAMESPACE_IRIS_START
25
26 class IrisInstance;
27 class IrisReceivedRequest;
28
29 inline uint64_t resourceReadBitField(uint64_t parentValue, const ResourceInfo& resourceInfo)
30 {
31     return (resourceInfo.registerInfo.lsbOffset < 64) ?
32         ((parentValue » resourceInfo.registerInfo.lsbOffset) & maskWidthLsb(resourceInfo.bitWidth, 0))
33         : 0;
34 }
35
36 template<class T>
37 inline void resourceWriteBitField(T& parentValue, uint64_t fieldValue, const ResourceInfo& resourceInfo)
38 {
39     T mask = T(maskWidthLsb(resourceInfo.bitWidth, resourceInfo.registerInfo.lsbOffset));
40     parentValue &= ~mask;
41     parentValue |= (resourceInfo.registerInfo.lsbOffset < 64) ?
42         ((fieldValue « resourceInfo.registerInfo.lsbOffset) & mask)
43         : 0;
44 }
45
46 struct ResourceWriteValue
47 {
48     const uint64_t* data{};
49     const std::string* str{};
50 };
51
52 typedef IrisDelegate<const ResourceInfo&, ResourceReadResult&> ResourceReadDelegate;
53
54 typedef IrisDelegate<const ResourceInfo&, const ResourceWriteValue&> ResourceWriteDelegate;
55
56 class IrisInstanceResource
57 {
58 public:
59     struct ResourceInfoAndAccess
60     {
61         ResourceInfo resourceInfo;
62         ResourceReadDelegate readDelegate; // May be invalid. In this case defaultReadDelegate is
63         used.
64         ResourceWriteDelegate writeDelegate; // May be invalid. In this case defaultWriteDelegate is
65         used.
66     };
67
68     IrisInstanceResource(IrisInstance* irisInstance = 0);
69
70     void attachTo(IrisInstance* irisInstance);
71
72     ResourceInfoAndAccess& addResource(const std::string& type,
73                                       const std::string& name,
74                                       const std::string& description);
75
76     void beginResourceGroup(const std::string& name,
77                            const std::string& description,
78                            uint64_t startSubRscId = IRIS_UINT64_MAX,
79                            const std::string& cname = std::string());
80
81     void setNextSubRscId(ResourceId nextSubRscId_)
82     {
83         nextSubRscId = nextSubRscId_;
84     }
85
86     void setTag(ResourceId rscId, const std::string& tag);
87
88     ResourceInfoAndAccess* getResourceInfo(ResourceId rscId);
89
90     static void calcHierarchicalNames(std::vector<ResourceInfo>& resourceInfos);
91
92     static void makeNamesHierarchical(std::vector<ResourceInfo>& resourceInfos);
93

```



```

255
256 protected:
257     // --- Iris function implementations ---
258
259     void impl_resource_getList(IrisReceivedRequest& request);
260
261     void impl_resource_getListOfResourceGroups(IrisReceivedRequest& request);
262
263     void impl_resource_getResourceInfo(IrisReceivedRequest& request);
264
265     void impl_resource_read(IrisReceivedRequest& request);
266
267     void impl_resource_write(IrisReceivedRequest& request);
268
269 private:
270
271     static void calcHierarchicalNamesInternal(std::vector<ResourceInfo>& resourceInfos, const
std::map<ResourceId, size_t>& rscIdToIndex, std::vector<bool>& done, size_t index);
272
273     // --- State ---
274
275     IrisInstance* irisInstance;
276
277     IrisLogger log;
278
279     typedef std::vector<ResourceInfoAndAccess> ResourceInfoList;
280     ResourceInfoList resourceInfos;
281
282     typedef std::vector<ResourceGroupInfo> GroupInfoList;
283     GroupInfoList groupInfos;
284
285     typedef std::map<std::string, size_t> GroupNameToIndex;
286     GroupNameToIndex groupNameToIndex;
287
288     ResourceGroupInfo* currentAddGroup;
289
290     uint64_t nextSubRscId{IRIS_UINT64_MAX};
291 };
292
293 namespace IRIS_END
294
295 #endif // #ifndef ARM_INCLUDE_IrisInstanceResource_source

```

9.39 IrisInstanceSemihosting.h File Reference

IrisInstance add-on to implement semihosting functionality.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include "iris/IrisInstanceEvent.h"
#include <mutex>
#include <queue>

```

Classes

- class [iris::IrisInstanceSemihosting](#)

9.39.1 Detailed Description

IrisInstance add-on to implement semihosting functionality.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.40 IrisInstanceSemihosting.h

[Go to the documentation of this file.](#)

```

1
8 #ifndef ARM_INCLUDE_IrisInstanceSemihosting_h
9 #define ARM_INCLUDE_IrisInstanceSemihosting_h
10

```

```

11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisLogger.h"
13 #include "iris/detail/IrisObjects.h"
14
15 #include "iris/IrisInstanceEvent.h"
16
17 #include <mutex>
18 #include <queue>
19
20 NAMESPACE_IRIS_START
21
22 class IrisInstance;
23 class IrisInstanceEvent;
24 class IrisReceivedRequest;
25
26 namespace semihost
27 {
28
29     static const uint64_t COOKED = (0 << 0);
30
31     static const uint64_t RAW = (1 << 0);
32
33     static const uint64_t BLOCK = (0 << 1);
34
35     static const uint64_t NONBLOCK = (1 << 1);
36
37     static const uint64_t EMIT_EVENT = (0 << 2);
38
39     static const uint64_t NO_EVENT = (1 << 2);
40
41     static const uint64_t DEFAULT = COOKED | BLOCK | EMIT_EVENT;
42
43     static const uint64_t STDIN = 0;
44
45     static const uint64_t STDOUT = 1;
46
47     static const uint64_t STDERR = 2;
48
49 } // namespace semihost
50
51 class IrisInstanceSemihosting
52 {
53 private:
54     IrisInstance* iris_instance{nullptr};
55
56     IrisInstanceEvent* inst_event{nullptr};
57
58     std::map<uint64_t, unsigned> evSrcId_map{};
59
60     std::vector<IrisEventRegistry> event_registries{};
61
62     struct InputBuffer
63     {
64         std::queue<uint8_t> buffer;
65         bool empty_write{false};
66     };
67     std::map<uint64_t, InputBuffer> buffered_input_data{};
68
69     std::mutex buffer_mutex{};
70
71     std::mutex extension_mutex{};
72
73     uint64_t extension_retval{0};
74
75     IrisLogger log{};
76
77     std::atomic<bool> unblock_requested{false};
78
79     enum ExtensionState
80     {
81         XS_DISABLED, // Semihosting extensions are not supported
82         XS_DORMANT, // No ongoing semihosting extension call in progress
83         XS_WAITING_FOR_REPLY, // Event has been emitted, waiting for a reply for a client
84         XS_RETURNED, // A client instance has called semihosting_return()
85         XS_NOT_IMPLEMENTED // A client instance has called semihosting_notImplemented()
86     };
87     extension_state{XS_DISABLED};
88
89 public:
90     IrisInstanceSemihosting(IrisInstance* iris_instance = nullptr, IrisInstanceEvent* inst_event =
91         nullptr);
92
93     ~IrisInstanceSemihosting();
94
95     void attachTo(IrisInstance* iris_instance);
96
97     void setEventHandler(IrisInstanceEvent* handler);

```

```

162
177     std::vector<uint8_t> readData(uint64_t fDes, uint64_t max_size = 0, uint64_t flags =
      semihost::DEFAULT);
178
179     /*
180     * @brief Write data for a given file descriptor
181     *
182     * @param fDes      File descriptor to write to. Usually semihost::STDOUT or semihost::STDERR.
183     * @param data      Buffer containing the data to write.
184     * @param size      Size of the data buffer in bytes.
185     * @return          Returns false if no client is registered for IRIS_SEMIHOSTING_OUTPUT events.
186     */
187     bool writeData(uint64_t fDes, const uint8_t* data, uint64_t size);
188
189     void enableExtensions();
190
191     std::pair<bool, uint64_t> semihostedCall(uint64_t operation, uint64_t parameter);
192
193     void unblock();
194
195 private:
196     void impl_semihosting_provideInputData(IrisReceivedRequest& request);
197
198     void impl_semihosting_return(IrisReceivedRequest& request);
199
200     void impl_semihosting_notImplemented(IrisReceivedRequest& request);
201
202     IrisErrorCode createEventStream(EventStream* stream_out, const EventSourceInfo& info,
      const std::vector<std::string>& requested_fields);
203
204     void notifyCall(uint64_t operation, uint64_t parameter);
205
206     class SemihostingEventStream;
207
208     IrisErrorCode enableEventStream(EventStream* stream, unsigned event_type);
209     IrisErrorCode disableEventStream(EventStream* stream, unsigned event_type);
210 };
211
212 namespace IRIS_END
213
214 #endif // ARM_INCLUDE_IrisInstanceSemihosting_h

```

9.41 IrisInstanceSimulation.h File Reference

IrisInstance add-on to implement simulation_* functions.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include "iris/IrisInstantiationContext.h"
#include <map>
#include <mutex>
#include <string>
#include <vector>

```

Classes

- class [iris::IrisInstanceSimulation](#)
An *IrisInstance* add-on that adds simulation functions for the *SimulationEngine* instance.
- class [iris::IrisSimulationResetContext](#)
Provides context to a reset delegate call.

Typedefs

- typedef IrisDelegate< std::vector< ResourceInfo > & > [iris::SimulationGetParameterInfoDelegate](#)
Delegate to get a list of parameter information.
- typedef IrisDelegate< InstantiationResult & > [iris::SimulationInstantiateDelegate](#)
Delegate to instantiate the simulation.
- typedef IrisDelegate [iris::SimulationRequestShutdownDelegate](#)

Delegate to request that the simulation be shut down.

- typedef IrisDelegate< const IrisSimulationResetContext & > [iris::SimulationResetDelegate](#)

Delegate to reset the simulation.

- typedef IrisDelegate< const InstantiationParameterValue & > [iris::SimulationSetParameterValueDelegate](#)

Delegate to set the value of an instantiation parameter.

Enumerations

- enum [iris::IrisSimulationPhase](#) {
 IRIS_SIM_PHASE_INITIAL_PLUGIN_LOADING_COMPLETE , IRIS_SIM_PHASE_INSTANTIATE_ENTER , IRIS_SIM_PHASE_INSTANTIATE_LEAVE ,
 IRIS_SIM_PHASE_INIT_ENTER , IRIS_SIM_PHASE_INIT_LEAVE , IRIS_SIM_PHASE_BEFORE_END_OF_ELABORATION ,
 IRIS_SIM_PHASE_END_OF_ELABORATION , IRIS_SIM_PHASE_INITIAL_RESET_ENTER , IRIS_SIM_PHASE_INITIAL_RESET_LEAVE ,
 IRIS_SIM_PHASE_START_OF_SIMULATION , IRIS_SIM_PHASE_RESET_ENTER , IRIS_SIM_PHASE_RESET_LEAVE ,
 IRIS_SIM_PHASE_END_OF_SIMULATION , IRIS_SIM_PHASE_TERMINATE_ENTER , IRIS_SIM_PHASE_TERMINATE_LEAVE ,
 IRIS_SIM_PHASE_NUM }
List of IRIS_SIMULATION_PHASE events.

9.41.1 Detailed Description

IrisInstance add-on to implement simulation_* functions.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.41.2 Typedef Documentation

9.41.2.1 SimulationGetParameterInfoDelegate

```
typedef IrisDelegate<std::vector<ResourceInfo>&> iris::SimulationGetParameterInfoDelegate
```

Delegate to get a list of parameter information.

```
IrisErrorCode getInstantiationParameterInfo(std::vector<ResourceInfo> &parameters_out)
```

9.41.2.2 SimulationInstantiateDelegate

```
typedef IrisDelegate<InstantiationResult&> iris::SimulationInstantiateDelegate
```

Delegate to instantiate the simulation.

```
IrisErrorCode instantiate(InstantiationResult &result_out)
```

9.41.2.3 SimulationRequestShutdownDelegate

```
typedef IrisDelegate iris::SimulationRequestShutdownDelegate
```

Delegate to request that the simulation be shut down.

```
IrisErrorCode requestShutdown()
```

9.41.2.4 SimulationResetDelegate

```
typedef IrisDelegate<const IrisSimulationResetContext&> iris::SimulationResetDelegate
```

Delegate to reset the simulation.

```
IrisErrorCode reset(const IrisSimulationResetContext &)
```

9.41.2.5 SimulationSetParameterValueDelegate

typedef IrisDelegate<const InstantiationParameterValue&> [iris::SimulationSetParameterValueDelegate](#)
 Delegate to set the value of an instantiation parameter.
 IrisErrorCode setInstantiationParameterValue(const InstantiationParameterValue &value)

9.42 IrisInstanceSimulation.h

[Go to the documentation of this file.](#)

```

1
2 #ifndef ARM_INCLUDE_IrisInstanceSimulation_h
3 #define ARM_INCLUDE_IrisInstanceSimulation_h
4
5 #include "iris/detail/IrisCommon.h"
6 #include "iris/detail/IrisDelegate.h"
7 #include "iris/detail/IrisLogger.h"
8 #include "iris/detail/IrisObjects.h"
9
10 #include "iris/IrisInstantiationContext.h"
11
12 #include <map>
13 #include <mutex>
14 #include <string>
15 #include <vector>
16
17 NAMESPACE_IRIS_START
18
19 class IrisInstance;
20 class IrisReceivedRequest;
21 class IrisInstanceEvent;
22 class IrisEventRegistry;
23
24 class EventStream;
25
26 typedef IrisDelegate<InstantiationResult&> SimulationInstantiateDelegate;
27
28 class IrisSimulationResetContext
29 {
30 private:
31     static const uint64_t ALLOW_PARTIAL = (1 << 0);
32
33     uint64_t flags;
34
35     bool getFlag(uint64_t mask) const
36     {
37         return (flags & mask) != 0;
38     }
39
40     void setFlag(uint64_t mask, bool value)
41     {
42         flags &= ~mask;
43         flags |= (value ? mask : 0);
44     }
45
46 public:
47     IrisSimulationResetContext()
48         : flags(0)
49     {
50     }
51
52     bool getAllowPartialReset() const
53     {
54         return getFlag(ALLOW_PARTIAL);
55     }
56
57     // Set/clear the allowPartialReset flag.
58     void setAllowPartialReset(bool value = true)
59     {
60         setFlag(ALLOW_PARTIAL, value);
61     }
62 };
63
64 typedef IrisDelegate<const IrisSimulationResetContext&> SimulationResetDelegate;
65
66 typedef IrisDelegate<> SimulationRequestShutdownDelegate;
67
68 typedef IrisDelegate<std::vector<ResourceInfo>&&> SimulationGetParameterInfoDelegate;
69
70 typedef IrisDelegate<const InstantiationParameterValue&> SimulationSetParameterValueDelegate;
71
72 enum IrisSimulationPhase
73 {
74     IRIS_SIM_PHASE_INITIAL_PLUGIN_LOADING_COMPLETE,

```

```

121     IRIS_SIM_PHASE_INSTANTIATE_ENTER,
122     IRIS_SIM_PHASE_INSTANTIATE,
123     IRIS_SIM_PHASE_INSTANTIATE_LEAVE,
124     IRIS_SIM_PHASE_INIT_ENTER,
125     IRIS_SIM_PHASE_INIT,
126     IRIS_SIM_PHASE_INIT_LEAVE,
127     IRIS_SIM_PHASE_BEFORE_END_OF_ELABORATION,
128     IRIS_SIM_PHASE_END_OF_ELABORATION,
129     IRIS_SIM_PHASE_INITIAL_RESET_ENTER,
130     IRIS_SIM_PHASE_INITIAL_RESET,
131     IRIS_SIM_PHASE_INITIAL_RESET_LEAVE,
132     IRIS_SIM_PHASE_START_OF_SIMULATION,
133     IRIS_SIM_PHASE_RESET_ENTER,
134     IRIS_SIM_PHASE_RESET,
135     IRIS_SIM_PHASE_RESET_LEAVE,
136     IRIS_SIM_PHASE_END_OF_SIMULATION,
137     IRIS_SIM_PHASE_TERMINATE_ENTER,
138     IRIS_SIM_PHASE_TERMINATE,
139     IRIS_SIM_PHASE_TERMINATE_LEAVE,
140     IRIS_SIM_PHASE_NUM
141 };
142 static const size_t IrisSimulationPhase_total = IRIS_SIM_PHASE_NUM;
143
144 class IrisInstanceSimulation
145 {
146 private:
147     IrisInstance* iris_instance;
148
149     IrisConnectionInterface* connection_interface;
150
151     SimulationInstantiateDelegate instantiate;
152
153     SimulationResetDelegate reset;
154
155     SimulationRequestShutdownDelegate requestShutdown;
156
157     SimulationGetParameterInfoDelegate getParameterInfo;
158
159     SimulationSetParameterValueDelegate setParameterValue;
160
161     enum
162     {
163         CACHE_DISABLED,
164         CACHE_EMPTY,
165         CACHE_SET
166     } parameter_info_cache_state;
167
168     std::vector<ResourceInfo> cached_parameter_info;
169
170     std::mutex mutex;
171
172     std::vector<IrisEventRegistry*> simulation_phase_event_registries;
173
174     std::map<uint64_t, IrisSimulationPhase> evSrcId_to_phase;
175
176     IrisLogger log;
177
178     bool simulation_has_been_initialised;
179
180     std::vector<uint64_t> requests_waiting_for_instantiation;
181
182     unsigned logLevel{};
183
184 public:
185     IrisInstanceSimulation(IrisInstance* iris_instance = nullptr,
186                           IrisConnectionInterface* connection_interface = nullptr);
187     ~IrisInstanceSimulation();
188
189     void attachTo(IrisInstance* iris_instance);
190
191     void setConnectionInterface(IrisConnectionInterface* connection_interface_)
192     {
193         connection_interface = connection_interface_;
194     }
195
196     void setInstantiateDelegate(SimulationInstantiateDelegate delegate)
197     {
198         instantiate = delegate;
199     }
200
201     template <typename T, IrisErrorCode (T::*METHOD)(InstantiationResult*)>
202     void setInstantiateDelegate(T* instance)
203     {
204         setInstantiateDelegate(SimulationInstantiateDelegate::make<T, METHOD>(instance));
205     }
206
207     template <IrisErrorCode (*FUNC)(InstantiationResult*)>

```

```

268 void setInstantiateDelegate()
269 {
270     setInstantiateDelegate(SimulationInstantiateDelegate::make<FUNC>());
271 }
272
273 void setResetDelegate(SimulationResetDelegate delegate)
274 {
275     reset = delegate;
276 }
277
278 template <typename T, IrisErrorCode (T::*METHOD)(const IrisSimulationResetContext&)>
279 void setResetDelegate(T* instance)
280 {
281     setResetDelegate(SimulationResetDelegate::make<T, METHOD>(instance));
282 }
283
284 template <IrisErrorCode (*FUNC)(const IrisSimulationResetContext&)>
285 void setResetDelegate()
286 {
287     setResetDelegate(SimulationResetDelegate::make<FUNC>());
288 }
289
290 void setRequestShutdownDelegate(SimulationRequestShutdownDelegate delegate)
291 {
292     requestShutdown = delegate;
293 }
294
295 template <typename T, IrisErrorCode (T::*METHOD)()>
296 void setRequestShutdownDelegate(T* instance)
297 {
298     setRequestShutdownDelegate(SimulationRequestShutdownDelegate::make<T, METHOD>(instance));
299 }
300
301 template <IrisErrorCode (*FUNC)()>
302 void setRequestShutdownDelegate()
303 {
304     setRequestShutdownDelegate(SimulationRequestShutdownDelegate::make<FUNC>());
305 }
306
307 void setGetParameterInfoDelegate(SimulationGetParameterInfoDelegate delegate, bool cache_result =
true)
308 {
309     getParameterInfo = delegate;
310     parameter_info_cache_state = cache_result ? CACHE_EMPTY : CACHE_DISABLED;
311     cached_parameter_info.clear();
312 }
313
314 template <typename T, IrisErrorCode (T::*METHOD)(std::vector<ResourceInfo>&)>
315 void setGetParameterInfoDelegate(T* instance, bool cache_result = true)
316 {
317     typedef SimulationGetParameterInfoDelegate D;
318     setGetParameterInfoDelegate(D::make<T, METHOD>(instance), cache_result);
319 }
320
321 template <IrisErrorCode (*FUNC)(std::vector<ResourceInfo>&)>
322 void setGetParameterInfoDelegate(bool cache_result = true)
323 {
324     typedef SimulationGetParameterInfoDelegate D;
325     setGetParameterInfoDelegate(D::make<FUNC>(), cache_result);
326 }
327
328 void setSetParameterValueDelegate(SimulationSetParameterValueDelegate delegate)
329 {
330     setParameterValue = delegate;
331 }
332
333 template <typename T, IrisErrorCode (T::*METHOD)(const InstantiationParameterValue&)>
334 void setSetParameterValueDelegate(T* instance)
335 {
336     setSetParameterValueDelegate(SimulationSetParameterValueDelegate::make<T, METHOD>(instance));
337 }
338
339 template <IrisErrorCode (*FUNC)(const InstantiationParameterValue&)>
340 void setSetParameterValueDelegate()
341 {
342     setSetParameterValueDelegate(SimulationSetParameterValueDelegate::make<FUNC>());
343 }
344
345 void enterPostInstantiationPhase();
346
347 void setEventHandler(IrisInstanceEvent* handler);
348
349 void notifySimPhase(uint64_t time, IrisSimulationPhase phase);
350
351 void registerSimEventsOnGlobalInstance();
352
353 static std::string getSimulationPhaseName(IrisSimulationPhase phase);

```

```

488
494     static std::string getSimulationPhaseDescription(IrisSimulationPhase phase);
495
501     void setLogLevel(unsigned logLevel_);
502
503 private:
505     void impl_simulation_getInstantiationParameterInfo(IrisReceivedRequest& request);
506
508     void impl_simulation_setInstantiationParameterValues(IrisReceivedRequest& request);
509
511     void impl_simulation_instantiate(IrisReceivedRequest& request);
512
514     void impl_simulation_reset(IrisReceivedRequest& request);
515
517     void impl_simulation_requestShutdown(IrisReceivedRequest& request);
518
520     void impl_simulation_waitForInstantiation(IrisReceivedRequest& request);
521
523     IrisErrorCode createEventStream(EventStream*& event_stream_out, const EventSourceInfo& info,
524                                     const std::vector<std::string>& fields);
525 };
526
527 namespace IRIS_END
528
529 #endif // ARM_INCLUDE_IrisInstanceSimulation_h

```

9.43 IrisInstanceSimulationTime.h File Reference

IrisInstance add-on to implement simulationTime functions.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include <string>
#include <vector>
#include <functional>
```

Classes

- class `iris::IrisInstanceSimulationTime`
Simulation time add-on for `IrisInstance`.

Typedefs

- typedef IrisDelegate< uint64_t &, uint64_t &, bool & > [iris::SimulationTimeGetDelegate](#)
Delegate to get the simulation time.
- typedef IrisDelegate [iris::SimulationTimeRunDelegate](#)
Delegate to resume the simulation time progress.
- typedef IrisDelegate [iris::SimulationTimeStopDelegate](#)
Delegate to stop the simulation time progress.

Enumerations

- enum iris::TIME_EVENT_REASON {
 iris::TIME_EVENT_NO_REASON = 0 , iris::TIME_EVENT_UNKNOWN = (1 << 0) , iris::TIME_EVENT_STOP
 = (1 << 1) , iris::TIME_EVENT_BREAKPOINT = (1 << 2) ,
 iris::TIME_EVENT_EVENT_COUNTER_OVERFLOW = (1 << 3) , iris::TIME_EVENT_STEPPING_COMPLETED
 = (1 << 4) , iris::TIME_EVENT_REACHED_DEBUGGABLE_STATE = (1 << 5) , iris::TIME_EVENT_EVENT
 = (1 << 6) ,
 iris::TIME_EVENT_STATE_CHANGED = (1 << 7) }
- The reasons why the simulation time stopped. Bit masks.*

9.43.1 Detailed Description

IrisInstance add-on to implement simulationTime functions.

Copyright

Copyright (C) 2017-2023 Arm Limited. All rights reserved.

9.43.2 Typedef Documentation

9.43.2.1 SimulationTimeGetDelegate

```
typedef IrisDelegate<uint64_t&, uint64_t&, bool&> iris::SimulationTimeGetDelegate
```

Delegate to get the simulation time.

```
IrisErrorCode getTime(uint64_t &ticks, uint64_t &tickHz, bool &running);
```

9.43.2.2 SimulationTimeRunDelegate

```
typedef IrisDelegate iris::SimulationTimeRunDelegate
```

Delegate to resume the simulation time progress.

```
IrisErrorCode run();
```

9.43.2.3 SimulationTimeStopDelegate

```
typedef IrisDelegate iris::SimulationTimeStopDelegate
```

Delegate to stop the simulation time progress.

```
IrisErrorCode stop();
```

9.43.3 Enumeration Type Documentation

9.43.3.1 TIME_EVENT_REASON

```
enum iris::TIME_EVENT_REASON
```

The reasons why the simulation time stopped. Bit masks.
Note that Fast Models only ever emits TIME_EVENT_UNKNOWN.

Enumerator

TIME_EVENT_NO_REASON	Do not emit a REASON field.
TIME_EVENT_UNKNOWN	Simulation stopped for any reason.
TIME_EVENT_STOP	simulationTime_stop() was called.
TIME_EVENT_BREAKPOINT	Breakpoint was hit.
TIME_EVENT_EVENT_COUNTER_OVERFLOW	EventCounterMode.overflowStopSim.
TIME_EVENT_STEPPING_COMPLETED	step_setup() and then simulationTime_run().
TIME_EVENT_REACHED_DEBUGGABLE_STATE	simulationTime_runUntilDebuggableState().
TIME_EVENT_EVENT	eventStream_create(stop=true).
TIME_EVENT_STATE_CHANGED	State of any component changed.

9.44 IrisInstanceSimulationTime.h

[Go to the documentation of this file.](#)

```
1
8 #ifndef ARM_INCLUDE_IrisInstanceSimulationTime_h
9 #define ARM_INCLUDE_IrisInstanceSimulationTime_h
10
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisDelegate.h"
```

Generated by Doxygen

```

178     }
179
185     void setSimTimeGetDelegate(SimulationTimeGetDelegate delegate)
186     {
187         get_time_delegate = delegate;
188     }
189
197     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t&, uint64_t&, bool&)>
198     void setSimTimeGetDelegate(T* instance)
199     {
200         setSimTimeGetDelegate(SimulationTimeGetDelegate::make<T, METHOD>(instance));
201     }
202
210     template <IrisErrorCode (*FUNC)(uint64_t&, uint64_t&, bool&)>
211     void setSimTimeGetDelegate()
212     {
213         setSimTimeGetDelegate(SimulationTimeGetDelegate::make<FUNC>());
214     }
215
233     void setSimTimeNotifyStateChanged(std::function<void()> func)
234     {
235         notify_state_changed_delegate = func;
236     }
237
239     void notifySimulationTimeEvent(uint64_t reason = TIME_EVENT_UNKNOWN);
240
252     void registerSimTimeEventsOnGlobalInstance();
253
254 private:
256     void impl_simulationTime_run(IrisReceivedRequest& request);
257     void impl_simulationTime_stop(IrisReceivedRequest& request);
258     void impl_simulationTime_get(IrisReceivedRequest& request);
259     void impl_simulationTime_notifyStateChanged(IrisReceivedRequest& request);
260
261     IrisErrorCode createEventStream(EventStream*&, const EventSourceInfo&, const
std::vector<std::string>&);
262 };
263
264 NAMESPACE_IRIS_END
265
266 #endif // ARM_INCLUDE_IrisInstanceSimulationTime_h

```

9.45 IrisInstanceStep.h File Reference

Stepping-related add-on to an IrisInstance.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include <cstdio>

```

Classes

- class [iris::IrisInstanceStep](#)
Step add-on for [IrisInstance](#).

Typedefs

- typedef IrisDelegate< uint64_t &, const std::string & > [iris::RemainingStepGetDelegate](#)
Delegate to get the value of the currently remaining steps.
- typedef IrisDelegate< uint64_t, const std::string & > [iris::RemainingStepSetDelegate](#)
Delegate to set the remaining steps measured in the specified unit.
- typedef IrisDelegate< uint64_t &, const std::string & > [iris::StepCountGetDelegate](#)
Delegate to get the value of the step count.

9.45.1 Detailed Description

Stepping-related add-on to an IrisInstance.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

The IrisInstanceStep class implements all stepping-related Iris functions.

9.45.2 Typedef Documentation

9.45.2.1 RemainingStepGetDelegate

```
typedef IrisDelegate<uint64_t&, const std::string&> iris::RemainingStepGetDelegate
```

Delegate to get the value of the currently remaining steps.

```
IrisErrorCode getRemainingSteps(uint64_t &steps, const std::string &unit)
```

Error: Return E_* error code if it failed to get the remaining steps.

9.45.2.2 RemainingStepSetDelegate

```
typedef IrisDelegate<uint64_t, const std::string&> iris::RemainingStepSetDelegate
```

Delegate to set the remaining steps measured in the specified unit.

```
IrisErrorCode setRemainingSteps(uint64_t steps, const std::string &unit)
```

Error: Return E_* error code if it failed to set the steps.

9.45.2.3 StepCountGetDelegate

```
typedef IrisDelegate<uint64_t&, const std::string&> iris::StepCountGetDelegate
```

Delegate to get the value of the step count.

```
IrisErrorCode getStepCount(uint64_t &count, const std::string &unit)
```

Error: Return E_* error code if it failed to get the step count.

9.46 IrisInstanceStep.h

[Go to the documentation of this file.](#)

```
1
2
3
4
5
6
7
8
9 #ifndef ARM_INCLUDE_IrisInstanceStep_h
10 #define ARM_INCLUDE_IrisInstanceStep_h
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisDelegate.h"
14 #include "iris/detail/IrisLogger.h"
15 #include "iris/detail/IrisObjects.h"
16
17 #include <cstdio>
18
19 NAMESPACE_IRIS_START
20
21 class IrisInstance;
22 class IrisReceivedRequest;
23
24 typedef IrisDelegate<uint64_t, const std::string&> RemainingStepSetDelegate;
25
26 typedef IrisDelegate<uint64_t&, const std::string&> RemainingStepGetDelegate;
27
28 typedef IrisDelegate<uint64_t&, const std::string&> StepCountGetDelegate;
29
30 class IrisInstanceStep
31 {
32 public:
33     IrisInstanceStep(IrisInstance* irisInstance = nullptr);
34
35     void attachTo(IrisInstance* irisInstance);
36
37     void setRemainingStepSetDelegate(RemainingStepSetDelegate delegate);
38
39     void setRemainingStepGetDelegate(RemainingStepGetDelegate delegate);
40
41     void setStepCountGetDelegate(StepCountGetDelegate delegate);
42
43 private:
44     void impl_step_setup(IrisReceivedRequest& request);
45
46     void impl_step_getRemainingSteps(IrisReceivedRequest& request);
47
48 }
```

```

105
106 void impl_step_getStepCounterValue(IrisReceivedRequest& request);
107
108 void impl_step_syncStep(IrisReceivedRequest& request);
109
110 void impl_step_syncStepSetup(IrisReceivedRequest& request);
111
112
113
114
115 IrisInstance* irisInstance;
116
117 RemainingStepSetDelegate stepSetDel;
118 RemainingStepGetDelegate stepGetDel;
119
120
121 StepCountGetDelegate stepCountGetDel;
122
123
124
125 IrisLogger log;
126 };
127
128 NAMESPACE_IRIS_END
129
130 #endif // #ifndef ARM_INCLUDE_IrisInstanceStep_h

```

9.47 IrisInstanceTable.h File Reference

Table add-on to IrisInstance.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisObjects.h"

```

Classes

- class [iris::IrisInstanceTable](#)
Table add-on for [IrisInstance](#).
- struct [iris::IrisInstanceTable::TableInfoAndAccess](#)
Entry in 'tableInfos'.

Typedefs

- typedef IrisDelegate< const TableInfo &, uint64_t, uint64_t, TableReadResult & > [iris::TableReadDelegate](#)
Delegate to read table data.
- typedef IrisDelegate< const TableInfo &, const TableRecords &, TableWriteResult & > [iris::TableWriteDelegate](#)
Delegate to write table data.

9.47.1 Detailed Description

Table add-on to IrisInstance.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

The IrisInstanceTable class implements all table-related Iris functions.

9.47.2 Typedef Documentation

9.47.2.1 TableReadDelegate

```
typedef IrisDelegate<const TableInfo&, uint64_t, uint64_t, TableReadResult&> iris::TableReadDelegate
```

Delegate to read table data.

```
IrisErrorCode read(const TableInfo &tableInfo, uint64_t index, uint64_t count, TableReadResult &result)
```

tableInfo, index, and count are guaranteed to be valid. count is non-zero.

TableReadResult holds the read results and any errors from reading table cell values.

9.47.2.2 TableWriteDelegate

typedef IrisDelegate<const TableInfo&, const TableRecords&, TableWriteResult&> [iris::TableWriteDelegate](#)
 Delegate to write table data.

IrisErrorCode write(const TableInfo &tableInfo, const TableRecords &records, TableWriteResult &result)

records is guaranteed to be non-empty.

TableWriteResult holds any errors from writing table cell values.

9.48 IrisInstanceTable.h

[Go to the documentation of this file.](#)

```

1
9 #ifndef ARM_INCLUDE_IrisInstanceTable_h
10 #define ARM_INCLUDE_IrisInstanceTable_h
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisDelegate.h"
14 #include "iris/detail/IrisObjects.h"
15
16 NAMESPACE_IRIS_START
17
18 class IrisInstance;
19 class IrisReceivedRequest;
20
31 typedef IrisDelegate<const TableInfo&, uint64_t, uint64_t, TableReadResult&> TableReadDelegate;
32
43 typedef IrisDelegate<const TableInfo&, const TableRecords&, TableWriteResult&> TableWriteDelegate;
44
50 class IrisInstanceTable
51 {
52 public:
58     struct TableInfoAndAccess
59     {
60         TableInfo      tableInfo;
61         TableReadDelegate readDelegate;
62         TableWriteDelegate writeDelegate;
63     };
64
70     IrisInstanceTable(IrisInstance* irisInstance = nullptr);
71
79     void attachTo(IrisInstance* irisInstance);
80
88     TableInfoAndAccess& addTableInfo(const std::string& name);
89
96     void setDefaultReadDelegate(TableReadDelegate delegate = TableReadDelegate())
97     {
98         defaultReadDelegate = delegate;
99     }
100
107     void setDefaultWriteDelegate(TableWriteDelegate delegate = TableWriteDelegate())
108     {
109         defaultWriteDelegate = delegate;
110     }
111
112 private:
113     void impl_table_getList(IrisReceivedRequest& request);
114     void impl_table_read(IrisReceivedRequest& request);
115     void impl_table_write(IrisReceivedRequest& request);
116
117     void impl_table_write(IrisReceivedRequest& request);
118
119     IrisInstance* irisInstance;
120
121     typedef std::vector<TableInfoAndAccess> TableInfoAndAccessList;
122     TableInfoAndAccessList tableInfos;
123
124     TableReadDelegate defaultReadDelegate;
125     TableWriteDelegate defaultWriteDelegate;
126 };
127
128 NAMESPACE_IRIS_END
129
130 #endif // #ifndef ARM_INCLUDE_IrisInstanceTable_h

```

9.49 IrisInstantiationContext.h File Reference

Helper class used to instantiate Iris instances from generic factories.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisObjects.h"
#include "iris/detail/IrisUtils.h"
#include <string>
#include <vector>
```

Classes

- class [iris::IrisInstantiationContext](#)

Provides context when instantiating an Iris instance from a factory.

9.49.1 Detailed Description

Helper class used to instantiate Iris instances from generic factories.

Copyright

Copyright (C) 2017-2023 Arm Limited. All rights reserved.

9.50 IrisInstantiationContext.h

[Go to the documentation of this file.](#)

```
1
2
3 #ifndef ARM_INCLUDE_IrisInstantiationContext_h
4 #define ARM_INCLUDE_IrisInstantiationContext_h
5
6 #include "iris/detail/IrisCommon.h"
7 #include "iris/detail/IrisObjects.h"
8 #include "iris/detail/IrisUtils.h"
9
10 #include <string>
11 #include <vector>
12
13 NAMESPACE_IRIS_START
14
15 class IrisInstantiationContext
16 {
17 private:
18     IrisConnectionInterface* connection_interface;
19
20     InstantiationResult& result;
21
22     IrisValueMap params;
23
24     std::string prefix;
25
26     std::string component_name;
27
28     uint64_t instance_flags;
29
30     std::vector<IrisInstantiationContext*> children;
31
32     void errorInternal(const std::string& severity,
33                       const std::string& code,
34                       const std::string& parameterName,
35                       const char* format,
36                       va_list args);
37
38     void processParameters(const std::vector<ResourceInfo>& param_info_,
39                           const std::vector<InstantiationParameterValue>& param_values_);
40
41     IrisInstantiationContext(const IrisInstantiationContext* parent, const std::string& instance_name);
42
43 public:
44     IrisInstantiationContext(IrisConnectionInterface* connection_interface_,
45                             InstantiationResult& result_,
46                             const std::vector<ResourceInfo>& param_info_,
47                             const std::vector<InstantiationParameterValue>& param_values_,
48                             const std::string& prefix_,
49                             const std::string& component_name_,
50                             uint64_t instance_flags_);
51
52     ~IrisInstantiationContext();
53
54     IrisInstantiationContext* getSubcomponentContext(const std::string& child_name);
```

```

86
96     template <typename T>
97     void getParameter(const std::string& name, T& value)
98     {
99         getParameter(name).get(value);
100     }
101
102     const IrisValue& getParameter(const std::string& name)
103     {
104         IrisValueMap::const_iterator it = params.find(name);
105         if (it == params.end())
106         {
107             throw IrisInternalError("getParameter(" + name + "): Unknown parameter");
108         }
109         return it->second;
110     }
111
112     std::string getStringParameter(const std::string& name)
113     {
114         return getParameter(name).getAsString();
115     }
116
117     uint64_t getU64Parameter(const std::string& name)
118     {
119         return getParameter(name).getAsU64();
120     }
121
122     int64_t getS64Parameter(const std::string& name)
123     {
124         return getParameter(name).getAsS64();
125     }
126
127     bool getBoolParameter(const std::string& name)
128     {
129         return getParameter(name).getAsBool();
130     }
131
132     void getParameter(const std::string& name, std::vector<uint64_t>& value);
133
134     uint64_t getRecommendedInstanceFlags() const
135     {
136         return instance_flags;
137     }
138
139     std::string getInstanceName() const
140     {
141         return prefix + "." + component_name;
142     }
143
144     IrisConnectionInterface* getConnectionInterface() const
145     {
146         return connection_interface;
147     }
148
149     void warning(const std::string& code, const char* format, ...) INTERNAL_IRIS_PRINTF(3, 4);
150
151     void parameterWarning(const std::string& code, const std::string& parameterName, const char* format,
152 ... ) INTERNAL_IRIS_PRINTF(4, 5);
153
154     void error(const std::string& code, const char* format, ...) INTERNAL_IRIS_PRINTF(3, 4);
155
156     void parameterError(const std::string& code, const std::string& parameterName, const char* format,
157 ... ) INTERNAL_IRIS_PRINTF(4, 5);
158 };
159
160 namespace IRIS
161 {
162     #endif // ARM_INCLUDE_IrisInstantiationContext_h

```

9.51 IrisParameterBuilder.h File Reference

Helper class to construct instantiation parameters.

```

#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisObjects.h"
#include <string>
#include <vector>

```

Classes

- class [iris::IrisParameterBuilder](#)

Helper class to construct instantiation parameters.

9.51.1 Detailed Description

Helper class to construct instantiation parameters.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.52 IrisParameterBuilder.h

[Go to the documentation of this file.](#)

```

1
2 #ifndef ARM_INCLUDE_IrisParameterBuilder_h
3 #define ARM_INCLUDE_IrisParameterBuilder_h
4
5 #include "iris/detail/IrisCommon.h"
6 #include "iris/detail/IrisObjects.h"
7
8 #include <string>
9 #include <vector>
10
11 NAMESPACE_IRIS_START
12
13 class IrisParameterBuilder
14 {
15 private:
16     ResourceInfo& info;
17
18     IrisParameterBuilder& setValueExtend(std::vector<uint64_t>& arr, uint64_t value, uint64_t extension)
19     {
20         arr.resize(info.getDataSizeInU64Chunks(), extension);
21         arr[0] = value;
22         return *this;
23     }
24
25     IrisParameterBuilder& setValueExtend(std::vector<uint64_t>& arr, const std::vector<uint64_t>& value,
26     uint64_t extension)
27     {
28         size_t param_size = info.getDataSizeInU64Chunks();
29         if (param_size < value.size())
30         {
31             throw IrisInternalError("Invalid parameter configuration");
32         }
33         arr = value;
34         arr.resize(info.getDataSizeInU64Chunks(), extension);
35         return *this;
36     }
37
38     IrisParameterBuilder& setValueSignExtend(std::vector<uint64_t>& arr, int64_t value)
39     {
40         return setValueExtend(arr, static_cast<uint64_t>(value), (value < 0) ? IRIS_UINT64_MAX : 0);
41     }
42
43     IrisParameterBuilder& setValueZeroExtend(std::vector<uint64_t>& arr, uint64_t value)
44     {
45         return setValueExtend(arr, value, 0);
46     }
47
48     IrisParameterBuilder& setValueSignExtend(std::vector<uint64_t>& arr, const std::vector<uint64_t>&
49     value)
50     {
51         return setValueExtend(arr, value, (static_cast<int64_t>(value.back()) < 0) ? IRIS_UINT64_MAX :
52         0);
53     }
54
55     IrisParameterBuilder& setValueZeroExtend(std::vector<uint64_t>& arr, const std::vector<uint64_t>&
56     value)
57     {
58         return setValueExtend(arr, value, 0);
59     }
60
61     IrisParameterBuilder& setValueDouble(std::vector<uint64_t>& arr, double value)
62     {
63         arr.resize(1);
64         *static_cast<double*>((void*) (&arr[0])) = value;
65         return *this;
66     }
67
68
69
70
71
72

```

```

73     }
74
75 public:
80     IrisParameterBuilder(ResourceInfo& info_)
81         : info(info_)
82     {
83         info.isParameter = true;
84     }
85
91     IrisParameterBuilder& setName(const std::string& name)
92     {
93         info.name = name;
94         return *this;
95     }
96
102    IrisParameterBuilder& setDescr(const std::string& description)
103    {
104        info.description = description;
105        return *this;
106    }
107
113    IrisParameterBuilder& setFormat(const std::string& format)
114    {
115        info.format = format;
116        return *this;
117    }
118
124    IrisParameterBuilder& setBitWidth(uint64_t bitWidth)
125    {
126        info.bitWidth = bitWidth;
127        return *this;
128    }
129
135    IrisParameterBuilder& setRwMode(const std::string& rwMode)
136    {
137        info.rwMode = rwMode;
138        return *this;
139    }
140
146    IrisParameterBuilder& setSubRscId(uint64_t subRscId)
147    {
148        info.subRscId = subRscId;
149        return *this;
150    }
151
157    IrisParameterBuilder& setTopology(bool value = true)
158    {
159        info.parameterInfo.topology = value;
160        return *this;
161    }
162
168    IrisParameterBuilder& setInitOnly(bool value = true)
169    {
170        info.parameterInfo.initOnly = value;
171        return *this;
172    }
173
179    IrisParameterBuilder& setMin(uint64_t min)
180    {
181        return setValueZeroExtend(info.parameterInfo.min, min);
182    }
183
189    IrisParameterBuilder& setMax(uint64_t max)
190    {
191        return setValueZeroExtend(info.parameterInfo.max, max);
192    }
193
200    IrisParameterBuilder& setRange(uint64_t min, uint64_t max)
201    {
202        return setMin(min).setMax(max);
203    }
204
213    IrisParameterBuilder& setMin(const std::vector<uint64_t>& min)
214    {
215        return setValueZeroExtend(info.parameterInfo.min, min);
216    }
217
226    IrisParameterBuilder& setMax(const std::vector<uint64_t>& max)
227    {
228        return setValueZeroExtend(info.parameterInfo.max, max);
229    }
230
240    IrisParameterBuilder& setRange(const std::vector<uint64_t>& min, const std::vector<uint64_t>& max)
241    {
242        return setMin(min).setMax(max);
243    }
244

```

```

253     IrisParameterBuilder& setMinSigned(int64_t min)
254     {
255         return setValueSignExtend(info.parameterInfo.min, min)
256             .setType("numericSigned");
257     }
258
259     IrisParameterBuilder& setMaxSigned(int64_t max)
260     {
261         return setValueSignExtend(info.parameterInfo.max, max)
262             .setType("numericSigned");
263     }
264
265     IrisParameterBuilder& setRangeSigned(int64_t min, int64_t max)
266     {
267         return setValueSignExtend(info.parameterInfo.min, min)
268             .setValueSignExtend(info.parameterInfo.max, max)
269             .setType("numericSigned");
270     }
271
272     IrisParameterBuilder& setMinSigned(const std::vector<uint64_t>& min)
273     {
274         return setValueSignExtend(info.parameterInfo.min, min)
275             .setType("numericSigned");
276     }
277
278     IrisParameterBuilder& setMaxSigned(const std::vector<uint64_t>& max)
279     {
280         return setValueSignExtend(info.parameterInfo.max, max)
281             .setType("numericSigned");
282     }
283
284     IrisParameterBuilder& setRangeSigned(const std::vector<uint64_t>& min, const std::vector<uint64_t>&
285 max)
286     {
287         return setValueSignExtend(info.parameterInfo.min, min)
288             .setValueSignExtend(info.parameterInfo.max, max)
289             .setType("numericSigned");
290     }
291
292     IrisParameterBuilder& setMinFloat(double min)
293     {
294         return setValueDouble(info.parameterInfo.min, min)
295             .setType("numericFp");
296     }
297
298     IrisParameterBuilder& setMaxFloat(double max)
299     {
300         return setValueDouble(info.parameterInfo.max, max)
301             .setType("numericFp");
302     }
303
304     IrisParameterBuilder& setRangeFloat(double min, double max)
305     {
306         return setValueDouble(info.parameterInfo.min, min)
307             .setValueDouble(info.parameterInfo.max, max)
308             .setType("numericFp");
309     }
310
311     IrisParameterBuilder& addEnum(const std::string& symbol, const IrisValue& value, const std::string&
312 description = std::string())
313     {
314         info.enums.push_back(EnumElementInfo(value, symbol, description));
315         return *this;
316     }
317
318     IrisParameterBuilder& addStringEnum(const std::string& value, const std::string& description =
319 std::string())
320     {
321         info.enums.push_back(EnumElementInfo(IrisValue(value), std::string(), description));
322         return *this;
323     }
324
325     IrisParameterBuilder& setTag(const std::string& tag)
326     {
327         info.tags[tag] = IrisValue(true);
328         return *this;
329     }
330
331     IrisParameterBuilder& setHidden(bool hidden)
332     {
333         info.isHidden = hidden;
334         return *this;
335     }
336
337     IrisParameterBuilder& setTag(const std::string& tag, const IrisValue& value)
338     {
339         info.tags[tag] = value;

```

```

443         return *this;
444     }
445
446     IrisParameterBuilder& setDefault(const std::string& value)
447     {
448         info.parameterInfo.defaultString = value;
449         return *this;
450     }
451
452     IrisParameterBuilder& setDefault(uint64_t value)
453     {
454         return setValueZeroExtend(info.parameterInfo.defaultData, value);
455     }
456
457     IrisParameterBuilder& setDefault(const std::vector<uint64_t>& value)
458     {
459         return setValueZeroExtend(info.parameterInfo.defaultData, value);
460     }
461
462     IrisParameterBuilder& setDefaultSigned(int64_t value)
463     {
464         return setValueSignExtend(info.parameterInfo.defaultData, value);
465     }
466
467     IrisParameterBuilder& setDefaultSigned(const std::vector<uint64_t>& value)
468     {
469         return setValueSignExtend(info.parameterInfo.defaultData, value);
470     }
471
472     IrisParameterBuilder& setDefaultFloat(double value)
473     {
474         return setValueDouble(info.parameterInfo.defaultData, value);
475     }
476
477     IrisParameterBuilder& setType(const std::string& type)
478     {
479         if ((info.bitWidth != 32) && (info.bitWidth != 64) && (type == "numericFp"))
480         {
481             throw IrisInternalError(
482                 "Invalid parameter configuration."
483                 " NumericFp parameters must have a bitWidth of 32 or 64");
484         }
485
486         info.type = type;
487         return *this;
488     }
489 };
490
491 namespace iris {
492 #endif // ARM_INCLUDE_IrisParameterBuilder_h

```

9.53 IrisPluginFactory.h File Reference

A generic plug-in factory for instantiating plug-in instances.

```

#include "iris/IrisCConnection.h"
#include "iris/IrisInstance.h"
#include "iris/IrisInstanceFactoryBuilder.h"
#include "iris/IrisInstantiationContext.h"
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisFunctionInfo.h"
#include "iris/detail/IrisObjects.h"
#include "iris/detail/IrisU64JsonReader.h"
#include "iris/detail/IrisU64JsonWriter.h"
#include <mutex>
#include <string>
#include <vector>

```

Classes

- class [iris::IrisNonFactoryPlugin< PLUGIN_CLASS >](#)
Wrapper to instantiate a non-factory plugin.
- class [iris::IrisPluginFactory< PLUGIN_CLASS >](#)

- class [iris::IrisPluginFactoryBuilder](#)
Set meta data for instantiating a plug-in instance.

Macros

- `#define IRIS_NON_FACTORY_PLUGIN(PluginClassName)`
Create plugin entry point for non-factory plugins (i.e. plugins which do not have parameters and which are always instantiated just once).
- `#define IRIS_PLUGIN_FACTORY(PluginClassName)`
Create plugin entry point for plugins which have a factory (i.e. plugins which have parameters and/or plugins which are potentially instantiated multiple times).

9.53.1 Detailed Description

A generic plug-in factory for instantiating plug-in instances.

Copyright

Copyright (C) 2017-2023 Arm Limited. All rights reserved.

9.53.2 Macro Definition Documentation

9.53.2.1 IRIS_NON_FACTORY_PLUGIN

```
#define IRIS_NON_FACTORY_PLUGIN(  
    PluginClassName )
```

Value:

```
extern "C" IRIS_EXPORT int64_t irisInitPlugin(IrisC_Functions* functions)  
{  
    return ::iris::IrisNonFactoryPlugin<PluginClassName>::initPlugin(functions, #PluginClassName);  
}
```

Create plugin entry point for non-factory plugins (i.e. plugins which do not have parameters and which are always instantiated just once).

Parameters

<i>PluginClassName</i>	Class name of the plugin.
------------------------	---------------------------

9.53.2.2 IRIS_PLUGIN_FACTORY

```
#define IRIS_PLUGIN_FACTORY(  
    PluginClassName )
```

Value:

```
extern "C" IRIS_EXPORT int64_t irisInitPlugin(IrisC_Functions* functions)  
{  
    return ::iris::IrisPluginFactory<PluginClassName>::initPlugin(functions, #PluginClassName);  
}
```

Create plugin entry point for plugins which have a factory (i.e. plugins which have parameters and/or plugins which are potentially instantiated multiple times).

Parameters

<i>PluginClassName</i>	Objects of this type are instantiated for each plug-in instance created.
------------------------	--

9.54 IrisPluginFactory.h

[Go to the documentation of this file.](#)

```

1
2
3
4
5
6
7 #ifndef ARM_INCLUDE_IrisPluginFactory_h
8 #define ARM_INCLUDE_IrisPluginFactory_h
9
10 #include "iris/IrisCConnection.h"
11 #include "iris/IrisInstance.h"
12 #include "iris/IrisInstanceFactoryBuilder.h"
13 #include "iris/IrisInstantiationContext.h"
14 #include "iris/detail/IrisCommon.h"
15 #include "iris/detail/IrisFunctionInfo.h"
16 #include "iris/detail/IrisObjects.h"
17 #include "iris/detail/IrisU64JsonReader.h"
18 #include "iris/detail/IrisU64JsonWriter.h"
19
20 #include <mutex>
21 #include <string>
22 #include <vector>
23
24 NAMESPACE_IRIS_START
25
26 // Iris plugins
27 // =====
28 //
29 // This header supports declaring two different kind of plugins by using one of two macros:
30 //
31 // 1. Factory plugins:
32 //
33 // IRIS_PLUGIN_FACTORY(PluginClassName)
34 //
35 // where PluginClassName is the class of the plugin, not the factory. The factory is instantiated
36 // automatically by the macro.
37 //
38 // This declares a plugin which has a plugin factory. This type of plugin must be used
39 // for plugins which have parameters and for plugins where it makes sense to instantiate them multiple
40 // times.
41 //
42 // If unsure, use this type.
43 // PluginClassName must have this constructor and a static buildPluginFactory() function to declare the
44 // parameters:
45 //
46 // PluginClassName(iris::IrisInstantiationContext& context) { ... initialize plugin ... }
47 // static void buildPluginFactory(iris::IrisPluginFactoryBuilder& b) { ... declare parameters ... }
48 //
49 // 2. Non-factory plugins:
50 //
51 // IRIS_NON_FACTORY_PLUGIN(PluginClassName)
52 //
53 // where PluginClassName is the class of the plugin.
54 //
55 // This declares a plugin which is automatically instantiated exactly once when the DSO is loaded.
56 // The plugin cannot have parameters and cannot be instantiated multiple times. A non-factory plugin
57 // plays the same role as the factory instance of factory plugins.
58 //
59 // PluginClassName must have this constructor:
60 //
61 // PluginClassName(iris::IrisInstantiationContext& context) { ... initialize plugin ... }
62 //
63 // Both types of plugins have identical entry points (irisInitPlugin()), and the plugin loader treats
64 // them the same way.
65 // After loading a plugin DSO, the plugin loader calls irisInitPlugin() which creates a single plugin
66 // instance.
67 // This is either a plugin factory, indicated by the fact that this instance has the functions
68 // plugin_getInstantiationParameterInfo(), or a non-factory plugin, when these plugin_*() functions are not present. In
69 // the latter case the
70 // plugin loader is now done. For factory-plugins the plugin loader now instantiates all desired plugins
71 // by calling plugin_instantiate()
72 // with the respective parameter values.
73
74 class IrisPluginFactoryBuilder : public IrisInstanceFactoryBuilder
75 {
76 private:
77     std::string plugin_name;
78
79     std::string instance_name_prefix;
80
81     std::string default_instance_name;
82
83 public:
84     IrisPluginFactoryBuilder(const std::string& name)
85         : IrisInstanceFactoryBuilder(/*parameter_prefix=*/"")
86         , plugin_name(name)
87         , instance_name_prefix("client.plugin")
88     {
89     }
90

```

```

91     }
92
101    void setPluginName(const std::string& name)
102    {
103        plugin_name = name;
104    }
105
110    const std::string& getPluginName() const
111    {
112        return plugin_name;
113    }
114
123    void setInstanceNamePrefix(const std::string& prefix)
124    {
125        instance_name_prefix = prefix;
126    }
127
132    const std::string& getInstanceNamePrefix() const
133    {
134        return instance_name_prefix;
135    }
136
145    void setDefaultInstanceName(const std::string& name)
146    {
147        default_instance_name = name;
148    }
149
155    const std::string& getDefaultInstanceName() const
156    {
157        if (default_instance_name.empty())
158        {
159            return getPluginName();
160        }
161        else
162        {
163            return default_instance_name;
164        }
165    }
166 };
167
168 template <class PLUGIN_CLASS>
169 class IrisPluginFactory
170 {
171 private:
172     IrisCConnection connection_interface;
173
174     IrisInstance factory_instance;
175
176     std::vector<PLUGIN_CLASS*> plugin_instances;
177
178     std::mutex plugin_instances_mutex;
179
180     IrisPluginFactoryBuilder builder;
181
182     void impl_plugin_getInstantiationParameterInfo(IrisReceivedRequest& req)
183     {
184         factory_instance.sendResponse(req.generateOkResponse(builder.getParameterInfo()));
185     }
186
187     void impl_plugin_instantiate(IrisReceivedRequest& req)
188     {
189         InstantiationResult result;
190         result.success = true; // Assume we will succeed until proven otherwise
191
192         uint64_t instance_flags = IrisInstance::DEFAULT_FLAGS;
193
194         std::string instName;
195
196         if (!req.getOptionalArg(ISTR("instName"), instName))
197         {
198             instName = builder.getDefaultInstanceName();
199             instance_flags |= IrisInstance::UNIQUEIFY;
200         }
201
202         std::vector<InstantiationParameterValue> param_values;
203         req.getOptionalArg(ISTR("paramValues"), param_values);
204
205         // Build the full parameter info list
206         const std::vector<ResourceInfo>& param_info = builder.getParameterInfo();
207         const std::vector<ResourceInfo>& hidden_param_info = builder.getHiddenParameterInfo();
208
209         std::vector<ResourceInfo> all_param_info;
210         all_param_info.insert(all_param_info.end(), param_info.begin(), param_info.end());
211         all_param_info.insert(all_param_info.end(), hidden_param_info.begin(), hidden_param_info.end());
212
213         IrisInstantiationContext init_context(&connection_interface, result,
214                                             all_param_info, param_values,

```

```

221                                     builder.getInstanceNamePrefix(),
222                                     instName, instance_flags);
223
224     // Parameters have been validated. If they all passed we can instantiate the plugin.
225
226     if (result.success)
227     {
228         try
229         {
230             std::lock_guard<std::mutex> lock(plugin_instances_mutex);
231
232             plugin_instances.push_back(new PLUGIN_CLASS(init_context));
233
234             if (!result.success)
235             {
236                 // The plugin instance set an error in its constructor so destroy it.
237                 delete plugin_instances.back();
238                 plugin_instances.pop_back();
239             }
240         }
241         catch (IrisErrorException& e)
242         {
243             result.success = false;
244             result.errors.resize(result.errors.size() + 1);
245
246             InstantiationError& error = result.errors.back();
247             error.severity             = "error";
248             error.code                  = "error_general_error";
249             error.message               = e.getMessage();
250         }
251         catch (...)
252         {
253             result.success = false;
254             result.errors.resize(result.errors.size() + 1);
255
256             InstantiationError& error = result.errors.back();
257             error.severity             = "error";
258             error.code                  = "error_general_error";
259             error.message               = "Internal error while instantiating plugin";
260         }
261     }
262
263     factory_instance.sendResponse(req.generateOkResponse(result));
264 }
265
266 public:
267     IrisPluginFactory(IrisC_Functions* iris_c_functions, const std::string& plugin_name)
268     : connection_interface(iris_c_functions)
269     , factory_instance(&connection_interface)
270     , builder(plugin_name)
271     {
272         PLUGIN_CLASS::buildPluginFactory(builder);
273
274         typedef IrisPluginFactory<PLUGIN_CLASS> Self;
275
276         factory_instance.irisRegisterFunction(this, Self, plugin_getInstantiationParameterInfo,
277                                             function_info::plugin_getInstantiationParameterInfo);
278
279         factory_instance.irisRegisterFunction(this, Self, plugin_instantiate,
280                                             "{description:'Instantiate an instance of the " +
281                                             builder.getPluginName() +
282                                             " plugin', "
283                                             "args:{ "
284                                             "  instName:{type:'String', description:'Used to "
285                                             "construct the instance name for the new instance."
286                                             "  Instance name will be \" "
287                                             "    + builder.getInstanceNamePrefix() +
288                                             "<instName>\", "
289                                             "defval:\" "
290                                             "    + builder.getDefaultInstanceName() +
291                                             "\", optional:true}, "
292                                             "  paramValues:{type:'Array', "
293                                             "description:'Instantiation parameter values' } "
294                                             "}, "
295                                             "retval:{type:'InstantiationResult', "
296                                             "description:'Indicates success of and errors/warnings "
297                                             "that occurred during plugin instantiation.'}}");
298
299         // Register factory instance
300         uint64_t flags = IrisInstance::DEFAULT_FLAGS
301             | IrisInstance::UNIQUEIFY;
302
303         std::string factory_instName = "framework.plugin." + builder.getPluginName() + "Factory";
304         factory_instance.registerInstance(factory_instName, flags);
305         factory_instance.setProperty("componentType", "IrisPluginFactory");
306
307         IrisLogger log("IrisPluginFactory");

```



```

304     }
305
306 ~IrisPluginFactory()
307 {
308     {
309         std::lock_guard<std::mutex> lock(plugin_instances_mutex);
310
311         // Clean up plugin instances
312         typename std::vector<PLUGIN_CLASS*>::iterator it;
313         for (it = plugin_instances.begin(); it != plugin_instances.end(); ++it)
314         {
315             delete *it;
316         }
317     }
318 }
319
320 // Unregister factory instance. Call this when unloading a plugin before simulation termination.
321 IrisErrorCode unregisterInstance()
322 {
323     return factory_instance.unregisterInstance();
324 }
325
326 // Implementation of the plugin entry point.
327 // This will initialize an IrisPluginFactory the first time it is called.
328 static int64_t initPlugin(IrisC_Funcions* functions, const std::string& plugin_name)
329 {
330     static IrisPluginFactory<PLUGIN_CLASS*> factory = nullptr;
331
332     if (factory == nullptr)
333     {
334         factory = new IrisPluginFactory<PLUGIN_CLASS*>(functions, plugin_name);
335         return E_ok;
336     }
337     else
338     {
339         return E_plugin_already_loaded;
340     }
341 }
342 };
343
344 #define IRIS_PLUGIN_FACTORY(PluginClassName)
345 extern "C" IRIS_EXPORT int64_t irisInitPlugin(IrisC_Funcions* functions)
346 {
347     return ::iris::IrisPluginFactory<PluginClassName*>::initPlugin(functions, #PluginClassName);
348 }
349
350 // --- Non-factory plugin support. ---
351 // Non-factory plugins are plugins which instantiate themselves directly in the entry point function.
352 // There is no factory instance. The singleton instance is the plugin rather than used to instantiate
353 // the plugins.
354 // They cannot receive partameters and cannot be instantiated multiple times.
355 // These are usually very simple singleton plugins.
356
357 template<class PLUGIN_CLASS>
358 class IrisNonFactoryPlugin
359 {
360 public:
361     IrisNonFactoryPlugin(IrisC_Funcions* functions, const std::string& pluginName)
362         : connectionInterface(functions)
363         , instantiationContext(&connectionInterface, instantiationResult,
364             std::vector<iris::ResourceInfo>(), std::vector<iris::InstantiationParameterValue>(), "client.plugin",
365             pluginName, iris::IrisInstance::DEFAULT_FLAGS | iris::IrisInstance::UNIQUEIFY)
366         , plugin(instantiationContext)
367     {
368     }
369
370     // Implementation of the plugin entry point.
371     // This will instantiate a new plugin.
372     static int64_t initPlugin(IrisC_Funcions* functions, const std::string& pluginName)
373     {
374         new IrisNonFactoryPlugin<PLUGIN_CLASS>(functions, pluginName);
375         return E_ok;
376     }
377
378 private:
379     iris::IrisCConnection connectionInterface;
380     iris::IrisInstantiationContext instantiationContext;
381     PLUGIN_CLASS plugin;
382     iris::InstantiationResult instantiationResult;
383 };
384
385 #define IRIS_NON_FACTORY_PLUGIN(PluginClassName)
386 extern "C" IRIS_EXPORT int64_t irisInitPlugin(IrisC_Funcions* functions)

```

```

409 {
410     return ::iris::IrisNonFactoryPlugin<PluginClassName>::initPlugin(functions, #PluginClassName);
411 }
412
413 NAMESPACE_IRIS_END
414
415 #endif // ARM_INCLUDE_IrisPluginFactory_h

```

9.55 IrisRegisterEventEmitter.h File Reference

Utility classes for emitting register read and register update events.

```
#include "iris/detail/IrisCommon.h"
```

```
#include "iris/detail/IrisRegisterEventEmitterBase.h"
```

Classes

- class [iris::IrisRegisterReadEventEmitter< REG_T, ARGS >](#)
An EventEmitter class for register read events.
- class [iris::IrisRegisterUpdateEventEmitter< REG_T, ARGS >](#)
An EventEmitter class for register update events.

9.55.1 Detailed Description

Utility classes for emitting register read and register update events.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

9.56 IrisRegisterEventEmitter.h

[Go to the documentation of this file.](#)

```

1
2
3
4
5
6
7
8 #ifndef ARM_INCLUDE_IrisRegisterEventEmitter_h
9 #define ARM_INCLUDE_IrisRegisterEventEmitter_h
10
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisRegisterEventEmitterBase.h"
13
14 NAMESPACE_IRIS_START
15
16
17
18
19
20
21
22
23
24
25
26
27 template <typename REG_T, typename... ARGS>
28 class IrisRegisterReadEventEmitter : public IrisRegisterEventEmitterBase
29 {
30 public:
31     IrisRegisterReadEventEmitter()
32         : IrisRegisterEventEmitterBase(sizeof...(ARGS) + 3)
33     {
34     }
35
36
37
38 void operator()(ResourceId rscId, bool debug, REG_T value, ARGS... args)
39 {
40     // Emit event
41     emitEvent(rscId, debug, value, args...);
42
43     // Check if this event indicates a breakpoint was hit
44     if (!debug)
45     {
46         checkBreakpointHit(rscId, value, /*is_read=*/true);
47     }
48 }
49 };
50
51
52
53
54
55
56
57 template <typename REG_T, typename... ARGS>
58 class IrisRegisterUpdateEventEmitter : public IrisRegisterEventEmitterBase
59 {
60 public:
61     IrisRegisterUpdateEventEmitter()
62         : IrisRegisterEventEmitterBase(sizeof...(ARGS) + 4)
63     {
64     }
65 }
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133

```

```

134
144 void operator() (ResourceId rscId, bool debug, REG_T old_value, REG_T new_value, ARGS... args)
145 {
146     // Emit event
147     emitEvent(rscId, debug, old_value, new_value, args...);
148
149     // Check if this event indicates a breakpoint was hit
150     if (!debug)
151     {
152         checkBreakpointHit(rscId, new_value, /*is_read=*/false);
153     }
154 }
155 };
156
157 NAMESPACE_IRIS_END
158
159 #endif // ARM_INCLUDE_IrisRegisterEventEmitter_h

```

9.57 IrisTcpClient.h File Reference

IrisTcpClient Type alias for IrisClient.

```
#include "iris/IrisClient.h"
```

Typedefs

- using **iris::IrisTcpClient** = IrisClient
Alias for backward compatibility.

9.57.1 Detailed Description

IrisTcpClient Type alias for IrisClient.

Date

Copyright ARM Limited 2022 All Rights Reserved.

9.58 IrisTcpClient.h

[Go to the documentation of this file.](#)

```

1
7 #ifndef ARM_INCLUDE_IrisTcpClient_h
8 #define ARM_INCLUDE_IrisTcpClient_h
9
10 #include "iris/IrisClient.h"
11
12 NAMESPACE_IRIS_START
13
15 using IrisTcpClient = IrisClient;
16
17 NAMESPACE_IRIS_END
18
19 #endif // #ifndef ARM_INCLUDE_IrisTcpClient_h

```