Contents

1 IrisSupportLib Reference Manual 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 .................. 16
      7.2.1 Detailed Description ............................... 16
      7.2.2 Function Documentation ............................ 17
         7.2.2.1 addNoValueRegister() .......................... 17
         7.2.2.2 addParameter() ................................ 18
         7.2.2.3 addRegister() .................................. 18
         7.2.2.4 addStringParameter() .......................... 19
7.2.2.5 addStringRegister() ........................................ 20
7.2.2.6 beginResourceGroup() ..................................... 20
7.2.2.7 enhanceParameter() .......................................... 21
7.2.2.8 enhanceRegister() ........................................... 21
7.2.2.9 getResourceInfo() ............................................. 22
7.2.2.10 setDefaultResourceDelegates() ......................... 22
7.2.2.11 setDefaultResourceReadDelegate() [1/3] ........... 23
7.2.2.12 setDefaultResourceReadDelegate() [2/3] ........... 24
7.2.2.13 setDefaultResourceReadDelegate() [3/3] ........... 24
7.2.2.14 setDefaultResourceWriteDelegate() [1/3] ......... 25
7.2.2.15 setDefaultResourceWriteDelegate() [2/3] ......... 25
7.2.2.16 setDefaultResourceWriteDelegate() [3/3] ......... 26
7.2.2.17 setNextSubRsId() ........................................... 27
7.2.2.18 setPropertyCanonicalRnScheme() ..................... 27
7.2.2.19 setTag() ...................................................... 28

7.3 IrisInstanceBuilder event APIs .................................. 29
7.3.1 Detailed Description ........................................... 29
7.3.2 Function Documentation ........................................ 29
7.3.2.1 addEventSource() [1/2] .................................... 30
7.3.2.2 addEventSource() [2/2] .................................... 30
7.3.2.3 finalizeRegisterReadEvent() ............................. 31
7.3.2.4 finalizeRegisterUpdateEvent() ......................... 31
7.3.2.5 getIrisInstanceEvent() ..................................... 31
7.3.2.6 setDefaultEsCreateDelegate() [1/3] ................. 31
7.3.2.7 setDefaultEsCreateDelegate() [2/3] ................. 32
7.3.2.8 setDefaultEsCreateDelegate() [3/3] ................. 32
7.3.2.9 setRegisterReadEvent() [1/2] ............................. 33
7.3.2.10 setRegisterReadEvent() [2/2] ........................... 34
7.3.2.11 setRegisterUpdateEvent() [1/2] ....................... 34
7.3.2.12 setRegisterUpdateEvent() [2/2] ....................... 35
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4</td>
<td>IrisInstanceBuilder breakpoint APIs</td>
<td>37</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Detailed Description</td>
<td>37</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Function Documentation</td>
<td>37</td>
</tr>
<tr>
<td>7.4.2.1</td>
<td>getBreakpointInfo()</td>
<td>37</td>
</tr>
<tr>
<td>7.4.2.2</td>
<td>notifyBreakpointHit()</td>
<td>38</td>
</tr>
<tr>
<td>7.4.2.3</td>
<td>notifyBreakpointHitData()</td>
<td>38</td>
</tr>
<tr>
<td>7.4.2.4</td>
<td>notifyBreakpointHitRegister()</td>
<td>39</td>
</tr>
<tr>
<td>7.4.2.5</td>
<td>setBreakpointDeleteDelegate() [1/3]</td>
<td>39</td>
</tr>
<tr>
<td>7.4.2.6</td>
<td>setBreakpointDeleteDelegate() [2/3]</td>
<td>40</td>
</tr>
<tr>
<td>7.4.2.7</td>
<td>setBreakpointDeleteDelegate() [3/3]</td>
<td>40</td>
</tr>
<tr>
<td>7.4.2.8</td>
<td>setBreakpointSetDelegate() [1/3]</td>
<td>41</td>
</tr>
<tr>
<td>7.4.2.9</td>
<td>setBreakpointSetDelegate() [2/3]</td>
<td>41</td>
</tr>
<tr>
<td>7.4.2.10</td>
<td>setBreakpointSetDelegate() [3/3]</td>
<td>42</td>
</tr>
<tr>
<td>7.5</td>
<td>IrisInstanceBuilder memory APIs</td>
<td>43</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Detailed Description</td>
<td>44</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Function Documentation</td>
<td>44</td>
</tr>
<tr>
<td>7.5.2.1</td>
<td>addAddressTranslation()</td>
<td>44</td>
</tr>
<tr>
<td>7.5.2.2</td>
<td>addMemorySpace()</td>
<td>44</td>
</tr>
<tr>
<td>7.5.2.3</td>
<td>setDefaultAddressTranslateDelegate() [1/3]</td>
<td>45</td>
</tr>
<tr>
<td>7.5.2.4</td>
<td>setDefaultAddressTranslateDelegate() [2/3]</td>
<td>46</td>
</tr>
<tr>
<td>7.5.2.5</td>
<td>setDefaultAddressTranslateDelegate() [3/3]</td>
<td>46</td>
</tr>
<tr>
<td>7.5.2.6</td>
<td>setDefaultGetMemorySidebandInfoDelegate() [1/3]</td>
<td>47</td>
</tr>
<tr>
<td>7.5.2.7</td>
<td>setDefaultGetMemorySidebandInfoDelegate() [2/3]</td>
<td>48</td>
</tr>
<tr>
<td>7.5.2.8</td>
<td>setDefaultGetMemorySidebandInfoDelegate() [3/3]</td>
<td>48</td>
</tr>
<tr>
<td>7.5.2.9</td>
<td>setDefaultMemoryReadDelegate() [1/3]</td>
<td>49</td>
</tr>
<tr>
<td>7.5.2.10</td>
<td>setDefaultMemoryReadDelegate() [2/3]</td>
<td>50</td>
</tr>
<tr>
<td>7.5.2.11</td>
<td>setDefaultMemoryReadDelegate() [3/3]</td>
<td>51</td>
</tr>
<tr>
<td>7.5.2.12</td>
<td>setDefaultMemoryWriteDelegate() [1/3]</td>
<td>51</td>
</tr>
<tr>
<td>7.5.2.13</td>
<td>setDefaultMemoryWriteDelegate() [2/3]</td>
<td>52</td>
</tr>
<tr>
<td>7.5.2.14</td>
<td>setDefaultMemoryWriteDelegate() [3/3]</td>
<td>53</td>
</tr>
</tbody>
</table>
7.5.2.15 setPropertyCanonicalMsnScheme() ....................................................... 53

7.6 IrisInstanceBuilder image loading APIs ......................................................... 55
  7.6.1 Detailed Description ....................................................................................... 55
  7.6.2 Function Documentation ................................................................................ 55
    7.6.2.1 setLoadImageDataDelegate() ................................................................. 55
    7.6.2.2 setLoadImageDataDelegate() ................................................................. 56
    7.6.2.3 setLoadImageDataDelegate() ................................................................. 56
    7.6.2.4 setLoadImageFileDelegate() ................................................................. 57
    7.6.2.5 setLoadImageFileDelegate() ................................................................. 57
    7.6.2.6 setLoadImageFileDelegate() ................................................................. 58

7.7 IrisInstanceBuilder image readData callback APIs ............................................. 59
  7.7.1 Detailed Description ....................................................................................... 59
  7.7.2 Function Documentation ................................................................................ 59
    7.7.2.1 openImage() .......................................................................................... 59

7.8 IrisInstanceBuilder execution stepping APIs ....................................................... 60
  7.8.1 Detailed Description ....................................................................................... 60
  7.8.2 Function Documentation ................................................................................ 60
    7.8.2.1 setRemainingStepGetDelegate() ............................................................. 61
    7.8.2.2 setRemainingStepGetDelegate() ............................................................. 61
    7.8.2.3 setRemainingStepGetDelegate() ............................................................. 62
    7.8.2.4 setRemainingStepSetDelegate() ............................................................. 62
    7.8.2.5 setRemainingStepSetDelegate() ............................................................. 63
    7.8.2.6 setRemainingStepSetDelegate() ............................................................. 63
    7.8.2.7 setStepCountGetDelegate() ................................................................. 64
    7.8.2.8 setStepCountGetDelegate() ................................................................. 64
    7.8.2.9 setStepCountGetDelegate() ................................................................. 65

7.9 Disassembler delegate functions ....................................................................... 66
  7.9.1 Detailed Description ....................................................................................... 66
  7.9.2 Typedef Documentation .................................................................................. 66
    7.9.2.1 DisassembleOpcodeDelegate ................................................................. 67
    7.9.2.2 GetCurrentDisassemblyModeDelegate ..................................................... 67
    7.9.2.3 GetDisassemblyDelegate ....................................................................... 67
  7.9.3 Function Documentation ................................................................................ 67
    7.9.3.1 addDisassemblyMode() ......................................................................... 67
    7.9.3.2 attachTo() .......................................................................................... 68
    7.9.3.3 IrisInstanceDisassembler() ................................................................... 68
    7.9.3.4 setDisassembleOpcodeDelegate() ......................................................... 68
    7.9.3.5 setGetCurrentModeDelegate() .............................................................. 69
    7.9.3.6 setGetDisassemblyDelegate() .............................................................. 69

7.10 Semihosting data request flag constants ......................................................... 70
  7.10.1 Detailed Description ..................................................................................... 70
  7.10.2 Variable Documentation .............................................................................. 70
    7.10.2.1 BLOCK .............................................................................................. 70
# Class Documentation

## 8.1 iris::IrisInstanceBuilder::AddressTranslationBuilder Class Reference

### 8.1.1 Detailed Description

### 8.1.2 Member Function Documentation

1. **setTranslateDelegate()**
   - 1/3
   - 71
2. **setTranslateDelegate()**
   - 2/3
   - 72
3. **setTranslateDelegate()**
   - 3/3
   - 72

## 8.2 iris::IrisInstanceMemory::AddressTranslationInfoAndAccess Struct Reference

### 8.2.1 Detailed Description

## 8.3 iris::IrisInstanceBuilder::EventSourceBuilder Class Reference

### 8.3.1 Detailed Description

### 8.3.2 Member Function Documentation

1. **addEnumElement()**
   - 74
2. **addField()**
   - 75
3. **setCounter()**
   - 75
4. **setDescription()**
   - 76
5. **setEventStreamCreateDelegate()**
   - 1/2
   - 76
6. **setEventStreamCreateDelegate()**
   - 2/2
   - 76
7. **setFormat()**
   - 77
8. **setHidden()**
   - 77
9. **setName()**
   - 78
10. **setRingBuffer()**
    - 78

## 8.4 iris::IrisInstanceEvent::EventSourceInfoAndDelegate Struct Reference

### 8.4.1 Detailed Description

## 8.5 iris::EventStream Class Reference

### 8.5.1 Detailed Description

### 8.5.2 Member Function Documentation

1. **addField()**
   - 1/4
   - 81
2. **addField()**
   - 2/4
   - 81
3. **addField()**
   - 3/4
   - 83
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5.2.4 addField()</td>
<td>83</td>
</tr>
<tr>
<td>8.5.2.5 addFieldSlow()</td>
<td>83</td>
</tr>
<tr>
<td>8.5.2.6 addFieldSlow()</td>
<td>84</td>
</tr>
<tr>
<td>8.5.2.7 addFieldSlow()</td>
<td>84</td>
</tr>
<tr>
<td>8.5.2.8 addFieldSlow()</td>
<td>85</td>
</tr>
<tr>
<td>8.5.2.9 checkRangePc()</td>
<td>85</td>
</tr>
<tr>
<td>8.5.2.10 disable()</td>
<td>85</td>
</tr>
<tr>
<td>8.5.2.11 emitEventBegin()</td>
<td>86</td>
</tr>
<tr>
<td>8.5.2.12 emitEventBegin()</td>
<td>86</td>
</tr>
<tr>
<td>8.5.2.13 emitEventEnd()</td>
<td>86</td>
</tr>
<tr>
<td>8.5.2.14 enable()</td>
<td>87</td>
</tr>
<tr>
<td>8.5.2.15 getCountVal()</td>
<td>87</td>
</tr>
<tr>
<td>8.5.2.16 getEcInstId()</td>
<td>87</td>
</tr>
<tr>
<td>8.5.2.17 getState()</td>
<td>87</td>
</tr>
<tr>
<td>8.5.2.18 isCounter()</td>
<td>88</td>
</tr>
<tr>
<td>8.5.2.19 isEnabled()</td>
<td>88</td>
</tr>
<tr>
<td>8.5.2.20 selfRelease()</td>
<td>88</td>
</tr>
<tr>
<td>8.5.2.21 setCounter()</td>
<td>88</td>
</tr>
<tr>
<td>8.5.2.22 setProperties()</td>
<td>89</td>
</tr>
<tr>
<td>8.5.2.23 setRanges()</td>
<td>89</td>
</tr>
<tr>
<td>8.5.3.1 counter</td>
<td>90</td>
</tr>
<tr>
<td>8.5.3.2 irisInstance</td>
<td>90</td>
</tr>
<tr>
<td>8.6.1 Detailed Description</td>
<td>92</td>
</tr>
<tr>
<td>8.6.2 Member Function Documentation</td>
<td>92</td>
</tr>
<tr>
<td>8.6.2.1 addEnum()</td>
<td>92</td>
</tr>
<tr>
<td>8.6.2.2 addField()</td>
<td>93</td>
</tr>
<tr>
<td>8.6.2.3 addLogicalField()</td>
<td>93</td>
</tr>
<tr>
<td>8.6.2.4 addStringEnum()</td>
<td>93</td>
</tr>
<tr>
<td>Section</td>
<td>Function</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>8.6.2.5</td>
<td>getRscId()</td>
</tr>
<tr>
<td>8.6.2.6</td>
<td>getRscId()</td>
</tr>
<tr>
<td>8.6.2.7</td>
<td>parent()</td>
</tr>
<tr>
<td>8.6.2.8</td>
<td>setAddressOffset()</td>
</tr>
<tr>
<td>8.6.2.9</td>
<td>setBitWidth()</td>
</tr>
<tr>
<td>8.6.2.10</td>
<td>setCanonicalRn()</td>
</tr>
<tr>
<td>8.6.2.11</td>
<td>setCanonicalRnElfDwarf()</td>
</tr>
<tr>
<td>8.6.2.12</td>
<td>setCname()</td>
</tr>
<tr>
<td>8.6.2.13</td>
<td>setDescr()</td>
</tr>
<tr>
<td>8.6.2.14</td>
<td>setFormat()</td>
</tr>
<tr>
<td>8.6.2.15</td>
<td>setLsbOffset()</td>
</tr>
<tr>
<td>8.6.2.16</td>
<td>setName()</td>
</tr>
<tr>
<td>8.6.2.17</td>
<td>setParentRscId()</td>
</tr>
<tr>
<td>8.6.2.18</td>
<td>setReadDelegate()</td>
</tr>
<tr>
<td>8.6.2.19</td>
<td>setReadDelegate()</td>
</tr>
<tr>
<td>8.6.2.20</td>
<td>setReadDelegate()</td>
</tr>
<tr>
<td>8.6.2.21</td>
<td>setResetData()</td>
</tr>
<tr>
<td>8.6.2.22</td>
<td>setResetData()</td>
</tr>
<tr>
<td>8.6.2.23</td>
<td>setResetDataFromContainer()</td>
</tr>
<tr>
<td>8.6.2.24</td>
<td>setResetString()</td>
</tr>
<tr>
<td>8.6.2.25</td>
<td>setRwModel()</td>
</tr>
<tr>
<td>8.6.2.26</td>
<td>setSubRscId()</td>
</tr>
<tr>
<td>8.6.2.27</td>
<td>setTag()</td>
</tr>
<tr>
<td>8.6.2.28</td>
<td>setTag()</td>
</tr>
<tr>
<td>8.6.2.29</td>
<td>setType()</td>
</tr>
<tr>
<td>8.6.2.30</td>
<td>setWriteDelegate()</td>
</tr>
<tr>
<td>8.6.2.31</td>
<td>setWriteDelegate()</td>
</tr>
<tr>
<td>8.6.2.32</td>
<td>setWriteDelegate()</td>
</tr>
<tr>
<td>8.6.2.33</td>
<td>setWriteMask()</td>
</tr>
<tr>
<td>8.6.2.34</td>
<td>setWriteMask()</td>
</tr>
</tbody>
</table>
8.6.2.35 setWriteMaskFromContainer() .......................... 108

8.7 iris::IrisCConnection Class Reference ....................... 109
8.7.1 Detailed Description ........................................ 110

8.8 iris::IrisEventEmitter<ARGS> Class Template Reference .... 110
8.8.1 Detailed Description ....................................... 110
8.8.2 Member Function Documentation ........................... 111
8.8.2.1 operator()() ........................................... 111

8.9 iris::IrisEventRegistry Class Reference ..................... 111
8.9.1 Detailed Description ....................................... 112
8.9.2 Member Function Documentation ........................... 112
8.9.2.1 addField() ........................................... 112
8.9.2.2 addFieldSlow() ....................................... 112
8.9.2.3 begin() .............................................. 113
8.9.2.4 emitEventEnd() ....................................... 113
8.9.2.5 empty() .............................................. 113
8.9.2.6 end() ................................................ 114
8.9.2.7 registerEventStream() ................................ 114
8.9.2.8 unregisterEventStream() ................................ 114

8.10 iris::IrisEventStream Class Reference ....................... 115
8.10.1 Detailed Description ....................................... 115
8.10.2 Member Function Documentation ........................... 115
8.10.2.1 disable() ........................................... 115
8.10.2.2 enable() ............................................ 116

8.11 iris::IrisGlobalInstance Class Reference .................. 116
8.11.1 Member Function Documentation ........................... 116
8.11.1.1 registerChannel() ................................ 117
8.11.1.2 registerIrisInterfaceChannel() ..................... 117
8.11.1.3 unregisterIrisInterfaceChannel() .................... 117

8.12 iris::IrisInstance Class Reference ........................ 117
8.12.1 Constructor & Destructor Documentation .................. 119
8.12.1.1 IrisInstance() [1/2] ........................................ 119
8.12.1.2 IrisInstance() [2/2] ........................................ 119

8.12.2 Member Function Documentation .................................. 120
8.12.2.1 getBuilder() ........................................ 120
8.12.2.2 getInstanceName() ...................................... 120
8.12.2.3 getInstId() ........................................ 120
8.12.2.4 getLocalIrisInterface() .................................. 121
8.12.2.5 getPropertyMap() ....................................... 121
8.12.2.6 getRemoteIrisInterface() .................................. 121
8.12.2.7 irisCall() ........................................ 121
8.12.2.8 irisCallNoThrow() ....................................... 122
8.12.2.9 irisCallThrow() .......................................... 122
8.12.2.10 registerEventCallback() [1/3] ......................... 122
8.12.2.11 registerEventCallback() [2/3] ......................... 123
8.12.2.13 registerFunction() ...................................... 123
8.12.2.14 registerInstance() ..................................... 124
8.12.2.15 sendRequest() .......................................... 124
8.12.2.16 sendResponse() ......................................... 125
8.12.2.17 setConnectionInterface() ................................ 125
8.12.2.18 setProperty() .......................................... 125
8.12.2.19 setThrowOnError() ...................................... 127
8.12.2.20 unregisterInstance() ................................... 127

8.13 iris::IrisInstanceBreakpoint Class Reference ......................... 127
8.13.1 Detailed Description ............................................ 128
8.13.2 Member Function Documentation .................................. 128
8.13.2.1 addCondition() .......................................... 128
8.13.2.2 attachTo() ........................................ 129
8.13.2.3 getBreakpointInfo() .................................... 129
8.13.2.4 notifyBreakpointHit() ................................... 129
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.14.3.21 setExecutionStateSetDelegate()</td>
<td>150</td>
</tr>
<tr>
<td>8.14.3.22 setGetCurrentDisassemblyModeDelegate()</td>
<td>150</td>
</tr>
<tr>
<td>8.15 iris::IrisInstanceDebuggableState Class Reference</td>
<td>151</td>
</tr>
<tr>
<td>8.15.1 Detailed Description</td>
<td>151</td>
</tr>
<tr>
<td>8.15.2 Member Function Documentation</td>
<td>151</td>
</tr>
<tr>
<td>8.15.2.1 attachTo()</td>
<td>151</td>
</tr>
<tr>
<td>8.15.2.2 setGetAcknowledgeDelegate()</td>
<td>152</td>
</tr>
<tr>
<td>8.15.2.3 setSetRequestDelegate()</td>
<td>152</td>
</tr>
<tr>
<td>8.16 iris::IrisInstanceDisassembler Class Reference</td>
<td>152</td>
</tr>
<tr>
<td>8.16.1 Detailed Description</td>
<td>153</td>
</tr>
<tr>
<td>8.17 iris::IrisInstanceEvent Class Reference</td>
<td>153</td>
</tr>
<tr>
<td>8.17.1 Detailed Description</td>
<td>154</td>
</tr>
<tr>
<td>8.17.2 Constructor &amp; Destructor Documentation</td>
<td>154</td>
</tr>
<tr>
<td>8.17.2.1 IrisInstanceEvent()</td>
<td>154</td>
</tr>
<tr>
<td>8.17.3 Member Function Documentation</td>
<td>154</td>
</tr>
<tr>
<td>8.17.3.1 addEventSource()</td>
<td>154</td>
</tr>
<tr>
<td>8.17.3.2 addEventSource()</td>
<td>155</td>
</tr>
<tr>
<td>8.17.3.3 attachTo()</td>
<td>155</td>
</tr>
<tr>
<td>8.17.3.4 setDefaultEsCreateDelegate()</td>
<td>155</td>
</tr>
<tr>
<td>8.18 iris::IrisInstanceFactoryBuilder Class Reference</td>
<td>156</td>
</tr>
<tr>
<td>8.18.1 Detailed Description</td>
<td>156</td>
</tr>
<tr>
<td>8.18.2 Constructor &amp; Destructor Documentation</td>
<td>156</td>
</tr>
<tr>
<td>8.18.2.1 IrisInstanceFactoryBuilder()</td>
<td>156</td>
</tr>
<tr>
<td>8.18.3 Member Function Documentation</td>
<td>157</td>
</tr>
<tr>
<td>8.18.3.1 addBooleanParameter()</td>
<td>157</td>
</tr>
<tr>
<td>8.18.3.2 addHiddenBooleanParameter()</td>
<td>157</td>
</tr>
<tr>
<td>8.18.3.3 addHiddenStringParameter()</td>
<td>158</td>
</tr>
<tr>
<td>8.18.3.4 addHidenParameter()</td>
<td>158</td>
</tr>
<tr>
<td>8.18.3.5 addParameter()</td>
<td>159</td>
</tr>
<tr>
<td>8.18.3.6 addStringParameter()</td>
<td>159</td>
</tr>
</tbody>
</table>
8.25.3 Member Function Documentation ........................................ 178
  8.25.3.1 attachTo() ......................................................... 178
  8.25.3.2 enterPostInstantiationPhase() .................................. 179
  8.25.3.3 notifySimPhase() ................................................ 179
  8.25.3.4 setConnectionInterface() ........................................ 179
  8.25.3.5 setEventHandler() ............................................... 179
  8.25.3.6 setGetParameterInfoDelegate() [1/3] .............................. 180
  8.25.3.7 setGetParameterInfoDelegate() [2/3] .............................. 180
  8.25.3.8 setGetParameterInfoDelegate() [3/3] .............................. 181
  8.25.3.9 setInstantiateDelegate() [1/3] ................................ 181
  8.25.3.10 setInstantiateDelegate() [2/3] ................................ 181
  8.25.3.11 setInstantiateDelegate() [3/3] ................................ 182
  8.25.3.12 setRequestShutdownDelegate() [1/3] ............................... 182
  8.25.3.13 setRequestShutdownDelegate() [2/3] ............................... 182
  8.25.3.14 setRequestShutdownDelegate() [3/3] ............................... 183
  8.25.3.15 setResetDelegate() [1/3] ...................................... 183
  8.25.3.16 setResetDelegate() [2/3] ...................................... 183
  8.25.3.17 setResetDelegate() [3/3] ...................................... 184
  8.25.3.18 setSetParameterValueDelegate() [1/3] ............................ 184
  8.25.3.19 setSetParameterValueDelegate() [2/3] ............................ 184
  8.25.3.20 setSetParameterValueDelegate() [3/3] ............................ 185

8.26 iris::IrisInstanceSimulationTime Class Reference .......................... 185
  8.26.1 Detailed Description .................................................. 186
  8.26.2 Constructor & Destructor Documentation ............................... 186
    8.26.2.1 IrisInstanceSimulationTime() ................................ 186
  8.26.3 Member Function Documentation ....................................... 186
    8.26.3.1 attachTo() ...................................................... 186
8.26.3.5 setSimTimeGetDelegate() [3/3] ................................................. 188
8.26.3.6 setSimTimeRunDelegate() [1/3] .................................................. 188
8.26.3.7 setSimTimeRunDelegate() [2/3] .................................................. 188
8.26.3.8 setSimTimeRunDelegate() [3/3] .................................................. 189
8.26.3.9 setSimTimeStopDelegate() [1/3] .................................................. 189
8.26.3.10 setSimTimeStopDelegate() [2/3] ............................................... 189
8.26.3.11 setSimTimeStopDelegate() [3/3] ............................................... 190

8.27 iris::IrisInstanceStep Class Reference .............................................. 190
8.27.1 Detailed Description ................................................................. 190
8.27.2 Constructor & Destructor Documentation ........................................ 191
  8.27.2.1 IrisInstanceStep() ............................................................... 191
8.27.3 Member Function Documentation ................................................ 191
  8.27.3.1 attachTo() ................................................................. 191
  8.27.3.2 setRemainingStepGetDelegate() .............................. 191
  8.27.3.3 setRemainingStepSetDelegate() .............................. 192
  8.27.3.4 setStepCountGetDelegate() ........................................ 192

8.28 iris::IrisInstanceTable Class Reference ......................................... 192
8.28.1 Detailed Description ................................................................. 193
8.28.2 Constructor & Destructor Documentation ........................................ 193
  8.28.2.1 IrisInstanceTable() ............................................................... 193
8.28.3 Member Function Documentation ................................................ 193
  8.28.3.1 addTableInfo() ................................................................. 193
  8.28.3.2 attachTo() ................................................................. 193
  8.28.3.3 setDefaultReadDelegate() ........................................ 194
  8.28.3.4 setDefaultWriteDelegate() ........................................ 194

8.29 iris::IrisInstantiationContext Class Reference ................................ 194
8.29.1 Detailed Description ................................................................. 195
8.29.2 Member Function Documentation ................................................ 195
  8.29.2.1 error() ................................................................. 195
  8.29.2.2 getConnectionInterface() ........................................ 196
8.29.2.3 getInstanceName() .............................................. 196
8.29.2.4 getParameter() [1/2] ............................................. 196
8.29.2.5 getParameter() [2/2] ............................................. 197
8.29.2.6 getRecommendedInstanceFlags() .......................... 197
8.29.2.7 getSubcomponentContext() .................................... 197
8.29.2.8 parameterError() ................................................ 198
8.29.2.9 parameterWarning() ............................................. 198
8.29.2.10 warning() ......................................................... 199

8.30 iris::IrisParameterBuilder Class Reference ............................ 199
  8.30.1 Detailed Description .............................................. 201
  8.30.2 Constructor & Destructor Documentation ............................ 201
    8.30.2.1 IrisParameterBuilder() ..................................... 201
  8.30.3 Member Function Documentation .................................... 201
    8.30.3.1 addEnum() .................................................. 201
    8.30.3.2 addStringEnum() .......................................... 202
    8.30.3.3 setBitWidth() .............................................. 202
    8.30.3.4 setDefault() [1/3] ......................................... 203
    8.30.3.5 setDefault() [2/3] ......................................... 203
    8.30.3.6 setDefault() [3/3] ......................................... 203
    8.30.3.7 setDefaultFloat() ........................................ 204
    8.30.3.8 setDefaultSigned() [1/2] .................................. 204
    8.30.3.9 setDefaultSigned() [2/2] .................................. 204
    8.30.3.10 setDescr() ............................................... 205
    8.30.3.11 setFormat() ............................................... 205
    8.30.3.12 setInitOnly() ............................................. 206
    8.30.3.13 setMax() [1/2] ............................................. 206
    8.30.3.14 setMax() [2/2] ............................................. 206
    8.30.3.15 setMaxFloat() ........................................... 207
    8.30.3.16 setMaxSigned() [1/2] .................................... 207
    8.30.3.17 setMaxSigned() [2/2] .................................... 207
8.30.3.18 setMin() [1/2] ........................................ 209
8.30.3.19 setMin() [2/2] ........................................ 209
8.30.3.20 setMinFloat() ........................................ 210
8.30.3.21 setMinSigned() [1/2] .................................. 210
8.30.3.22 setMinSigned() [2/2] .................................. 210
8.30.3.23 setName() ........................................ 211
8.30.3.24 setRange() [1/2] ....................................... 211
8.30.3.25 setRange() [2/2] ....................................... 212
8.30.3.26 setRangeFloat() ...................................... 212
8.30.3.27 setRangeSigned() [1/2] ................................ 212
8.30.3.28 setRangeSigned() [2/2] ................................ 213
8.30.3.29 setRwMode() ........................................ 213
8.30.3.30 setSubRscId() ........................................ 214
8.30.3.31 setTag() [1/2] ....................................... 214
8.30.3.32 setTag() [2/2] ....................................... 214
8.30.3.33 setTopology() ........................................ 215
8.30.3.34 setType() ........................................ 215

8.31 iris::IrisPluginFactory< PLUGIN_INSTANCE > Class Template Reference ........................................ 215

8.32 iris::IrisPluginFactoryBuilder Class Reference ................................................................. 216
8.32.1 Detailed Description ........................................ 216
8.32.2 Constructor & Destructor Documentation .............................................................. 216
8.32.2.1 IrisPluginFactoryBuilder() ................................ 216
8.32.3 Member Function Documentation ................................................................. 217
8.32.3.1 getDefaultInstanceName() ................................ 217
8.32.3.2 getInstanceNamePrefix() ................................ 217
8.32.3.3 getPluginName() ........................................ 217
8.32.3.4 setDefaultInstanceName() ................................ 217
8.32.3.5 setInstanceNamePrefix() ................................ 218
8.32.3.6 setPluginName() ........................................ 218

8.33 iris::IrisRegisterReadEventEmitter< REG_T, ARGS > Class Template Reference ........................................ 218
8.37.2.6 setEndianness() ...................................................... 228
8.37.2.7 setMaxAddr() .......................................................... 229
8.37.2.8 setMinAddr() .......................................................... 229
8.37.2.9 setName() .............................................................. 230
8.37.2.10 setReadDelegate() [1/3] ........................................... 230
8.37.2.11 setReadDelegate() [2/3] ........................................... 230
8.37.2.12 setReadDelegate() [3/3] ........................................... 231
8.37.2.13 setSidebandDelegate() [1/3] ..................................... 231
8.37.2.14 setSidebandDelegate() [2/3] ..................................... 232
8.37.2.15 setSidebandDelegate() [3/3] ..................................... 232
8.37.2.16 setWriteDelegate() [1/3] ......................................... 233
8.37.2.17 setWriteDelegate() [2/3] ......................................... 233
8.37.2.18 setWriteDelegate() [3/3] ......................................... 234

8.38 iris::IrisInstanceBuilder::ParameterBuilder Class Reference .............................................. 235

8.38.1 Detailed Description .................................................. 236
8.38.2 Member Function Documentation ................................... 236
8.38.2.1 addEnum() ........................................................... 237
8.38.2.2 addStringEnum() .................................................... 237
8.38.2.3 getRscId() [1/2] ....................................................... 238
8.38.2.4 getRscId() [2/2] ....................................................... 238
8.38.2.5 setBitWidth() ........................................................ 238
8.38.2.6 setCname() .......................................................... 239
8.38.2.7 setDefaultData() [1/2] ............................................ 239
8.38.2.8 setDefaultData() [2/2] ............................................ 240
8.38.2.9 setDefaultDataFromContainer() ............................... 240
8.38.2.10 setDefaultString() ............................................... 242
8.38.2.11 setDescr() .......................................................... 242
8.38.2.12 setFormat() ........................................................ 243
8.38.2.13 setInitOnly() ....................................................... 243
8.38.2.14 setMax() [1/2] ....................................................... 243

Generated by Doxygen
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.38.2.15</td>
<td>setMax()</td>
</tr>
<tr>
<td>8.38.2.16</td>
<td>setMaxFromContainer()</td>
</tr>
<tr>
<td>8.38.2.17</td>
<td>setMin()</td>
</tr>
<tr>
<td>8.38.2.18</td>
<td>setMin()</td>
</tr>
<tr>
<td>8.38.2.19</td>
<td>setMinFromContainer()</td>
</tr>
<tr>
<td>8.38.2.20</td>
<td>setName()</td>
</tr>
<tr>
<td>8.38.2.21</td>
<td>setParentRscId()</td>
</tr>
<tr>
<td>8.38.2.22</td>
<td>setReadDelegate()</td>
</tr>
<tr>
<td>8.38.2.23</td>
<td>setReadDelegate()</td>
</tr>
<tr>
<td>8.38.2.24</td>
<td>setReadDelegate()</td>
</tr>
<tr>
<td>8.38.2.25</td>
<td>setRwMode()</td>
</tr>
<tr>
<td>8.38.2.26</td>
<td>setSubRscId()</td>
</tr>
<tr>
<td>8.38.2.27</td>
<td>setTag()</td>
</tr>
<tr>
<td>8.38.2.28</td>
<td>setTag()</td>
</tr>
<tr>
<td>8.38.2.29</td>
<td>setType()</td>
</tr>
<tr>
<td>8.38.2.30</td>
<td>setWriteDelegate()</td>
</tr>
<tr>
<td>8.38.2.31</td>
<td>setWriteDelegate()</td>
</tr>
<tr>
<td>8.38.2.32</td>
<td>setWriteDelegate()</td>
</tr>
</tbody>
</table>

8.39 iris::IrisInstanceBuilder::RegisterBuilder Class Reference

8.39.1 Detailed Description

8.39.2 Member Function Documentation

8.39.2.1 addEnum()

8.39.2.2 addField()

8.39.2.3 addLogicalField()

8.39.2.4 addStringEnum()

8.39.2.5 getRscId() |

8.39.2.6 getRscId() |

8.39.2.7 setAddressOffset() |

8.39.2.8 setBitWidth() |

8.39.2.9 setCanonicalRn() |

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>244</td>
</tr>
<tr>
<td>244</td>
</tr>
<tr>
<td>245</td>
</tr>
<tr>
<td>246</td>
</tr>
<tr>
<td>246</td>
</tr>
<tr>
<td>247</td>
</tr>
<tr>
<td>247</td>
</tr>
<tr>
<td>248</td>
</tr>
<tr>
<td>249</td>
</tr>
<tr>
<td>249</td>
</tr>
<tr>
<td>250</td>
</tr>
<tr>
<td>250</td>
</tr>
<tr>
<td>251</td>
</tr>
<tr>
<td>251</td>
</tr>
<tr>
<td>252</td>
</tr>
<tr>
<td>252</td>
</tr>
<tr>
<td>253</td>
</tr>
<tr>
<td>253</td>
</tr>
<tr>
<td>254</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>257</td>
</tr>
<tr>
<td>257</td>
</tr>
<tr>
<td>258</td>
</tr>
<tr>
<td>258</td>
</tr>
<tr>
<td>259</td>
</tr>
<tr>
<td>259</td>
</tr>
<tr>
<td>259</td>
</tr>
<tr>
<td>260</td>
</tr>
</tbody>
</table>
8.42 Detailed Description .................................................................................. 274
8.42.2 Member Function Documentation ....................................................... 275
  8.42.2.1 readData() ..................................................................................... 275
  8.42.2.2 semihostedCall() .......................................................................... 275
8.43 iris::IrisInstanceMemory::SpaceInfoAndAccess Struct Reference .......... 276
  8.43.1 Detailed Description .......................................................................... 276
8.44 iris::IrisInstanceBuilder::TableBuilder Class Reference ......................... 276
  8.44.1 Detailed Description .......................................................................... 277
  8.44.2 Member Function Documentation ....................................................... 277
    8.44.2.1 addColumn() .............................................................................. 278
    8.44.2.2 addColumnInfo() ......................................................................... 278
    8.44.2.3 setDescription() ....................................................................... 278
    8.44.2.4 setFormatLong() ....................................................................... 280
    8.44.2.5 setFormatShort() ....................................................................... 280
    8.44.2.6 setIndexFormatHint() .................................................................. 280
    8.44.2.7 setMaxIndex() ........................................................................... 282
    8.44.2.8 setMinIndex() ........................................................................... 282
    8.44.2.9 setName() .................................................................................. 282
    8.44.2.10 setReadDelegate() [1/3] ............................................................... 284
    8.44.2.11 setReadDelegate() [2/3] ............................................................... 284
    8.44.2.12 setReadDelegate() [3/3] ............................................................... 285
    8.44.2.13 setWriteDelegate() [1/3] .............................................................. 285
    8.44.2.14 setWriteDelegate() [2/3] .............................................................. 286
    8.44.2.15 setWriteDelegate() [3/3] .............................................................. 286
8.45 iris::IrisInstanceBuilder::TableColumnBuilder Class Reference ............. 287
  8.45.1 Detailed Description .......................................................................... 287
  8.45.2 Member Function Documentation ....................................................... 288
    8.45.2.1 addColumn() .............................................................................. 288
    8.45.2.2 addColumnInfo() ......................................................................... 288
    8.45.2.3 endColumn() ............................................................................. 289
    8.45.2.4 setBitWidth() ............................................................................ 289
    8.45.2.5 setDescription() ....................................................................... 289
    8.45.2.6 setFormat() ............................................................................... 290
    8.45.2.7 setFormatLong() ....................................................................... 290
    8.45.2.8 setFormatShort() ....................................................................... 290
    8.45.2.9 setName() .................................................................................. 291
    8.45.2.10 setRwMode() ............................................................................ 291
    8.45.2.11 setType() ................................................................................. 291
8.46 iris::IrisInstanceTable::TableInfoAndAccess Struct Reference ............... 292
  8.46.1 Detailed Description .......................................................................... 292
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.17</td>
<td>IrisInstanceSimulation.h File Reference</td>
</tr>
<tr>
<td>9.17.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>9.17.2</td>
<td>Typedef Documentation</td>
</tr>
<tr>
<td>9.17.2.1</td>
<td>SimulationGetParameterInfoDelegate</td>
</tr>
<tr>
<td>9.17.2.2</td>
<td>SimulationInstantiateDelegate</td>
</tr>
<tr>
<td>9.17.2.3</td>
<td>SimulationRequestShutdownDelegate</td>
</tr>
<tr>
<td>9.17.2.4</td>
<td>SimulationResetDelegate</td>
</tr>
<tr>
<td>9.17.2.5</td>
<td>SimulationSetParameterValueDelegate</td>
</tr>
<tr>
<td>9.18</td>
<td>IrisInstanceSimulationTime.h File Reference</td>
</tr>
<tr>
<td>9.18.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>9.18.2</td>
<td>Typedef Documentation</td>
</tr>
<tr>
<td>9.18.2.1</td>
<td>SimulationTimeGetDelegate</td>
</tr>
<tr>
<td>9.18.2.2</td>
<td>SimulationTimeRunDelegate</td>
</tr>
<tr>
<td>9.18.2.3</td>
<td>SimulationTimeStopDelegate</td>
</tr>
<tr>
<td>9.18.3</td>
<td>Enumeration Type Documentation</td>
</tr>
<tr>
<td>9.19</td>
<td>IrisInstanceStep.h File Reference</td>
</tr>
<tr>
<td>9.19.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>9.19.2</td>
<td>Typedef Documentation</td>
</tr>
<tr>
<td>9.19.2.1</td>
<td>RemainingStepGetDelegate</td>
</tr>
<tr>
<td>9.19.2.2</td>
<td>RemainingStepSetDelegate</td>
</tr>
<tr>
<td>9.19.2.3</td>
<td>StepCountGetDelegate</td>
</tr>
<tr>
<td>9.20</td>
<td>IrisInstanceTable.h File Reference</td>
</tr>
<tr>
<td>9.20.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>9.20.2</td>
<td>Typedef Documentation</td>
</tr>
<tr>
<td>9.20.2.1</td>
<td>TableReadDelegate</td>
</tr>
<tr>
<td>9.20.2.2</td>
<td>TableWriteDelegate</td>
</tr>
<tr>
<td>9.21</td>
<td>IrisInstantiationContext.h File Reference</td>
</tr>
<tr>
<td>9.21.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>9.22</td>
<td>IrisParameterBuilder.h File Reference</td>
</tr>
<tr>
<td>9.22.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>9.23</td>
<td>IrisPluginFactory.h File Reference</td>
</tr>
<tr>
<td>9.23.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>9.23.2</td>
<td>Macro Definition Documentation</td>
</tr>
<tr>
<td>9.23.2.1</td>
<td>IRIS_PLUGIN_FACTORY</td>
</tr>
<tr>
<td>9.24</td>
<td>IrisRegisterEventEmitter.h File Reference</td>
</tr>
<tr>
<td>9.24.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>9.25</td>
<td>IrisTcpClient.h File Reference</td>
</tr>
<tr>
<td>9.25.1</td>
<td>Detailed Description</td>
</tr>
</tbody>
</table>
Chapter 1

IrisSupportLib Reference Manual

Copyright © 2018, 2019 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 source code is public and it is intended to be buildable and modifiable by customers to integrate with their components or clients. Headers are found in the directories include/iris and include/iris/detail and the source code is found under the directory IrisSupportLib.

Other information

For more information, see the Iris Developer Guide.

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 number 101319_0100_04_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.

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, 2019 Arm Limited (or its affiliates). All rights reserved.


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


Release Information

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100-00</td>
<td>23 Nov 2018</td>
<td>Non-Confidential</td>
<td>New document for Fast Models v11.5.</td>
</tr>
<tr>
<td>0100-01</td>
<td>26 Feb 2019</td>
<td>Non-Confidential</td>
<td>Update for v11.6.</td>
</tr>
<tr>
<td>0100-02</td>
<td>17 May 2019</td>
<td>Non-Confidential</td>
<td>Update for v11.7.</td>
</tr>
<tr>
<td>0100-03</td>
<td>05 Sep 2019</td>
<td>Non-Confidential</td>
<td>Update for v11.8.</td>
</tr>
<tr>
<td>0100-04</td>
<td>28 Nov 2019</td>
<td>Non-Confidential</td>
<td>Update for v11.9.</td>
</tr>
</tbody>
</table>
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:

```c
#ifdef 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 .......................... 16
- IrisInstanceBuilder event APIs ............................... 29
- IrisInstanceBuilder breakpoint APIs ......................... 37
- IrisInstanceBuilder memory APIs ............................ 43
- IrisInstanceBuilder image loading APIs ..................... 55
- IrisInstanceBuilder image readData callback APIs ........ 59
- IrisInstanceBuilder execution stepping APIs .............. 60
- Disassembler delegate functions ............................. 66
- Semihosting data request flag constants .................... 70
Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

iris::IrisInstanceBuilder::AddressTranslationBuilder ....................................................... 71
iris::IrisInstanceMemory::AddressTranslationInfoAndAccess .............................................. 73
iris::IrisInstanceBuilder::EventSourceBuilder ................................................................. 73
iris::IrisInstanceEvent::EventSourceInfoAndDelegate ......................................................... 78
iris::EventStream ................................................................................................................. 79
iris::IrisEventStream ............................................................................................................. 115
iris::IrisInstanceBuilder::FieldBuilder .................................................................................. 90
IrisConnectionInterface
  iris::IrisCConnection ........................................................................................................... 109
  iris::IrisGlobalInstance ....................................................................................................... 116
  iris::IrisTcpClient ............................................................................................................... 222
IrisEventEmitterBase
  iris::IrisEventEmitter<ARGS> ............................................................................................ 110
iris::IrisEventRegistry ......................................................................................................... 111
iris::IrisInstance ................................................................................................................... 117
iris::IrisInstanceBreakpoint .................................................................................................. 127
iris::IrisInstanceBuilder ........................................................................................................ 132
iris::IrisInstanceDebuggableState ......................................................................................... 151
iris::IrisInstanceDisassembler .............................................................................................. 152
iris::IrisInstanceEvent ........................................................................................................... 153
iris::IrisInstanceFactoryBuilder ............................................................................................ 156
  iris::IrisPluginFactoryBuilder .............................................................................................. 216
iris::IrisInstanceImage .......................................................................................................... 160
iris::IrisInstanceImage_Callback ......................................................................................... 162
iris::IrisInstanceMemory ........................................................................................................ 164
iris::IrisInstancePerInstanceExecution ................................................................................. 168
iris::IrisInstanceResource ..................................................................................................... 170
iris::IrisInstanceSemihosting ................................................................................................. 175
iris::IrisInstanceSimulation .................................................................................................. 177
iris::IrisInstanceSimulationTime ............................................................................................ 185
iris::IrisInstanceStep ............................................................................................................. 190
iris::IrisInstanceTable ............................................................................................................ 192
iris::IrisInstantiationContext ............................................................................................... 194
IrisInterface
  iris::IrisGlobalInstance ....................................................................................................... 116
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>iris::IrisTcpClient</td>
<td></td>
<td>222</td>
</tr>
<tr>
<td>iris::IrisParameterBuilder</td>
<td></td>
<td>199</td>
</tr>
<tr>
<td>iris::IrisPluginFactory&lt; PLUGIN_INSTANCE &gt;</td>
<td></td>
<td>215</td>
</tr>
<tr>
<td>IrisProcessEventsInterface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iris::IrisTcpClient</td>
<td></td>
<td>222</td>
</tr>
<tr>
<td>IrisRegisterEventEmitterBase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iris::IrisRegisterReadEventEmitter&lt; REG_T, ARGS &gt;</td>
<td></td>
<td>218</td>
</tr>
<tr>
<td>iris::IrisRegisterUpdateEventEmitter&lt; REG_T, ARGS &gt;</td>
<td></td>
<td>220</td>
</tr>
<tr>
<td>iris::IrisSimulationResetContext</td>
<td></td>
<td>221</td>
</tr>
<tr>
<td>iris::IrisInstanceBuilder::MemorySpaceBuilder</td>
<td></td>
<td>225</td>
</tr>
<tr>
<td>iris::IrisInstanceBuilder::ParameterBuilder</td>
<td></td>
<td>235</td>
</tr>
<tr>
<td>iris::IrisInstanceBuilder::RegisterBuilder</td>
<td></td>
<td>254</td>
</tr>
<tr>
<td>iris::IrisInstanceResource::ResourceInfoAndAccess</td>
<td></td>
<td>273</td>
</tr>
<tr>
<td>iris::ResourceWriteValue</td>
<td></td>
<td>274</td>
</tr>
<tr>
<td>iris::IrisInstanceBuilder::SemihostingManager</td>
<td></td>
<td>274</td>
</tr>
<tr>
<td>iris::IrisInstanceMemory::SpaceInfoAndAccess</td>
<td></td>
<td>276</td>
</tr>
<tr>
<td>iris::IrisInstanceBuilder::TableBuilder</td>
<td></td>
<td>276</td>
</tr>
<tr>
<td>iris::IrisInstanceBuilder::TableColumnBuilder</td>
<td></td>
<td>287</td>
</tr>
<tr>
<td>iris::IrisInstanceTable::TableInfoAndAccess</td>
<td></td>
<td>292</td>
</tr>
</tbody>
</table>
Chapter 5

Class Index

5.1 Class List

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

- `iris::IrisInstanceBuilder::AddressTranslationBuilder`
  Used to set metadata for an address translation ................................................. 71
- `iris::IrisInstanceMemory::AddressTranslationInfoAndAccess`
  Contains static address translation information ................................................ 73
- `iris::IrisInstanceBuilder::EventSourceBuilder`
  Used to set metadata on an EventSource .......................................................... 73
- `iris::IrisInstanceEvent::EventSourceInfoAndDelegate`
  Contains the metadata and delegates for a single EventSource .......................... 78
- `iris::EventStream`
  Base class for event streams ............................................................................. 79
- `iris::IrisInstanceBuilder::FieldBuilder`
  Used to set metadata on a register field resource .............................................. 90
- `iris::IrisCConnection`
  Provide an IrisConnectionInterface which loads an IrisC library ....................... 109
- `iris::IrisEventEmitter<ARGS>`
  A helper class for generating Iris events .......................................................... 110
- `iris::IrisEventRegistry`
  Class to register Iris event streams for an event .............................................. 111
- `iris::IrisEventStream`
  Event stream class for Iris-specific events ....................................................... 115
- `iris::IrisGlobalInstance` .................................................................................. 116
- `iris::IrisInstance` ............................................................................................ 117
- `iris::IrisInstanceDisassembler`
  Disassembler add-on for IrisInstance .................................................................. 152
- `iris::IrisInstanceEvent`
  Event add-on for IrisInstance ............................................................................ 153
- `iris::IrisInstanceFactoryBuilder`
  A builder class to construct instantiation parameter metadata .......................... 156
- `iris::IrisInstanceImage`
  Image loading add-on for IrisInstance ................................................................ 160
iris::IrisInstanceImage_Callback
   Image loading add-on for IrisInstance clients implementing image_loadDataRead() . . . . . . . 162
iris::IrisInstanceMemory
   Memory add-on for IrisInstance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 164
iris::IrisInstancePerInstanceExecution
   Per-instance execution control add-on for IrisInstance . . . . . . . . . . . . . . . . . . . . . . 168
iris::IrisInstanceResource
   Resource add-on for IrisInstance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 170
iris::IrisInstanceSemihosting
   An IrisInstance add-on that adds simulation functions for the SimulationEngine instance . . . 177
iris::IrisInstanceSimulationTime
   Simulation time add-on for IrisInstance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 185
iris::IrisInstanceStep
   Step add-on for IrisInstance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 190
iris::IrisInstanceTable
   Table add-on for IrisInstance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 192
iris::IrisInstantiationContext
   Provides context when instantiating an Iris instance from a factory . . . . . . . . . . . . . . 194
iris::IrisParameterBuilder
   Helper class to construct instantiation parameters . . . . . . . . . . . . . . . . . . . . . . . . 199
iris::IrisPluginFactory< <PLUGIN_INSTANCE > >
   Set metadata for instantiating a plug-in instance . . . . . . . . . . . . . . . . . . . . . . . . . 215
d::IrisInstanceBuilder::MemorySpaceBuilder
   Used to set metadata for a memory space . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 216
d::IrisInstanceBuilder::ParameterBuilder
   Used to set metadata on a parameter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 235
d::IrisInstanceBuilder::RegisterBuilder
   Used to set metadata on a register resource . . . . . . . . . . . . . . . . . . . . . . . . . . . . 254
d::IrisInstanceResource::ResourceInfoAndAccess
   Entry in 'resourceInfos' . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 273
d::ResourceWriteValue
   SemihostingApis IrisInstanceBuilder semihosting APIs . . . . . . . . . . . . . . . . . . . . . 274
d::IrisInstanceMemory::SpaceInfoAndAccess
   Entry in 'spaceInfos' . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 276
d::IrisInstanceBuilder::TableBuilder
   Used to set metadata for a table . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 276
d::IrisInstanceBuilder::TableColumnBuilder
   Used to set metadata for a table column . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 287
d::IrisInstanceTable::TableInfoAndAccess
   Entry in 'tableInfos' . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 292
Chapter 6

File Index

6.1 File List

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

- IrisCConnection.h
  IrisConnectionInterface implementation based on IrisC .......................... 293
- IrisElfDwarfArm.h
  Constants for the register.canonicalRnScheme "ElfDwarf" for architecture Arm .... 293
- Iris EventEmitter.h
  A utility class for emitting Iris events ............................................. 295
- IrisGlobalInstance.h
  Central instance which lives in the simulation engine and distributes all Iris messages ... 296
- IrisInstance.h
  Boilerplate code for an Iris instance, including clients and components ... 296
- IrisInstanceBreakpoint.h
  Breakpoint add-on to IrisInstance .................................................. 298
- IrisInstanceBuilder.h
  A high level interface to build up functionality on an IrisInstance .............. 300
- IrisInstanceDebuggableState.h
  IrisInstance add-on to implement debuggableState functions .................... 301
- IrisInstanceDisassembler.h
  Disassembler add-on to IrisInstance .............................................. 302
- IrisInstanceEvent.h
  Event add-on to IrisInstance ...................................................... 303
- IrisInstanceFactoryBuilder.h
  A helper class to build instantiation parameter metadata ......................... 304
- IrisInstanceImage.h
  Image-loading add-on to IrisInstance and image-loading callback add-on to the caller ... 305
- IrisInstanceMemory.h
  Memory add-on to IrisInstance ................................................... 306
- IrisInstancePerInstanceExecution.h
  Per-instance execution control add-on to IrisInstance .......................... 309
- IrisInstanceResource.h
  Resource add-on to IrisInstance .................................................. 310
- IrisInstanceSemihosting.h
  IrisInstance add-on to implement semihosting functionality ................... 312
- IrisInstanceSimulation.h
  IrisInstance add-on to implement simulation_ * functions .................... 313
- IrisInstanceSimulationTime.h
  IrisInstance add-on to implement simulationTime functions .................... 316
IrisInstanceStep.h
   Stepping-related add-on to an IrisInstance ........................................... 318
IrisInstanceTable.h
   Table add-on to IrisInstance ................................................................. 319
IrisInstantiationContext.h
   Helper class used to instantiate Iris instances from generic factories .......... 321
IrisParameterBuilder.h
   Helper class to construct instantiation parameters ................................... 321
IrisPluginFactory.h
   A generic plug-in factory for instantiating plug-in instances ....................... 322
IrisRegisterEventEmitter.h
   Utility classes for emitting register read and register update events ............ 323
IrisTcpClient.h
   TCP client used by Iris clients which connect via TCP ................................ 323
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 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::UNIQUIFY = (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 subRscIdStart=IRIS_UINT64_MAX, const std::string &cname=std::string())
  Begin a new resource group.
- ParameterBuilder iris::IrisInstanceBuilder::enhanceParameter (ResourceId rscId)
  Get ParameterBuilder to enhance a parameter.
- RegisterBuilder iris::IrisInstanceBuilder::enhanceRegister (ResourceId rscId)
  Get RegisterBuilder to enhance register.
- const ResourceInfo & iris::IrisInstanceBuilder::getResourceInfo (ResourceId rscId)
  Get ResourceInfo of a previously added register.
- template<typename T , IrisErrorCode(T::*METHOD)(const ResourceInfo &, const ResourceWriteValue &)> void iris::IrisInstanceBuilder::setDefaultResourceDelegates (T *instance)
  Set both read and write resource delegates if they are defined in the same class.
- template<typename T , IrisErrorCode(T::*METHOD)(const ResourceInfo &, const ResourceWriteValue &)> void iris::IrisInstanceBuilder::setDefaultResourceReadDelegate (T *instance)
  Set default read access function for all subsequently added resources.
7.2 IrisInstanceBuilder resource APIs

- 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::setNextSubRscId (uint64_t nextSubRscId)
  Set the rscId 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 rscId, 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()

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

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the resource. This is the same as the 'name' field of ResourceInfo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Human readable description of the resource. This is the same as the 'description' field of ResourceInfo.</td>
</tr>
<tr>
<td>format</td>
<td>The format used to display this resource.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns

A `RegisterBuilder` object that can be used to set additional metadata for this resource.

### 7.2.2.2 addParameter()

```cpp
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**

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the parameter. This is the same as the 'name' field of ResourceInfo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitWidth</td>
<td>Width of the parameter in bits. This is the same as the 'bitWidth' field of ResourceInfo.</td>
</tr>
<tr>
<td>description</td>
<td>Human readable description of the parameter. This is the same as the 'description' field of ResourceInfo.</td>
</tr>
</tbody>
</table>

Returns

A `ParameterBuilder` object that can be used to set additional metadata for this parameter.

### 7.2.2.3 addRegister()

```cpp
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.
7.2 IrisInstanceBuilder resource APIs

Parameters

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

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

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

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

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

7.2.2.4 addStringParameter()

```cpp
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the parameter. This is the same as the 'name' field of ResourceInfo.</td>
</tr>
<tr>
<td>description</td>
<td>Human readable description of the parameter. This is the same as the 'description' field of ResourceInfo.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the register. This is the same as the 'name' field of ResourceInfo.</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Human readable description of the resource. This is the same as the 'description' field of ResourceInfo.</td>
</tr>
</tbody>
</table>

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.
### 7.2 IrisInstanceBuilder resource APIs

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td>Name of the resource group.</td>
</tr>
<tr>
<td><strong>description</strong></td>
<td>Description of the resource group.</td>
</tr>
<tr>
<td><strong>subRscIdStart</strong></td>
<td>If not IRIS_UINT64_MAX, start counting from this subRscId when new resources are added.</td>
</tr>
<tr>
<td><strong>cname</strong></td>
<td>C identifier-style name to use for this group if it is different from <em>name</em>.</td>
</tr>
</tbody>
</table>

See also
- `addParameter`
- `addStringParameter`
- `addRegister`
- `addStringRegister`
- `addNoValueRegister`

#### 7.2.2.7 enhanceParameter()

```cpp
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.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rscId</strong></td>
</tr>
</tbody>
</table>

Returns

A `ParameterBuilder` object that can be used to set additional metadata for this parameter.

#### 7.2.2.8 enhanceRegister()

```cpp
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

| rscId | ResourcId 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()

```cpp
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

| rscId | Resource Id of the resource. |

7.2.2.10 setDefaultResourceDelegates()

```cpp
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

---

Generated by Doxygen
7.2 IrisInstanceBuilder resource APIs

Template Parameters

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines resource read and write delegate methods.</th>
</tr>
</thead>
<tbody>
<tr>
<td>READER</td>
<td>A method of class T which is a resource read delegate.</td>
</tr>
<tr>
<td>WRITER</td>
<td>A method of class T which is a resource write delegate.</td>
</tr>
</tbody>
</table>

Parameters

| instance | An instance of class T on which READER and WRITER should be called. |

7.2.2.11 setDefaultResourceReadDelegate() [1/3]

```cpp
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:

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

Parameters

| delegate | Delegate object which will be called to read resources. |
setDefaultResourceReadDelegate() [2/3]

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->setDefaultResourceReadDelegate<MyClass, &MyClass::myReadFunction>(myInstanceOfMyClass);
builder->addRegister(...); // Uses myReadFunction

Template Parameters

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines a resource read delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is a resource read delegate.</td>
</tr>
</tbody>
</table>

Parameters

| instance | An instance of class T on which METHOD should be called. |

setDefaultResourceReadDelegate() [3/3]

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:
7.2 IrisInstanceBuilder resource APIs

iris::IrisErrorCode myReadFunction(const iris::ResourceInfo &resourceInfo, iris::ResourceReadResult &result);

iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultReadDelegate<myReadFunction>();
builder->addRegister(...); // Uses myReadFunction

Template Parameters

| **FUNC** | A function which is a resource read delegate. |

7.2.2.14 setDefaultResourceWriteDelegate() [1/3]

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

| **delegate** | Delegate object which will be called to write resources. |

7.2.2.15 setDefaultResourceWriteDelegate() [2/3]

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

Generated by Doxygen
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:

```cpp
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

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines a resource write delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is a resource write delegate.</td>
</tr>
</tbody>
</table>

Parameters

| instance    | An instance of class T on which METHOD should be called. |

7.2.2.16 `setDefaultResourceWriteDelegate()` [3/3]

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:

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

Template Parameters

| FUNC | A function that is a resource write delegate. |

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

<table>
<thead>
<tr>
<th>nextSubRsclld</th>
</tr>
</thead>
<tbody>
<tr>
<td>The subRscId that is used for the next resource to be added.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>canonicalRnScheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of the canonical register number scheme used by this instance.</td>
</tr>
</tbody>
</table>
7.2.2.19  setTag()

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

Set a tag for a specific resource.

**Parameters**

<table>
<thead>
<tr>
<th>rscId</th>
<th>Resource Id for the resource that will have this tag set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>Name of the boolean tag that will be set to true.</td>
</tr>
</tbody>
</table>

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::finalizeRegisterReadEvent ()
  
  Finalize set up of an IrisEventEmitter.

- IrisInstanceEvent * iris::IrisInstanceBuilder::getIrisInstanceEvent ()

- 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.

- 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.

- 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]

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

Add metadata for an event source.

**Parameters**

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the new event source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>isHidden</td>
<td>If true, the event source is hidden.</td>
</tr>
</tbody>
</table>

**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]

```cpp
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**

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the new event source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_emitter</td>
<td>The IrisEventEmitter for this event source.</td>
</tr>
<tr>
<td>isHidden</td>
<td>If true, the event source is hidden.</td>
</tr>
</tbody>
</table>

**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 IrisInstanceBuilder event APIs

7.3.2.3 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.

7.3.2.4 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 finalizeRegisterReadEvent() to add the event sources to the IrisInstance.

7.3.2.5 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.

7.3.2.6 setDefaultEsCreateDelegate() [1/3]

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

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

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->setDefaultEsCreateDelegate(delegate);
builder->addEventSource(...); // Uses createEventStream
Parameters

| delegate | Delegate object that will be called to create an event stream. |

### 7.3.2.7 `setDefaultEsCreateDelegate()` [2/3]

```cpp
template<typename T , IrisErrorCode(T::*)(EventStream *&, 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.

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:

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

**Template Parameters**

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines an event stream creation method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is an event stream creation method.</td>
</tr>
</tbody>
</table>

**Parameters**

| instance | The instance of class T on which METHOD should be called. |

### 7.3.2.8 `setDefaultEsCreateDelegate()` [3/3]

```cpp
template<IrisErrorCode (*)(EventStream *&, const EventSourceInfo &, const std::vector<std::string> &)> FUNC>
void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate ( ) [inline]
```

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

Event sources that do not explicitly override the access function using

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

use this delegate.

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

```cpp
iris::IrisErrorCode createEventStream(iris::EventStream*, const iris::EventSourceInfo&, const std::vector<std::string>&)
```

```cpp
builder->setDefaultEsCreateDelegate<&MyClass::createEventStream>();
builder->addEventSource(...); // Uses createEventStream
```

**Template Parameters**

| FUNC | Global function to which to delegate event stream creation. |

7.3.2.9 setRegisterReadEvent() [1/2]

```cpp
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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTER</td>
<td>The Iris rscId of the register accessed.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>True if the read originated from a debug access.</td>
</tr>
<tr>
<td>VALUE</td>
<td>The value that was read.</td>
</tr>
</tbody>
</table>

**Parameters**

| name | Name of the event source. |
| description | Description of the event source. |
Returns

An `EventSourceBuilder` for the event allowing extra custom fields to be added.

### 7.3.2.10 `setRegisterReadEvent()` [2/2]  

```cpp
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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTER</td>
<td>The Iris rscId of the register accessed.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>True if the read originated from a debug access.</td>
</tr>
<tr>
<td>VALUE</td>
<td>The value that was read.</td>
</tr>
</tbody>
</table>

Parameters

| name | Name of the event source. |
| event_emitter | The event_emitter to associate with this event source. |

Returns

An `EventSourceBuilder` for the event allowing extra custom fields to be added.

### 7.3.2.11 `setRegisterUpdateEvent()` [1/2]  

```cpp
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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTER</td>
<td>The Iris rscId of the register accessed.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>True if the update originated from a debug access.</td>
</tr>
<tr>
<td>OLD_VALUE</td>
<td>The value that would have been read before the access was made.</td>
</tr>
<tr>
<td>NEW_VALUE</td>
<td>The value that would be read after the access was made.</td>
</tr>
</tbody>
</table>

Parameters

- `name` Name of the event source.
- `description` Description of the event source.

Returns

An `EventSourceBuilder` for the event allowing extra custom fields to be added.

### 7.3.2.12 setRegisterUpdateEvent()

```cpp
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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTER</td>
<td>The Iris rscId of the register accessed.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>True if the update originated from a debug access.</td>
</tr>
<tr>
<td>OLD_VALUE</td>
<td>The value that would have been read before the access was made.</td>
</tr>
<tr>
<td>NEW_VALUE</td>
<td>The value that would be read after the access was made.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the event source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_emitter</td>
<td>The event_emitter to associate with this event source.</td>
</tr>
</tbody>
</table>

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).

- void `iris::IrisInstanceBuilder::setBreakpointDeleteDelegate` (BreakpointDeleteDelegate delegate)

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

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

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

- void `iris::IrisInstanceBuilder::setBreakpointSetDelegate` (BreakpointSetDelegate delegate)

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

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

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

**bptId**

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()

```cpp
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

<table>
<thead>
<tr>
<th><strong>bptId</strong></th>
<th>Breakpoint id for the breakpoint that was hit.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>time</strong></td>
<td>Simulation time at which the breakpoint was hit.</td>
</tr>
<tr>
<td><strong>pc</strong></td>
<td>Value of the program counter when the breakpoint was hit.</td>
</tr>
<tr>
<td><strong>pcSpaceId</strong></td>
<td>Memory space id for the PC when the breakpoint was hit.</td>
</tr>
</tbody>
</table>

### 7.4.2.3 notifyBreakpointHitData()

```cpp
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bptId</code></td>
<td>Breakpoint id for the breakpoint that was hit.</td>
</tr>
<tr>
<td><code>time</code></td>
<td>Simulation time at which the breakpoint was hit.</td>
</tr>
<tr>
<td><code>pc</code></td>
<td>Value of the program counter when the breakpoint was hit.</td>
</tr>
<tr>
<td><code>pcSpaceId</code></td>
<td>Memory space id for the PC when the breakpoint was hit.</td>
</tr>
<tr>
<td><code>accessAddr</code></td>
<td>Address of the access that hit.</td>
</tr>
<tr>
<td><code>accessSize</code></td>
<td>Size in bytes of the access that hit.</td>
</tr>
<tr>
<td><code>accessRw</code></td>
<td>Access direction. Should be &quot;r&quot; for a read access or &quot;w&quot; for a write access.</td>
</tr>
<tr>
<td><code>data</code></td>
<td>The data transferred by the access that hit.</td>
</tr>
</tbody>
</table>

### 7.4.2.4 notifyBreakpointHitRegister()

```cpp
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bptId</code></td>
<td>Breakpoint id for the breakpoint that was hit.</td>
</tr>
<tr>
<td><code>time</code></td>
<td>Simulation time at which the breakpoint was hit.</td>
</tr>
<tr>
<td><code>pc</code></td>
<td>Value of the program counter when the breakpoint was hit.</td>
</tr>
<tr>
<td><code>pcSpaceId</code></td>
<td>Memory space id for the PC when the breakpoint was hit.</td>
</tr>
<tr>
<td><code>accessRw</code></td>
<td>Access direction. Should be &quot;r&quot; for a read access or &quot;w&quot; for a write access.</td>
</tr>
<tr>
<td><code>data</code></td>
<td>The data transferred by the access that hit.</td>
</tr>
</tbody>
</table>

### 7.4.2.5 setBreakpointDeleteDelegate() [1/3]

```cpp
void iris::IrisInstanceBuilder::setBreakpointDeleteDelegate (  
    BreakpointDeleteDelegate delegate ) [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:

Generated by Doxygen
class MyClass
{
    ...
    iris::IrisErrorCode deleteBreakpoint(iris::BreakpointInfo&);
};

MyClass myInstanceOfMyClass;

BreakpointSetDelegate delegate = BreakpointSetDelegate::make<MyClass,
&MyClass::deleteBreakpoint>(myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setBreakpointDeleteDelegate(delegate);

Parameters

| delegate | Delegate object which will be called to delete a breakpoint. |

7.4.2.6 setBreakpointDeleteDelegate() [2/3]

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(iris::BreakpointInfo&);
};

MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setBreakpointDeleteDelegate<MyClass, &MyClass::deleteBreakpoint>(myInstanceOfMyClass);

Template Parameters

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines a breakpoint delete method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is a breakpoint delete delegate method.</td>
</tr>
</tbody>
</table>

Parameters

| instance   | The instance of class T on which METHOD should be called. |

7.4.2.7 setBreakpointDeleteDelegate() [3/3]

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:

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

Template Parameters

| FUNC | Global function to call when a breakpoint is deleted. |

### 7.4.2.8 setBreakpointSetDelegate() [1/3]

```cpp
void iris::IrisInstanceBuilder::setBreakpointSetDelegate ( BreakpointSetDelegate delegate ) [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:

```cpp
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

| delegate | Delegate object which will be called to set a breakpoint. |

### 7.4.2.9 setBreakpointSetDelegate() [2/3]

```cpp
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:
```cpp
class MyClass {
    ...
    iris::IrisErrorCode setBreakpoint(iris::BreakpointInfo&);
};

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

### Template Parameters

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines a breakpoint set method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is a breakpoint set delegate method.</td>
</tr>
</tbody>
</table>

### Parameters

| instance | The instance of class T on which METHOD should be called. |

### 7.4.2.10 setBreakpointSetDelegate() [3/3]

**template<iris::IrisErrorCode(iris::BreakpointInfo &)* (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:

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

### Template Parameters

| FUNC | Global function to call when a breakpoint is set. |
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.

- 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 (*)(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::setDefaultGetMemorySidebandInfoDelegate (MemoryGetSidebandInfoDelegate delegate)
  
  Set the default sideband info 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::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::setDefaultGetMemorySidebandInfoDelegate ()
  
  Set the default sideband info 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, uint64_t, const AttributeValueMap &, MemoryWriteResult & 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, uint64_t, const AttributeValueMap &, MemoryWriteResult & FUNC> void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate ()
  
  Set the default read 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.
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()

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

| `inSpaceId` | Memory space id for the input memory space of this translation. |
| `outSpaceId` | Memory space id for the output memory space of this translation. |
| `description` | A human readable description of this translation. Return an `AddressTranslationBuilder` object which allows additional configuration of this translation. |

7.5.2.2 addMemorySpace()

Add a memory space.

Add metadata to the memory space indicated by `name`.

By explicitly adding a memory space using this function, the Iris instance can tell clients which memory spaces are supported and a component can provide a specific delegate function to perform memory operations.
Add metadata for one memory space.

Typical use pattern:

```cpp
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
name | 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]

```cpp
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

```cpp
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:
Parameters

**delegate** Delegate object which will be called to translate addresses.

### 7.5.2.4 setDefaultAddressTranslateDelegate()

```cpp
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:

```cpp
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>();
builder->addMemorySpace(...); // Uses translateAddress

Template Parameters

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines an address translation delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is an address translation delegate.</td>
</tr>
</tbody>
</table>

Parameters

**instance** An instance of class T on which METHOD should be called.

### 7.5.2.5 setDefaultAddressTranslateDelegate()

```cpp
template<IrisErrorCode (*)(uint64_t, uint64_t, uint64_t, MemoryAddress_translationResult &)> METHOD
void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate () [inline]
```

```cpp
template<typename T, IrisErrorCode(T::*)(uint64_t, uint64_t, uint64_t, MemoryAddress_translationResult &)> 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

```cpp
template <typename FUNC>
iris::IrisErrorCode translateAddress(MemorySpaceId inSpaceId, uint64_t address,
   MemorySpaceId outSpaceId, iris::MemoryAddressTranslationResult &result);
```

will use this delegate.

Usage:

```cpp
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);
};
```

```cpp
MyClass myInstanceOfMyClass;

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

### Template Parameters

- **FUNC**
  - Global function to call to translate addresses.

---

**7.5.2.6 setDefaultGetMemorySidebandInfoDelegate()**

```cpp
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

```cpp
template <typename FUNC>
void getSidebandInfo(MemorySpaceInfo &spaceInfo, uint64_t address,
   const IrisValueMap &attrib,
   const std::vector<std::string> &request,
   IrisValueMap &result);
```

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:

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

```cpp
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

| **delegate** | Delegate object which will be called to get sideband info. |

#### 7.5.2.7 `setDefaultGetMemorySidebandInfoDelegate()` [2/3]

```cpp
template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, 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

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

will use this delegate.

**Usage:**

```cpp
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);
};
```

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

**Template Parameters**

<table>
<thead>
<tr>
<th><strong>T</strong></th>
<th>Class that defines a sideband info delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>METHOD</strong></td>
<td>A method of class T which is a sideband info delegate.</td>
</tr>
</tbody>
</table>

**Parameters**

| **instance** | An instance of class T on which METHOD should be called. |

#### 7.5.2.8 `setDefaultGetMemorySidebandInfoDelegate()` [3/3]

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

Generated by Doxygen
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:

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

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

**Template Parameters**

| **FUNC** | Global function to call to get sideband info. |

---

### 7.5.2.9 setDefaultMemoryReadDelegate()

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:

```cpp
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);
};
```

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

```cpp
template<typename T, IrisErrorCode(T::*METHOD)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &)
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:

```cpp
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**

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines a memory read delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is a memory read delegate.</td>
</tr>
</tbody>
</table>

**Parameters**

| instance         | An instance of class T on which METHOD should be called. |
7.5.2.11  setDefaultMemoryReadDelegate() [3/3]

```cpp
template< typename IrisErrorCode (*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, 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

```cpp
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:

```cpp
iris::IrisErrorCode readMemory(const iris::MemorySpaceInfo &spaceInfo, uint64_t address, uint64_t byteWidth, uint64_t count, const iris::IrisValueMap &attrib, iris::MemoryReadResult &result);
```

```cpp
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultMemoryReadDelegate<readMemory>();
builder->addMemorySpace{...}; // Uses readMemory
```

**Template Parameters**

| **FUNC** | A memory read delegate function. |

7.5.2.12  setDefaultMemoryWriteDelegate() [1/3]

```cpp
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

```cpp
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:
```cpp
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

- **delegate**: Delegate object which will be called to write memory.

#### 7.5.2.13 setDefaultMemoryWriteDelegate()

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:

```cpp
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
```
7.5 IrisInstanceBuilder memory APIs

7.5.2.14 setDefaultMemoryWriteDelegate()

```
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
```

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/memoryspaces' 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_getProperties()`.

**Parameters**

| canonicalMsnScheme | Name of the canonical memory space number scheme used by this instance. |
7.6 IrisInstanceBuilder image loading APIs

Set up image-loading delegates.

Functions

- 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<uint64_t>&, uint64_t) METHOD>
  void iris::IrisInstanceBuilder::setLoadImageDataDelegate (T *instance)
  Set the delegate to load an image from the data provided.
- template<IrisErrorCode(*)(const std::vector<uint64_t>&, uint64_t) FUNC>
  void iris::IrisInstanceBuilder::setLoadImageDataDelegate ()
  Set the delegate to load an image from the data provided.
- 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.
- template<IrisErrorCode(*)(const std::string&) FUNC>
  void iris::IrisInstanceBuilder::setLoadImageFileDelegate ()
  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]

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

| delegate | Delegate object to call for image loading. |

7.6.2.2 setLoadImageDataDelegate() [2/3]

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

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

Usage:

```cpp
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

| T | Class that defines an image-loading delegate method. |
| METHOD | A method of class T which is an image-loading delegate. |

Parameters

| instance | An instance of class T on which METHOD should be called. |

7.6.2.3 setLoadImageDataDelegate() [3/3]

```cpp
template<IrisErrorCode( ) (const std::vector<uint64_t> &, uint64_t) FUNC>
void iris::IrisInstanceBuilder::setLoadImageDataDelegate ( ) [inline]
```

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

Usage:

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

Template Parameters

| FUNC | Global function to call for image loading. |

7.6.2.4 setLoadImageFileDelegate() [1/3]

```cpp
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:

```cpp
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

| delegate | Delegate object to call for image loading. |

7.6.2.5 setLoadImageFileDelegate() [2/3]

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

Set the delegate to load an image from a file.

Usage:

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

<table>
<thead>
<tr>
<th>$T$</th>
<th>Class that defines an image-loading delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$METHOD$</td>
<td>A method of class $T$ which is an image-loading delegate.</td>
</tr>
</tbody>
</table>

### Parameters

| $instance$ | An instance of class $T$ on which $METHOD$ should be called. |

### 7.6.2.6 `setLoadImageFileDelegate()` [3/3]

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

Set the delegate to load an image from a file.

#### Usage:

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

### Template Parameters

| $FUNC$       | Global function to call for image loading. |

Generated by Doxygen
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()

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

Open an image to be read using `image_loadDataPull()` or `image_loadDataRead()`.

Parameters

- `filename` | 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

- void iris::IrisInstanceBuilder::setRemainingStepGetDelegate (RemainingStepGetDelegate delegate)
  
  Set the delegate to get the remaining steps for this instance.

- template<typename T , IrisErrorCode(T::)() method>
  void iris::IrisInstanceBuilder::setRemainingStepGetDelegate (T *instance)

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

- template<IrisErrorCode()() method>
  void iris::IrisInstanceBuilder::setRemainingStepGetDelegate ()

  Set the delegate to get 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::)() method>
  void iris::IrisInstanceBuilder::setRemainingStepSetDelegate (T *instance)

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

- template<IrisErrorCode()() method>
  void iris::IrisInstanceBuilder::setRemainingStepSetDelegate ()

  Set the delegate to set the remaining steps 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::)() method>
  void iris::IrisInstanceBuilder::setStepCountGetDelegate (T *instance)

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

- template<IrisErrorCode()() method>
  void iris::IrisInstanceBuilder::setStepCountGetDelegate ()

  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 IrisInstanceBuilder execution stepping APIs

7.8.2.1 setRemainingStepGetDelegate() [1/3]

```cpp
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:

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

Parameters

- **delegate**: Delegate object to call to get the remaining steps.

7.8.2.2 setRemainingStepGetDelegate() [2/3]

```cpp
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:

```cpp
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

- **T**: Class that defines a get remaining steps delegate method.
- **METHOD**: A method of class T that is a get remaining steps delegate.
Parameters

| instance | An instance of class T on which METHOD should be called. |

7.8.2.3 setRemainingStepGetDelegate() [3/3]

```cpp
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:

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

Template Parameters

| FUNC | Global function to call to get the remaining steps. |

7.8.2.4 setRemainingStepSetDelegate() [1/3]

```cpp
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:

```cpp
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

| delegate | Delegate object to call to set the remaining steps. |

### 7.8.2.5 setRemainingStepSetDelegate()

```cpp
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:

```cpp
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

| T | Class that defines a set remaining steps delegate method. |
| METHOD | A method of class T that is a set remaining steps delegate. |

Parameters

| instance | An instance of class T on which METHOD should be called. |

### 7.8.2.6 setRemainingStepSetDelegate()

```cpp
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:

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

| **FUNC** | Global function to call to set the remaining steps. |

7.8.2.7  

**setStepCountGetDelegate()** [1/3]

```cpp
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:

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

Parameters

| **delegate** | Delegate object to call to get the step count. |

7.8.2.8  

**setStepCountGetDelegate()** [2/3]

```cpp
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:

```cpp
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);
```
7.8 IrisInstanceBuilder execution stepping APIs

Template Parameters

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines a get step count delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is a get step count delegate.</td>
</tr>
</tbody>
</table>

Parameters

| instance | An instance of class T on which METHOD should be called. |

7.8.2.9 `setStepCountGetDelegate()` [3/3]

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

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

Usage:

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

Template Parameters

| FUNC | Global function to call to get the step count. |
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::setGetCurrentModeDelegate (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 Disassembler delegate functions

### 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.
### 7.9.3.2 attachTo()

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

Attach this *IrisInstance* add-on to a specific *IrisInstance*.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>irisInstance</code></td>
<td>The <em>IrisInstance</em> to attach to.</td>
</tr>
</tbody>
</table>

### 7.9.3.3 IrisInstanceDisassembler()

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

Construct an *IrisInstanceDisassembler*.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>irisInstance</code></td>
<td><em>IrisInstance</em> to attach this add-on to.</td>
</tr>
</tbody>
</table>

### 7.9.3.4 setDisassembleOpcodeDelegate()

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

Set the delegate to get the disassembly of *Opcode*.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>delegate</code></td>
<td>Delegate object that will be called to get the disassembly of an opcode.</td>
</tr>
</tbody>
</table>
7.9 Disassembler delegate functions

7.9.3.5 setGetCurrentModeDelegate()

void iris::IrisInstanceDisassembler::setGetCurrentModeDelegate (GetCurrentDisassemblyModeDelegate delegate) [inline]

Set the delegate to get the current disassembly mode.

Parameters

| delegate | 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

| delegate | 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.

Variables

- static const uint64_t iris::semihost::BLOCK = (0 << 1) 
  Block until enough data has been provided based on the specified size hint.
- static const uint64_t iris::semihost::COOKED = (0 << 0) 
  Request that the client sends data when the user presses the return key.
- static const uint64_t iris::semihost::DEFAULT = COOKED | BLOCK | EMIT_EVENT 
  The default flags used if none are specified.
- static const uint64_t iris::semihost::EMIT_EVENT = (0 << 2) 
  If the buffer is empty, emit an event to request input.
- static const uint64_t iris::semihost::NO_EVENT = (1 << 2) 
  Do not request input. Return only the data currently buffered.
- static const uint64_t iris::semihost::NONBLOCK = (1 << 1) 
  Do not block if sufficient data is not available.
- static const uint64_t iris::semihost::RAW = (1 << 0) 
  Request that the client sends data to the model as soon as it is available.

7.10.1 Detailed Description

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

7.10.2 Variable Documentation

7.10.2.1 BLOCK

const uint64_t iris::semihost::BLOCK = (0 << 1) [static]

Block until enough data has been provided based on the specified size hint.

Remarks

- While the function blocks, Iris messages to all instances continue to be processed.
Chapter 8

Class Documentation

8.1 iris::IrisInstanceBuilder::AddressTranslationBuilder Class Reference

Used to set metadata for an address translation.

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

Public Member Functions

- `AddressTranslationBuilder (IrisInstanceMemory::AddressTranslationInfoAndAccess &info_)`
  - 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.
  - `template< IrisErrorCode(*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) FUNC > AddressTranslationBuilder & setTranslateDelegate ()`
  - 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]

```cpp
AddressTranslationBuilder& iris::IrisInstanceBuilder::AddressTranslationBuilder::setTranslateDelegate (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

| delegate | MemoryAddressTranslateDelegate object |

Returns

A reference to this MemorySpaceBuilder object allowing calls to be chained together.

8.1.2.2 setTranslateDelegate() [2/3]

```
template<typename T, IrisErrorCode(T::*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult&) METHOD>
AddressTranslationBuilder& iris::IrisInstanceBuilder::AddressTranslationBuilder::setTranslateDelegate (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

<table>
<thead>
<tr>
<th>T</th>
<th>A class that defines a method with the right signature to be a memory address translation delegate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A memory address translation delegate method in class T.</td>
</tr>
</tbody>
</table>

Parameters

| instance | The instance of class T on which to call METHOD. |

Returns

A reference to this MemorySpaceBuilder object allowing calls to be chained together.

8.1.2.3 setTranslateDelegate() [3/3]

```
template<IrisErrorCode(*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult&) FU-> NC>
AddressTranslationBuilder& iris::IrisInstanceBuilder::AddressTranslationBuilder::setTranslateDelegate ( ) [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

| FUNC | An address translation delegate function. |

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 (MemorySpaceId inSpaceId, MemorySpaceId outSpaceId, 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::IrisInstanceBuilder::EventSourceBuilder Class Reference

Used to set metadata on an EventSource.

#include <IrisInstanceBuilder.h>
Public Member Functions

- **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 size, const std::string &description)**
  
  Add a field to this event source.

- **EventSourceBuilder (IrisInstanceEvent::EventSourceInfoAndDelegate &info_)**

- **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.

- **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.

- **EventSourceBuilder & setRingBuffer (bool ringBuffer=true)**
  
  Set the ringBuffer field.

8.3.1 Detailed Description

Used to set metadata on an EventSource.

8.3.2 Member Function Documentation

8.3.2.1 addEnumElement()

```cpp
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the enum element.</td>
</tr>
<tr>
<td>symbol</td>
<td>The symbol string that will be displayed instead of the value.</td>
</tr>
<tr>
<td>description</td>
<td>A human readable description of this enum.</td>
</tr>
</tbody>
</table>
Returns

A reference to this EventSourceBuilder object allowing calls to be chained together.

8.3.2.2 addField()

```
EventSourceBuilder& iris::IrisInstanceBuilder::EventSourceBuilder::addField (  
    const std::string & name,  
    const std::string & type,  
    uint64_t size,  
    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

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The type of the field.</td>
</tr>
<tr>
<td>size</td>
<td>The size of the field in bytes.</td>
</tr>
<tr>
<td>description</td>
<td>A human readable description of the field.</td>
</tr>
</tbody>
</table>

Returns

A reference to this EventSourceBuilder object allowing calls to be chained together.

8.3.2.3 setCounter()

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

Set the counter field.

Parameters

| counter | The counter field of the EventSourceInfo object. |

Returns

A reference to this EventSourceBuilder object allowing calls to be chained together.
8.3.2.4 setDescription()

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

Set the description field.

**Parameters**

| description | The description field of the EventSourceInfo object. |

**Returns**

A reference to this EventSourceBuilder object allowing calls to be chained together.

8.3.2.5 setEventStreamCreateDelegate() [1/2]

```
EventSourceBuilder& iris::IrisInstanceBuilder::EventSourceBuilder::setEventStreamCreateDelegate (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**

| delegate | EventStreamCreateDelegate object. |

**Returns**

A reference to this EventSourceBuilder object allowing calls to be chained together.

8.3.2.6 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

<table>
<thead>
<tr>
<th>T</th>
<th>A class that defines a method with the right signature to be an event stream creation method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>An event stream creation delegate method in class T.</td>
</tr>
</tbody>
</table>

Parameters

| instance   | The instance of class T on which to call METHOD.                                                |

Returns

A reference to this EventSourceBuilder object allowing calls to be chained together.

8.3.2.7 setFormat()

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

Set the format field.

Parameters

| format     | The format field of the EventSourceInfo object.                                                  |

Returns

A reference to this EventSourceBuilder object allowing calls to be chained together.

8.3.2.8 setHidden()

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

Hide/unhide this event source.

Parameters

| hidden     | If true, this event source is not listed in event_getEventSources() calls but can still be accessed by event_getEventSource() for clients that know the event source’s name. |

Generated by Doxygen
Returns

A reference to this EventSourceBuilder object allowing calls to be chained together.

8.3.2.9 setName()

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

Set the name field.

Parameters

| name  | The name field of the EventSourceInfo object. |

Returns

A reference to this EventSourceBuilder object allowing calls to be chained together.

8.3.2.10 setRingBuffer()

EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::setRingBuffer (bool ringBuffer = true) [inline]

Set the ringBuffer field.

Parameters

| ringBuffer | The ringBuffer 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

8.4 iris::IrisInstanceEvent::EventSourceInfoAndDelegate Struct Reference

Contains the metadata and delegates for a single EventSource.

#include <IrisInstanceEvent.h>
Public Attributes

- `EventStreamCreateDelegate createEventStream`
- `EventSourceInfo info`

8.4.1 Detailed Description

Contains the metadata and delegates for a single EventSource.

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

- `IrisInstanceEvent.h`

8.5 iris::EventStream Class Reference

Base class for event streams.

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

Inherited by `iris::IrisEventStream`.

Public Member Functions

- `void addField (const IrisU64StringConstant &field, uint64_t value)` 
  Add a uint field value.
- `void addField (const IrisU64StringConstant &field, int64_t value)` 
  Add a sint field value.
- `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 addFieldSlow (const std::string &field, uint64_t value)` 
  Add a uint field value.
- `void addFieldSlow (const std::string &field, int64_t value)` 
  Add a sint 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.
- `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.
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.

- **uint64_t getCountVal () const**
  
  Get the current value of the counter.

- **InstanceId getEcInstId () const**
  
  Get the event callback instance id for this event stream.

- **virtual IrisErrorCode getState (IrisValueMap &fields)**
  
  Query the current state of the event.

- **bool isCounter () const**
  
  Is this event stream a counter.

- **bool isEnabled () const**
  
  Is this event stream currently enabled.

- **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.

- **void setProperties (IrisInstance *irisInstance, const EventSourceInfo *srcInfo, InstanceId ecInstId, const std::string &ecFunc, EventStreamId esId, bool syncEc)**
  
  Initialize this event stream.

- **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().

- **InstanceId ecInstId**
  
  Specify target instance that this event is sent to.

- **bool enabled**
  
  Event is only generated when the event stream is enabled.

- **EventStreamId esId**
  
  The event stream id.

- **IrisU64JsonWriter::Object fieldObj**

- **IrisRequest *internal_req**
8.5 iris::EventStream Class Reference

- IrisInstance * irisInstance
  - basic members —
- std::vector<uint64_t> ranges
- IrisRequest * req
  Generate callback requests.
- const EventSourceInfo * srcInfo
  The event source info.
- uint64_t startVal
  Start value and current value for a counter.
- bool syncEc
  Synchronous callback behavior.

8.5.1 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.

8.5.2 Member Function Documentation

8.5.2.1 addField() [1/4]

```cpp
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

<table>
<thead>
<tr>
<th>field</th>
<th>The name of the field whose value is set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the field.</td>
</tr>
</tbody>
</table>

8.5.2.2 addField() [2/4]

```cpp
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

<table>
<thead>
<tr>
<th>field</th>
<th>The name of the field whose value is set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the field.</td>
</tr>
</tbody>
</table>

### 8.5.2.3 `addField()` [3/4]

```cpp
def 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

<table>
<thead>
<tr>
<th>field</th>
<th>The name of the field whose value is set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the field.</td>
</tr>
</tbody>
</table>

### 8.5.2.4 `addField()` [4/4]

```cpp
def 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

<table>
<thead>
<tr>
<th>field</th>
<th>The name of the field whose value is set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the field.</td>
</tr>
</tbody>
</table>

### 8.5.2.5 `addFieldSlow()` [1/4]

```cpp
def addFieldSlow (const std::string &field, uint64_t value) [inline]
```

Generated by Doxygen
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

<table>
<thead>
<tr>
<th>field</th>
<th>The name of the field whose value is set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the field.</td>
</tr>
</tbody>
</table>

8.5.2.6 addFieldSlow() [2/4]

```cpp
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

<table>
<thead>
<tr>
<th>field</th>
<th>The name of the field whose value is set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the field.</td>
</tr>
</tbody>
</table>

8.5.2.7 addFieldSlow() [3/4]

```cpp
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

<table>
<thead>
<tr>
<th>field</th>
<th>The name of the field whose value is set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the field.</td>
</tr>
</tbody>
</table>
8.5.2.8 addFieldSlow()

```cpp
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**

<table>
<thead>
<tr>
<th>field</th>
<th>The name of the field whose value is set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the field.</td>
</tr>
</tbody>
</table>

8.5.2.9 checkRangePc()

```cpp
bool iris::EventStream::checkRangePc (  
    uint64_t pc ) const
```

Check the range for the PC.

This can optionally be called before generating the callback request (before calling emitEventBegin()).

**Parameters**

| pc     | The program counter value to check.       |

**Returns**

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

8.5.2.10 disable()

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

Disable this event stream.

**Returns**

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

Implemented in `iris::IrisEventStream`. 
8.5.2.11 emitEventBegin() [1/2]

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

Start to emit an event callback.

**Parameters**

<table>
<thead>
<tr>
<th>req</th>
<th>A request object to use to construct the event callback.</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>The time in simulation ticks at which the event occurred.</td>
</tr>
<tr>
<td>pc</td>
<td>The program counter value when the event occurred.</td>
</tr>
</tbody>
</table>

8.5.2.12 emitEventBegin() [2/2]

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

Start to emit an event callback.

**Parameters**

| time | The time in simulation ticks at which the event occurred. |
| pc   | The program counter value when the event occurred.     |

8.5.2.13 emitEventEnd()

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

Emit the callback.

This will also check the ranges and maintain the counter.

**Parameters**

| send | If true, event callbacks are sent to the callee immediately. If false, the callback are not sent immediately, allowing the caller to delay sending. |
8.5.2.14 enable()

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

Enable this event stream.

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.

8.5.2.15 getCountVal()

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

Get the current value of the counter.

Returns
The current value of the event counter.

8.5.2.16 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.

8.5.2.17 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

| fields | 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.

8.5.2.18 isCounter()

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

Is this event stream a counter.

Returns

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

8.5.2.19 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.

8.5.2.20 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.

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'.

8.5.2.21 setCounter()

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

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startVal</td>
<td>The starting value of the counter.</td>
</tr>
<tr>
<td>counterMode</td>
<td>The mode in which this counter operates.</td>
</tr>
</tbody>
</table>

8.5.2.22 setProperties()

```cpp
void iris::EventStream::setProperties (
    IrisInstance * irisInstance,
    const EventSourceInfo * srcInfo,
    InstanceId ecInstId,
    const std::string & ecFunc,
    EventStreamId esId,
    bool syncEc )
```

Initialize this event stream.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irisInstance</td>
<td>The IrisInstance that is producing this stream. This will be used to send event callback requests.</td>
</tr>
<tr>
<td>srcInfo</td>
<td>The metadata for the event source generating this stream.</td>
</tr>
<tr>
<td>ecInstId</td>
<td>The event callback instId: the instance that this stream calls when an event fires.</td>
</tr>
<tr>
<td>ecFunc</td>
<td>The event callback function: the function that is called when an event fires.</td>
</tr>
<tr>
<td>esId</td>
<td>The event stream id for this event stream.</td>
</tr>
<tr>
<td>syncEc</td>
<td>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.</td>
</tr>
</tbody>
</table>

8.5.2.23 setRanges()

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

Set the trace ranges for this event stream.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspect</td>
<td>The field whose range to check.</td>
</tr>
<tr>
<td>ranges</td>
<td>A list where each 3 elements form a 3-tuple of (mask, start, end) values to configure ranges.</td>
</tr>
</tbody>
</table>

Returns

An error code indicating whether the ranges could be set successfully.
8.5.3 Member Data Documentation

8.5.3.1 counter

bool iris::EventStream::counter [protected]
— members for a counter —
Is a counter?

8.5.3.2 irisInstance

IrisInstance* iris::EventStream::irisInstance [protected]
— basic members —
The Iris instance that created this event.

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

- IrisInstanceEvent.h

8.6 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** & **setCanonicalRn** (uint64_t canonicalRn_)
  Set the `canonicalRn` field.

• **FieldBuilder** & **setCanonicalRnElfDwarf** (uint16_t architecture, uint16_t dwarfRegNum)
  Set the `canonicalRn` field for "ElfDwarf" scheme.

• **FieldBuilder** & **setCname** (const std::string &cname)
  Set the `cname` field.

• **FieldBuilder** & **setDescr** (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<typename T, IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &)> METHOD
  **FieldBuilder** & **setReadDelegate** (T *instance)
  Set the delegate to read the resource.

• template<typename T, IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &)> METHOD
  **FieldBuilder** & **setReadDelegate** (ResourceReadDelegate readDelegate)
  Set the delegate to read the resource.

• **FieldBuilder** & **setResetData** (uint64_t value)
  Set the `resetData` field to a value <= 64 bit.

• template<typename T>
  **FieldBuilder** & **setResetData** (std::initializer_list<T> &&t)
  Set the `resetData` field for wide registers.

• 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.

• **FieldBuilder** & **setWriteDelegate** (ResourceWriteDelegate writeDelegate)
Set the delegate to write the resource.

- template<typename T, IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &)
  FieldBuilder & setWriteDelegate (T *instance)
  Set the delegate to write the resource.

FieldBuilder & setWriteDelegate (uint64_t value)
Set the writeMask field to a value <= 64 bit.

FieldBuilder & setWriteMask (std::initializer_list<T> &&t)
Set the writeMask field for wide registers.

FieldBuilder & setWriteMaskFromContainer (const Container &container)
Set the writeMask field for wide registers.

Protected Attributes

- IrisInstanceResource::ResourceInfoAndAccess * info()
- IrisInstanceBuilder * instance_builder()
- RegisterBuilder * parent_reg()
Parameters

<table>
<thead>
<tr>
<th>symbol</th>
<th>The symbol string to be associated with the specified value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of this symbol.</td>
</tr>
<tr>
<td>description</td>
<td>A description of this symbol.</td>
</tr>
</tbody>
</table>

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.2 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

8.6.2.3 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

8.6.2.4 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

<table>
<thead>
<tr>
<th>value</th>
<th>The string value of this symbol. This is also used as the symbols string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A description of this symbol.</td>
</tr>
</tbody>
</table>

Returns

A reference to this `FieldBuilder` object allowing calls to be chained together.

### 8.6.2.5 getRscId() [1/2]

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

Return the rscId that was allocated for this resource.

Returns

The rscId that was allocated for this resource.

### 8.6.2.6 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.

### 8.6.2.7 parent() 

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

Get the `RegisterBuilder` for the parent register.

Returns

The `RegisterBuilder` object for the parent register.

### 8.6.2.8 setAddressOffset()

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

Set the addressOffset field.
Parameters

| addressOffset       | The addressOffset field of the RegisterInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.9 setBitWidth()

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

Set the bitWidth field.

Parameters

| bitWidth         | The bitWidth field of the ResourceInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.10 setCanonicalRn()

FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setCanonicalRn (uint64_t canonicalRn_) [inline]

Set the canonicalRn field.

Note: Use setCanonicalRnElfDwarf() when using the "ElfDwarf" scheme.

Parameters

| canonicalRn | The canonicalRn field of the RegisterInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.
8.6.2.11 setCanonicalRnElfDwarf()

FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setCanonicalRnElfDwarf (  
    uint16_t architecture,  
    uint16_t dwarfRegNum ) [inline]

Set the canonicalRn field for "ElfDwarf" scheme.

Parameters

<table>
<thead>
<tr>
<th>architecture</th>
<th>ELF EM_ constant for architecture.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwarfRegNum</td>
<td>DWARF register number for architecture.</td>
</tr>
</tbody>
</table>

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.12 setCname()

FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setCname (  
    const std::string & cname ) [inline]

Set the cname field.

Parameters

| cname     | The cname field of the ResourceInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.13 setDescr()

FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setDescr (  
    const std::string & description ) [inline]

Set the description field.
Parameters

| description | The description field of the ResourceInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

### 8.6.2.14 setFormat()

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

Set the `format` field.

Parameters

| format | The format field of the ResourceInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

### 8.6.2.15 setLsbOffset()

FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setLsbOffset (uint64_t lsbOffset) [inline]

Set the `lsbOffset` field.

Parameters

| lsbOffset | The lsbOffset field of the RegisterInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.
8.6.2.16 setName()

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

Set the name field.

Parameters

| name | The name field of the ResourceInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.17 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

| parentRscId | The rscId of the parent register. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.18 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

| **FUNC** | A resource read delegate function. |

Returns

A reference to this `FieldBuilder` object allowing calls to be chained together.

---

**8.6.2.19 setReadDelegate() [2/3]**

```cpp
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

| `readDelegate` | ResourceReadDelegate object. |

Returns

A reference to this `FieldBuilder` object allowing calls to be chained together.
8.6.2.20 setReadDelegate() [3/3]

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

<table>
<thead>
<tr>
<th>T</th>
<th>A class that defines a method with the right signature to be a resource read delegate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A resource read delegate method in class T.</td>
</tr>
</tbody>
</table>

Parameters

| instance | The instance of class T on which to call METHOD.                                    |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.
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

| value | resetData value of the register. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

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

| t | Braced initializer-list. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.
8.6.2.23 setResetDataFromContainer()

```cpp
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

| container | Container containing the value in 64-bit chunks. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.24 setResetString()

```cpp
FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setResetString ( 
    const std::string & resetString ) [inline]
```

Set the resetString field.

Set the reset value for string registers.
Parameters

| resetString | The resetString field of the RegisterInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

### 8.6.2.25 setRwMode()

FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setRwMode (const std::string & rwMode) [inline]

Set the rwMode field.

Parameters

| rwMode | The rwMode field of the ResourceInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

### 8.6.2.26 setSubRscId()

FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setSubRscId (uint64_t subRscId) [inline]

Set the subRscId field.

Parameters

| subRscId | The subRscId field of the ResourceInfo object. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.
8.6.2.27 setTag() [1/2]

FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setTag (const std::string & tag) [inline]

Set the named boolean tag to true (e.g. isPc)

Parameters

| tag | The name of the tag to set. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.28 setTag() [2/2]

FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setTag (const std::string & tag, const IrisValue & value) [inline]

Set a tag to the specified value.

Parameters

| tag | The name of the tag to set. |
| value | The value to set the tag to. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.29 setType()

FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setType (const std::string & type) [inline]

Set the type field.
Parameters

| type       | The type field of the ResourceInfo object |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.30 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

| FUNC      | A resource write delegate function. |

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.31 setWriteDelegate() [2/3]

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

| `T` | A class that defines a method with the right signature to be a resource write delegate. |
| `METHOD` | A resource write delegate method in class `T`. |

Parameters

| `instance` | The instance of class `T` on which to call `METHOD`. |

Returns

A reference to this `FieldBuilder` object allowing calls to be chained together.

8.6.2.32 `setWriteDelegate()` [3/3]

```cpp
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.
8.6 iris::IrisInstanceBuilder::FieldBuilder Class Reference

See also

IrisInstanceBuilder::setDefaultResourceWriteDelegate

Parameters

<table>
<thead>
<tr>
<th>writeDelegate</th>
<th>ResourceWriteDelegate object.</th>
</tr>
</thead>
</table>

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.33 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

<table>
<thead>
<tr>
<th>t</th>
<th>Braced initializer-list.</th>
</tr>
</thead>
</table>

Returns

A reference to this FieldBuilder object allowing calls to be chained together.

8.6.2.34 setWriteMask() [2/2]

```
FieldBuilder& iris::IrisInstanceBuilder::FieldBuilder::setWriteMask (  
    uint64_t value ) [inline]
```
Set the `writeMask` field to a value \( \leq 64 \) 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**

| value | writeMask value of the register. |

**Returns**

A reference to this `FieldBuilder` object allowing calls to be chained together.

### 8.6.2.35 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**

| container | 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

8.7 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_Functions * 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) 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 * channelId_out)
- int64_t IrisC_unregisterChannel (uint64_t channelId)
- 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.
8.7.1 Detailed Description

Provide an IrisConnectionInterface which loads an IrisC library.

See also

- IrisTcpClient
- IrisGlobalInstance

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

- IrisCConnection.h

8.8 iris::IrisEventEmitter<ARGS> Class Template Reference

A helper class for generating Iris events.

#include <IrisEventEmitter.h>

Inherits IrisEventEmitterBase.

Public Member Functions

- iris::IrisEventEmitter ()
  Construct an event emitter.
- void operator() (ARGS... args)
  Emit an event.

8.8.1 Detailed Description

template<typename... ARGS>
class iris::IrisEventEmitter<ARGS>

A helper class for generating Iris events.

Template Parameters

<table>
<thead>
<tr>
<th>ARG5S</th>
<th>Argument types corresponding to the fields in this event.</th>
</tr>
</thead>
</table>

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")
```cpp
.iridisEventRegistry Class Reference

```addField("FLAG", "bool", 1, "A flag");

// Emit an event
my_event(0x1234, true);

8.8.2 Member Function Documentation

8.8.2.1 operator()()

```cpp
template<typename... ARGS>
void iridis::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

8.9 Iris::IrisEventRegistry Class Reference

Class to register Iris event streams for an event.

```cpp
#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
  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.
- bool registerEventStream (EventStream *evStream)
  Register an event stream.
- bool unregisterEventStream (EventStream *evStream)
  Unregister an event stream.
8.9.1 Detailed Description

Class to register Iris event streams for an event.

8.9.2 Member Function Documentation

8.9.2.1 addField()

```cpp
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**

| T | The type of value. |

**Parameters**

| field | The name of the field whose value is set. |
| value | The value of the field. |

8.9.2.2 addFieldSlow()

```cpp
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**

| T | The type of value. |
Parameters

<table>
<thead>
<tr>
<th>field</th>
<th>The name of the field whose value is set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the field.</td>
</tr>
</tbody>
</table>

8.9.2.3 begin()

```cpp
iterator iris::IrisEventRegistry::begin ( ) const [inline]
```

Get an iterator to the beginning of the event stream set.

See also

```cpp
end
```

Returns

An iterator to the beginning of the event stream set.

8.9.2.4 emitEventEnd()

```cpp
void iris::IrisEventRegistry::emitEventEnd ( ) const
```

Emit the callback.

This also checks the ranges and maintains the counter.

8.9.2.5 empty()

```cpp
bool iris::IrisEventRegistry::empty ( ) const [inline]
```

Return true if no event streams are registered.

Returns

true if no event streams are registered.
8.9.2.6 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.

8.9.2.7 registerEventStream()

bool iris::IrisEventRegistry::registerEventStream ( 
    EventStream * evStream )

Register an event stream.

Parameters

| evStream | The stream to be registered. |

Returns

true if the stream was registered successfully.

8.9.2.8 unregisterEventStream()

bool iris::IrisEventRegistry::unregisterEventStream ( 
    EventStream * evStream )

Unregister an event stream.

Parameters

| evStream | The stream to be unregistered. |

Returns

true if the stream was unregistered successfully.

The documentation for this class was generated from the following file:
8.10 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

8.10.1 Detailed Description

Event stream class for Iris-specific events.

8.10.2 Member Function Documentation

8.10.2.1 disable()

virtual IrisErrorCode iris::IrisEventStream::disable () [virtual]

Disable this event stream.

Returns

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

Implements iris::EventStream.
8.10.2.2 enable()

virtual IrisErrorCode iris::IrisEventStream::enable ( ) [virtual]

Enable this event stream.

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

8.11 iris::IrisGlobalInstance Class Reference

Inherits IrisInterface, and IrisConnectionInterface.

Public Member Functions

- 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

- virtual uint64_t registerChannel (IrisC_CommunicationChannel ∗ channel, const std::string & connection_info="")

- virtual uint64_t registerIrisInterfaceChannel (IrisInterface ∗ iris_interface) override

- virtual void setIrisProxyInterface (IrisProxyInterface ∗ irisProxyInterface_) override

  Set proxy interface.

- void unregisterChannel (uint64_t channelId)

  Unregister a channel.

- virtual void unregisterIrisInterfaceChannel (uint64_t channelId) override

- ∼IrisGlobalInstance ()

  Destructor.

8.11.1 Member Function Documentation
8.11.1 registerChannel()

```cpp
uint64_t iris::IrisGlobalInstance::registerChannel (IrisC_CommunicationChannel * channel, const std::string & connection_info = "")
```

Register a channel. Returns an associated channel id.

8.11.2 registerIrisInterfaceChannel()

```cpp
virtual uint64_t iris::IrisGlobalInstance::registerIrisInterfaceChannel (IrisInterface * iris_interface) [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.

8.11.3 unregisterIrisInterfaceChannel()

```cpp
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

8.12 iris::IrisInstance Class Reference

Public Member Functions

- 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::string & getInstanceName () const
  
  Get the instance name of this instance. This is valid after registerInstance() returns.

- 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.

- const PropertyMap & getPropertyMap () const
  
  Get property map.

- IrisInterface * getRemoteIrisInterface ()
  
  Get the remote Iris interface.

- 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.

- **void notifyStateChanged ()**
  Send an **IRIS_STATE_CHANGED** event if the simulation is not running.

- **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 registerEventCallback (T *instance, const std::string &name, const std::string &description, void(T::*memberFunctionPtr)(IrisReceivedRequest &), const std::string &instanceTypeStr)
  Register a general event callback.
```

```
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 &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.

- **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 setConnectionInterface (IrisConnectionInterface *connection_interface)
  Set the remote connection interface.
```

```
void setEventHandler (IrisInstanceEvent *handler)
  Set the event handler and enable the **IRIS_STATE_CHANGED** event.
```

```
template<T >
void setProperty (const std::string &propertyName, const T &propertyValue)
  Set/add instance property.
```

- **void setThrowOnError (bool throw_on_error)**
  Set default error behavior for **irisCall()**.

- **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.
Static Public Attributes

- static const uint64_t DEFAULT_FLAGS = THROW_ON_ERROR
  Default flags used if not otherwise specified.
- static const uint64_t THROW_ON_ERROR = (1 << 1)
  Throw an exception when an Iris call returns an error response.
- static const uint64_t UNIQUIFY = (1 << 0)
  Uniquify instance name when registering.

Protected Attributes

- InstanceInfo thisInstanceInfo {}
  InstanceInfo of this instance.

8.12.1 Constructor & Destructor Documentation

8.12.1.1 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

<table>
<thead>
<tr>
<th>connection_interface</th>
<th>The IrisConnectionInterface that this instance should use to connect to the simulation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>instName</td>
<td>Name of the instance. This should be prefixed with one of the following, as appropriate:</td>
</tr>
<tr>
<td></td>
<td>• &quot;client.&quot;</td>
</tr>
<tr>
<td></td>
<td>• &quot;component.&quot;</td>
</tr>
<tr>
<td></td>
<td>• &quot;framework.&quot;</td>
</tr>
<tr>
<td>flags</td>
<td>A bitwise OR of Instance Flags. Client instances should usually set the flag iris::IrisInstance::UNIQUIFY.</td>
</tr>
</tbody>
</table>

8.12.1.2 IrisInstance() [2/2]

iris::IrisInstance::IrisInstance (  
  IrisInstantiationContext * context )

Construct a new Iris instance using an IrisInstantiationContext.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>A context object that provides the necessary information to instantiate an instance.</td>
</tr>
</tbody>
</table>

8.12.2 Member Function Documentation

8.12.2.1 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.

8.12.2.2 getInstanceName()

*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 UNIQUIFY flag set and the name was modified to make it unique.

8.12.2.3 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.
8.12.2.4 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.

8.12.2.5 getPropertyMap()

const PropertyMap& iris::IrisInstance::getPropertyMap () const [inline]

Get property map.

This can be used to lookup properties: getWithDefault(my_instance->getPropertyMap(), "myStringProperty", "").getAsString();

8.12.2.6 getRemoteIrisInterface()

IrisInterface* iris::IrisInstance::getRemoteIrisInterface () [inline]

Get the remote Iris interface.

Returns

Returns the IrisInterface that this instance sends requests and responses to.

8.12.2.7 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().
8.12.2.8 **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:

```cpp
iris::IrisErrorCode code = irisCallNoThrow().resource_read(...);
```

8.12.2.9 **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:

```cpp
try
{
    irisCall().resource_read(...);
}
catch (iris::IrisErrorException &e)
{
    ...
}
```

8.12.2.10 **registerEventCallback()** [1/3]

```cpp
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**

<table>
<thead>
<tr>
<th><strong>instance</strong></th>
<th>An instance of class T on which to call the member function.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td>Name of the function as it will be published.</td>
</tr>
<tr>
<td><strong>description</strong></td>
<td>Description of this event callback function.</td>
</tr>
<tr>
<td><strong>memberFunctionPtr</strong></td>
<td>Pointer to the C++ implementation of the function.</td>
</tr>
<tr>
<td><strong>instanceTypeStr</strong></td>
<td>The name of class T. This is only used for logging purposes.</td>
</tr>
</tbody>
</table>
8.12.2.11 registerEventCallback() [2/3]

```cpp
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

<table>
<thead>
<tr>
<th>delegate</th>
<th>EventCallbackDelegate to call to handle the function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the function as it will be published.</td>
</tr>
<tr>
<td>description</td>
<td>Description of this event callback function.</td>
</tr>
<tr>
<td>dlgInstanceTypeStr</td>
<td>The name of the delegate type. This is only used for logging purposes.</td>
</tr>
</tbody>
</table>

8.12.2.12 registerEventCallback() [3/3]

```cpp
template<typename T , IrisErrorCode(T::*)(uint64_t, const AttributeValueMap &, uint64_t, uint64_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

<table>
<thead>
<tr>
<th>instance</th>
<th>An instance of class T on which to call the delegate T::METHOD().</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the function as it will be published.</td>
</tr>
<tr>
<td>description</td>
<td>Description of this event callback function.</td>
</tr>
<tr>
<td>dlgInstanceTypeStr</td>
<td>The name of the delegate type. This is only used for logging purposes.</td>
</tr>
</tbody>
</table>

8.12.2.13 registerFunction()

```cpp
template<class T >
void iris::IrisInstance::registerFunction (
    T * instance,
    const std::string & name,
```

Generated by Doxygen
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, implFunctionName, functionInfoJson)

Parameters

| instance | An instance of class T on which to call the member function. |
| name     | Name of the function as it will be published.             |
| memberFunctionPtr | Pointer to the C++ implementation of the function. |
| functionInfoJson | A string containing the JSON-encoded FunctionInfo object for this function. |
| instanceTypeStr | The name of class T. This is only used for logging purposes. |

8.12.2.14 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

| instName | Name of the instance. This should be prefixed with one of the following, as appropriate: |
| flags | A bitwise OR of Instance Flags. Client instances should usually set the flag iris::IrisInstance::UNIQUIFY. |

• "client."
• "component."
• "framework."

8.12.2.15 sendRequest()

bool iris::IrisInstance::sendRequest ( 
IrisRequest & req ) [inline]

Send an Iris request or notification and potentially wait for a response.

Parameters

| req | 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.

8.12.2.16 sendResponse()

```cpp
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

- `response` The Iris response message to send.

8.12.2.17 setConnectionInterface()

```cpp
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

- `connection_interface` The interface used to connect to an Iris simulation.

8.12.2.18 setProperty()

```cpp
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.
### 8.13 iris::IrisInstanceBreakpoint Class Reference

Breakpoint add-on for IrisInstance.

```
#include <IrisInstanceBreakpoint.h>
```

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

- IrisInstance.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.

- **IrisInstanceBreakpoint** (IrisInstance *irisInstance = nullptr)
  
  Set the event handler used to notify the clients that enable the IRIS BREAKPOINT_HIT event.

  Example usage:

  ```cpp
  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.

8.13.1 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:

```cpp
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);
```

8.13.2 Member Function Documentation

8.13.2.1 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

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the condition.</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The type of the value that clients set to configure the condition.</td>
</tr>
<tr>
<td>description</td>
<td>A description of the condition.</td>
</tr>
<tr>
<td>bpt_types</td>
<td>A list of breakpoint types that this condition can be applied to. An empty list indicates all types.</td>
</tr>
</tbody>
</table>

### 8.13.2.2 attachTo()

```cpp
def void attachTo (IrisInstance * irisInstance)
```

Attach this `IrisInstance` add-on to a specific `IrisInstance`.

Only use this method if `nullptr` was passed to the constructor.

Parameters

| irisInstance | The `IrisInstance` to attach to. |

### 8.13.2.3 getBreakpointInfo()

```cpp
def const BreakpointInfo* getBreakpointInfo (BreakpointId bptId)
```

Get BreakpointInfo for a breakpoint id.

Parameters

| bptId       | 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.

### 8.13.2.4 notifyBreakpointHit()

```cpp
def void notifyBreakpointHit (BreakpointId bptId, uint64_t time,
```

Generated by Doxygen
Notify clients that a code breakpoint was hit.

It notifies clients by emitting an `IRIS_BREAKPOINT_HIT` event.

**Parameters**

| bptId | Breakpoint id for the breakpoint that was hit. |
| time  | Simulation time at which the breakpoint hit.   |
| pc    | Value of the relevant program counter when the event hit. |
| pcSpaceId | Memory space Id for the memory space that the PC address corresponds to. |

### 8.13.2.5 notifyBreakpointHitData()

```cpp
def 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**

| bptId | Breakpoint id for the breakpoint that was hit. |
| time  | Simulation time at which the breakpoint hit.   |
| pc    | Value of the relevant program counter when the event hit. |
| pcSpaceId | Memory space Id for the memory space that the PC address corresponds to. |
| accessAddr | The address of the data access that triggered the breakpoint. |
| accessSize | The size of the data access that triggered the breakpoint. |
| accessRw | Indicates the direction of the access. "r" = read access or "w" = write access. |
| data | The data that was written or read during the access that triggered the breakpoint. |

### 8.13.2.6 notifyBreakpointHitRegister()

```cpp
def void iris::IrisInstanceBreakpoint::notifyBreakpointHitRegister (  
    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 )
```
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

<table>
<thead>
<tr>
<th>bptId</th>
<th>Breakpoint id for the breakpoint that was hit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Simulation time at which the breakpoint hit.</td>
</tr>
<tr>
<td>pc</td>
<td>Value of the relevant program counter when the event hit.</td>
</tr>
<tr>
<td>pcSpaceId</td>
<td>Memory space Id for the memory space that the PC address corresponds to.</td>
</tr>
<tr>
<td>accessRw</td>
<td>Indicates the direction of the access. &quot;r&quot; = read access or &quot;w&quot; = write access.</td>
</tr>
<tr>
<td>data</td>
<td>The data that was written or read during the access that triggered the breakpoint.</td>
</tr>
</tbody>
</table>

8.13.2.7 setBreakpointDeleteDelegate()

void iris::IrisInstanceBreakpoint::setBreakpointDeleteDelegate (BreakpointDeleteDelegate delegate)

Set breakpoint delete delegate for all breakpoints deleted by this instance.

Parameters

| delegate | A BreakpointDeleteDelegate to call when a breakpoint is deleted. |

8.13.2.8 setBreakpointSetDelegate()

void iris::IrisInstanceBreakpoint::setBreakpointSetDelegate (BreakpointSetDelegate delegate)

Set breakpoint set delegate for all breakpoints set by this instance.

Parameters

| delegate | A BreakpointSetDelegate to call when a breakpoint is set. |
8.13.2.9 setEventHandler()

```cpp
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.

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

- IrisInstanceBreakpoint.h

8.14 iris::IrisInstanceBuilder Class Reference

Builder interface to populate an IrisInstance with registers, memory etc.

```cpp
#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>(1))
  Add an optional component-specific condition.

- **EventSourceBuilder addEventSource** (const std::string &name, const std::string &type, const std::string &description, const std::vector<std::string> bpt_types=std::vector<std::string>())
  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=IRIS_UINT64_MAX, const std::string &cname=std::string())
  Begin a new resource group.

- **ParameterBuilder enhanceParameter** (ResourceId rscId)
  Get ParameterBuilder to enhance a parameter.

- **RegisterBuilder enhanceRegister** (ResourceId rscId)
  Get RegisterBuilder to enhance register.

- **void finalizeRegisterReadEvent** ()
- **void finalizeRegisterUpdateEvent** ()
  Finalize setup of an IrisEventEmitter.

- **const BreakpointInfo *getBreakpointInfo** (BreakpointId bptId)
  Get the breakpoint information for a given breakpoint.

- **IrisInstanceEvent *getIrisInstanceEvent** ()
  Get EventInfo of a previously added register.

- **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 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<typename T , IrisErrorCode(T::*)(const BreakpointInfo &) FUNC>
  void setBreakpointDeleteDelegate ()

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

• void setBreakpointSetDelegate (BreakpointSetDelegate delegate)

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

• template<typename T , IrisErrorCode(T::*)(EventStream *, const EventSourceInfo &, const std::vector<std::string> &)> METHOD>
  void setBreakpointSetDelegate (T *instance)

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

• template<typename T , IrisErrorCode(T::*)(EventStream *, const EventSourceInfo &, const std::vector<std::string> &)> FUNC>
  void setBreakpointSetDelegate ()

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

• void setDefaultAddressTranslateDelegate (MemoryAddressTranslateDelegate delegate=MemoryAddressTranslateDelegate())

Set the default address translation 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 setDefaultAddressTranslateDelegate (T *instance)

Set the default address translation 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 setDefaultAddressTranslateDelegate ()

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

• 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<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, const IrisValueMap &, const std::vector<std::string> &, IrisValueMap &)> METHOD>
  void setDefaultEsCreateDelegate (T *instance)

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

• void setDefaultGetMemorySidebandInfoDelegate (MemoryGetSidebandInfoDelegate delegate)

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

• template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, 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<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, const IrisValueMap &) METHOD>
  void setDefaultGetMemorySidebandInfoDelegate ()

Set the default sideband info 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 &, 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, const AttributeValueMap &, MemoryReadResult &)
  FUNC>
void setDefaultMemoryReadDelegate ()

  Set the default read 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, const AttributeValueMap &, const uint64_t *
  , MemoryWriteResult &)
  METHOD>
void setDefaultMemoryWriteDelegate (T *instance)

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

• template<IrisErrorCode(+) const MemorySpaceInfo &
  , uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *
  , MemoryWriteResult &)
  FUNC>
void setDefaultMemoryWriteDelegate ()

  Set 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.

• 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 &
  , ResourceReadResult &)
  FUNC>
void setDefaultResourceReadDelegate ()

  Set default read 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 ResourceInfo &
  , const ResourceWriteValue &)
  FUNC>
void setDefaultResourceWriteDelegate ()

  Set default write access function for all subsequently added resources.

• void setDefaultTableReadDelegate (TableReadDelegate delegate=TableReadDelegate())

  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.

• template<IrisErrorCode(+) const TableInfo &
  , uint64_t, uint64_t, TableReadResult &)
  FUNC>
void setDefaultTableReadDelegate ()

  Set the default table read function for all subsequently added tables.

• void setDefaultTableWriteDelegate (TableWriteDelegate delegate=TableWriteDelegate())

  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.

• template<IrisErrorCode(+) const TableInfo &
  , const TableRecords &, TableWriteResult &)
  FUNC>
void setDefaultTableWriteDelegate ()

  Set the default table write function for all subsequently added tables.

• void setExecutionStateGetDelegate (PerInstanceExecutionStateGetDelegate delegate)

  Set the delegate to get the execution state for this instance.
private:

• 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 setExecutionStateGetDelegate ()

  Set the delegate to get 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(+)(bool)
FUNC>
void setExecutionStateSetDelegate ()

  Set the delegate to set the execution state for this instance.

• void setLoadImageDataDelegate (ImageLoadDataDelegate delegate=ImageLoadDataDelegate())

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

• template<
typename T , IrisErrorCode(T::*)(const std::vector<uint64_t> &, uint64_t)
METHOD>
void setLoadImageDataDelegate (T *instance)

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

• template<IrisErrorCode(+)(const std::vector<uint64_t> &, uint64_t)
FUNC>
void setLoadImageDataDelegate ()

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

• 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.

• template<IrisErrorCode(+)(const std::string &)
FUNC>
void setLoadImageFileDelegate ()

  Set the delegate to load an image from a file.

• void setNextSubRscId (uint64_t nextSubRscId)

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

• voidsetPropertyCanonicalMsnScheme (const std::string &canonicalMsnScheme)

  Set the memory:canonicalMsnScheme instance property.

• voidsetPropertyCanonicalRnScheme (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.

• 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 & enclave, const std::string & name)\n  void setRemainingStepGetDelegate()

  Set the delegate to get 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 & enclave, const std::string & name)\n  void setRemainingStepSetDelegate(T *instance)

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

- template<irisErrorCode(+)>(uint64_t & enclave, const std::string & name)\n  void setRemainingStepSetDelegate()

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

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::)(std::string & name)\n  void setStepCountGetDelegate(T *instance)

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

- template<irisErrorCode(+)>(uint64_t & enclave, const std::string & name)\n  void setStepCountGetDelegate()

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

Set a tag for a specific resource.

- void setTag(ResourceId rscId, const std::string & tag)

  Set a tag for a specific resource.

- void addDisassemblyMode(const std::string & name, const std::string & description)

  Add a disassembly mode.

- void setDisassembleOpcodeDelegate(DisassembleOpcodeDelegate delegate)

  Set the delegate to get the disassembly of Opcode.

- template<typename T, IrisErrorCode(T::)(const std::vector<uint64_t> & enclave, \n  const std::string & name, DisassembleContext &, DisassemblyLine & name)\n  void setDisassembleOpcodeDelegate(T *instance)

  Set the delegate to get the disassembly of Opcode.

- template<irisErrorCode(+)>(const std::vector<uint64_t> & enclave, \n  const std::string & name, DisassembleContext &, DisassemblyLine & name)\n  void setDisassembleOpcodeDelegate()

  Set the delegate to get the disassembly of Opcode.

- void addDisassemblyMode(const std::string & name, const std::string & description)

  Add a disassembly mode.

- void setDbgStateSetRequestDelegate(DebuggableStateSetRequestDelegate delegate=DebuggableStateSetRequestDelegate())

  Set the delegate to get the debugging state set request.
### 8.14.1 Detailed Description

Builder interface to populate an IrisInstance with registers, memory etc.

See DummyComponent.h for a working example.

### 8.14.2 Constructor & Destructor Documentation

#### 8.14.2.1 IrisInstanceBuilder()

```cpp
iris::IrisInstanceBuilder::IrisInstanceBuilder ( IrisInstance * iris_instance )
```

Construct an IrisInstanceBuilder for an Iris instance.

**Parameters**

- **iris_instance** [The instance to build.]
8.14.3 Member Function Documentation

8.14.3.1 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

| name | Name of the new table. |

Returns

A TableBuilder object than can be used to set metadata for the new table.

8.14.3.2 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.
8.14.3.3  setDbgStateDelegates()

```cpp
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:

```cpp
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

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines both a debuggable state request set and a get acknowledge delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_REQUEST</td>
<td>A method of class T which is a debuggable state request set delegate.</td>
</tr>
<tr>
<td>GET_ACKNOWLEDGE</td>
<td>A method of class T which is a debuggable state get acknowledge delegate.</td>
</tr>
</tbody>
</table>

Parameters

| instance | An instance of class T on which SET_REQUEST and GET_ACKNOWLEDGE should be called. |

8.14.3.4  setDbgStateGetAcknowledgeDelegate() [1/3]

```cpp
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:

```cpp
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

| delegate | Delegate object to call to get the debuggable state acknowledge flag. |

8.14.3.5 setDbgStateGetAcknowledgeDelegate() [2/3]

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

| T | Class that defines a debuggable state get acknowledge delegate method. |
| METHOD | A method of class T which is a debuggable state get acknowledge delegate. |

Parameters

| instance | An instance of class T on which METHOD should be called. |

8.14.3.6 setDbgStateGetAcknowledgeDelegate() [3/3]

template<IrisErrorCode(*)(bool &) FUNC>
void iris::IrisInstanceBuilder::setDbgStateGetAcknowledgeDelegate( ) [inline]

Set the delegate to get the debuggable state acknowledge flag for this instance.

Usage:

Generated by Doxygen
iris::IrisErrorCode getAcknowledgeFlag(bool &debuggable_state_acknowledge);

iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateGetAcknowledgeDelegate<&getAcknowledgeFlag>();

Template Parameters

| FUNC | Global function to call to get the debuggable state acknowledge flag. |

8.14.3.7 setDbgStateSetRequestDelegate() [1/3]

void iris::IrisInstanceBuilder::setDbgStateSetRequestDelegate ( DebuggableStateSetRequestDelegate delegate = DebuggableStateSetRequestDelegate() ) [inline]

d cellpadding_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

| delegate | Delegate object to call to set the debuggable state request flag. |

8.14.3.8 setDbgStateSetRequestDelegate() [2/3]

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);

8.14.3.9 setDbgStateSetRequestDelegate() [3/3]

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

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines a debuggable state request set delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is a debuggable state request set delegate.</td>
</tr>
</tbody>
</table>

Parameters

| instance | An instance of class T on which METHOD should be called. |

8.14.3.10 setDefaultTableReadDelegate() [1/3]

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:

```cpp
class MyClass
{
    iris::IrisErrorCode readTable(const iris::TableInfo &tableInfo, uint64_t index, uint64_t count, iris::TableReadResult &result);
};
```

```cpp
MyClass myInstanceOfMyClass;
iris::TableReadDelegate delegate = iris::TableReadDelegate::make<MyClass, &MyClass::readTable>(&myInstanceOfMyClass);
```

```cpp
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableReadDelegate(delegate);
builder->addTable(...); // Uses readTable
```

**Parameters**

- **delegate** Delegate object to call to read a table.

### 8.14.3.11 setDefaultTableReadDelegate() [2/3]

```cpp
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

```cpp
addTable(...).setReadDelegate(...)```

will use this delegate.

Usage:

```cpp
class MyClass
{
    ... iris::IrisErrorCode readTable(const iris::TableInfo &tableInfo, uint64_t index, uint64_t count, iris::TableReadResult &result);
};
```

```cpp
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableReadDelegate<MyClass, &MyClass::readTable>(&myInstanceOfMyClass);
builder->addTable(...); // Uses readTable
```
### 8.14.3.12 setDefaultTableReadDelegate() [3/3]

```
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:

```cpp
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
```

### 8.14.3.13 setDefaultTableWriteDelegate() [1/3]

```
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:

```cpp
class MyClass
{
  ...
  iris::IrisErrorCode writeTable(const iris::TableInfo &tableInfo,
  const iris::TableRecords &records,
  iris::TableWriteResult &result);
};
```

```cpp
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**

| delegate | Delegate object to call to write a table. |

8.14.3.14  `setDefaultTableWriteDelegate()` [2/3]

```cpp
template<typename T, IrisErrorCode(T::*)(const TableInfo &, const TableRecords &, Table←
WriteResult &) 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

```cpp
addTable(...).setWriteDelegate(...)
```

will use this delegate.

Usage:

```cpp
class MyClass
{
  ...
  iris::IrisErrorCode writeTable(const iris::TableInfo &tableInfo,
  const iris::TableRecords &records,
  iris::TableWriteResult &result);
};
```

```cpp
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableWriteDelegate<MyClass, &MyClass::writeTable>(myInstanceOfMyClass);
builder->addTable(...); // Uses writeTable
```
Template Parameters

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines a table write delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is a table write delegate.</td>
</tr>
</tbody>
</table>

Parameters

| instance | An instance of class T on which METHOD should be called. |

### 8.14.3.15 setDefaultTableWriteDelegate() [3/3]

```cpp
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:

```cpp
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

| FUNC | Global function to call to write a table. |

### 8.14.3.16 setExecutionStateGetDelegate() [1/3]

```cpp
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:
```cpp
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

| delegate | Delegate object to call to get the execution state. |

8.14.3.17 setExecutionStateGetDelegate() [2/3]

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

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines a get execution state delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is a get execution state delegate.</td>
</tr>
</tbody>
</table>

Parameters

| instance | An instance of class T on which METHOD should be called. |

8.14.3.18 setExecutionStateGetDelegate() [3/3]

template<IrisErrorCode(*)(bool &)> FUNC
void iris::IrisInstanceBuilder::setExecutionStateGetDelegate ( ) [inline]
```
Set the delegate to get the execution state for this instance.

**Usage:**

```cpp
iris::IrisErrorCode getState(bool &execution_enabled);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateGetDelegate<&getState>();
```

**Template Parameters**

<table>
<thead>
<tr>
<th><strong>FUNC</strong></th>
<th>Global function to call to get the execution state.</th>
</tr>
</thead>
</table>

8.14.3.19  **setExecutionStateSetDelegate()**  [1/3]

```cpp
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:**

```cpp
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**

| **delegate** | Delegate object to call to set the execution state. |

8.14.3.20  **setExecutionStateSetDelegate()**  [2/3]

```cpp
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:

```cpp
class MyClass
{
...    iris::IrisErrorCode setState(bool enable_execution);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateSetDelegate<MyClass, &MyClass::setState>(&myInstanceOfMyClass);
```

Template Parameters

<table>
<thead>
<tr>
<th>T</th>
<th>Class that defines a set execution state delegate method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A method of class T which is a set execution state delegate.</td>
</tr>
</tbody>
</table>

Parameters

| instance | An instance of class T on which METHOD should be called. |

Set the delegate to set the execution state for this instance.

Usage:

```cpp
iris::IrisErrorCode setState(bool enable_execution);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateSetDelegate<&setState>();
```

Template Parameters

| FUNC | Global function to call to set the execution state. |

Set the delegate to set the execution state for this instance.

Usage:

```cpp
iris::IrisErrorCode setState(bool enable_execution);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateSetDelegate<&setState>();
```

Template Parameters

| FUNC | Global function to call to set the execution state. |

Set the delegate to set the execution state for this instance.

Usage:

```cpp
iris::IrisErrorCode setState(bool enable_execution);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateSetDelegate<&setState>();
```
Disassembler APIs

Set the delegates to get the current disassembly mode.

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

- IrisInstanceBuilder.h

**8.15 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)`
  Set the get acknowledge flag delegate.

- `void setGetAcknowledgeDelegate (DebuggableStateGetAcknowledgeDelegate delegate)`
  Set the get request flag delegate.

- `void setSetRequestDelegate (DebuggableStateSetRequestDelegate delegate)`
  Set the set request flag delegate.

**8.15.1 Detailed Description**

Debuggable-state add-on for IrisInstance.

**8.15.2 Member Function Documentation**

**8.15.2.1 attachTo()**

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

Attach this IrisInstance add-on to a specific IrisInstance.

**Parameters**

- `irisInstance` The IrisInstance to attach to.
8.15.2.2  setGetAcknowledgeDelegate()

```
void iris::IrisInstanceDebuggableState::setGetAcknowledgeDelegate (  
    DebuggableStateGetAcknowledgeDelegate delegate )  [inline]
```

Set the get acknowledge flag delegate.

**Parameters**

| delegate | Delegate that will be called to get the debuggable-state acknowledge flag. |

8.15.2.3  setSetRequestDelegate()

```
void iris::IrisInstanceDebuggableState::setSetRequestDelegate (  
    DebuggableStateSetRequestDelegate delegate )  [inline]
```

Set the set request flag delegate.

**Parameters**

| delegate | 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

8.16  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 setCurrentModeDelegate (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.
8.16.1 Detailed Description

Disassembler add-on for IrisInstance.

This class is used by instances that want to support disassembly functionality.

It implements all Iris disassembler*() 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

8.17 iris::IrisInstanceEvent Class Reference

Event add-on for IrisInstance.

#include <IrisInstanceEvent.h>

Classes

- struct EventSourceInfoAndDelegate
  
  *Contains the metadata and delegates for a single EventSource.*

Public Member Functions

- EventSourceInfoAndDelegate & addEventSource (const std::string &name, bool isHidden=false)
  
  *Add metadata for an event source.*

- uint64_t addEventSource (const EventSourceInfoAndDelegate &info)
  
  *Add metadata for an event source.*

- void attachTo (IrisInstance *irisInstance)
  
  *Attach this IrisInstanceEvent add-on to a specific IrisInstance.*

- IrisInstanceEvent (IrisInstance *irisInstance=nullptr)
  
  *Construct an IrisInstanceEvent add-on.*

- void setDefaultEsCreateDelegate (EventStreamCreateDelegate delegate)
  
  *Set the default delegate for creating EventStreams for the attached instance.*
8.17.1 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.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 IrisInstanceEvent()

iris::IrisInstanceEvent::IrisInstanceEvent ( IrisInstance * irisInstance = nullptr )

Construct an IrisInstanceEvent add-on.

Parameters

| irisInstance | The IrisInstance to which to attach this add-on. |

8.17.3 Member Function Documentation

8.17.3.1 addEventSource() [1/2]

EventSourceInfoAndDelegate& iris::IrisInstanceEvent::addEventSource ( const std::string & name,
bool isHidden = false )

Add metadata for an event source.

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the event source.</th>
</tr>
</thead>
<tbody>
<tr>
<td>isHidden</td>
<td>If true, this event source is hidden. The EventSourceInfo is not included in the list of event sources returned by event_getEventSources() but can be still be accessed by event_getEventSource() if the client knows the name of the hidden event.</td>
</tr>
</tbody>
</table>
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()`.

### 8.17.3.2 addEventSource() [2/2]

```cpp
uint64_t iris::IrisInstanceEvent::addEventSource (
    const EventSourceInfoAndDelegate & info
)
```

Add metadata for an event source.

**Parameters**

| info | The metadata and event-specific delegates (if applicable) for a new event to add. |

**Returns**

The evSrcId of the newly added event source.

### 8.17.3.3 attachTo()

```cpp
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**

| irisInstance | The `IrisInstance` to which to attach this add-on. |

### 8.17.3.4 setDefaultEsCreateDelegate()

```cpp
void iris::IrisInstanceEvent::setDefaultEsCreateDelegate (
    EventStreamCreateDelegate delegate
)
```

Set the default delegate for creating EventStreams for the attached instance.
### Parameters

| delegate | 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

## 8.18 iris::IrisInstanceFactoryBuilder Class Reference

A builder class to construct instantiation parameter metadata.

#include <IrisInstanceFactoryBuilder.h>

Inherited by iris::IrisPluginFactoryBuilder.

### Public Member Functions

- IrisParameterBuilder addBooleanParameter (const std::string &name, const std::string &description)
  - Add a new boolean parameter.
- IrisParameterBuilder addHiddenBooleanParameter (const std::string &name, const std::string &description)
  - Add a new hidden boolean parameter.
- IrisParameterBuilder addHiddenStringParameter (const std::string &name, const std::string &description)
  - Add a new hidden string parameter.
- IrisParameterBuilder addHiddenParameter (const std::string &name, uint64_t bitWidth, const std::string &description)
  - Add a new hidden numeric 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.
- IrisInstanceFactoryBuilder (const std::string &prefix)
  - Construct an IrisInstanceFactoryBuilder.

## 8.18.1 Detailed Description

A builder class to construct instantiation parameter metadata.

## 8.18.2 Constructor & Destructor Documentation

### 8.18.2.1 IrisInstanceFactoryBuilder()

iris::IrisInstanceFactoryBuilder::IrisInstanceFactoryBuilder (const std::string &prefix) [inline]

Construct an IrisInstanceFactoryBuilder.
Parameters

| prefix | All parameters added to this builder are prefixed with this string. |

8.18.3 Member Function Documentation

8.18.3.1 `addBooleanParameter()`

```cpp
IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addBooleanParameter (    
    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

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Description of the parameter.</td>
</tr>
</tbody>
</table>

Returns

An `IrisParameterBuilder` object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.18.3.2 `addHiddenBooleanParameter()`

```cpp
IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addHiddenBooleanParameter (    
    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

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Description of the parameter.</td>
</tr>
</tbody>
</table>
Returns

An IrisParameterBuilder object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.18.3.3 addHiddenStringParameter()

IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addHiddenStringParameter (const std::string & name, const std::string & description) [inline]

Add a new hidden string parameter.

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Description of the parameter.</td>
</tr>
</tbody>
</table>

Returns

An IrisParameterBuilder object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.18.3.4 addHidenParameter()

IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addHidenParameter (const std::string & name, uint64_t bitWidth, const std::string & description) [inline]

Add a new hidden numeric parameter.

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitWidth</td>
<td>Width of the parameter in bits.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the parameter.</td>
</tr>
</tbody>
</table>

Returns

An IrisParameterBuilder object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.
8.18.3.5 addParameter()

IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addParameter (  
    const std::string & name,  
    uint64_t bitWidth,  
    const std::string & description ) [inline]

Add a new numeric parameter.

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitWidth</td>
<td>Width of the parameter in bits.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the parameter.</td>
</tr>
</tbody>
</table>

Returns

An IrisParameterBuilder object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.18.3.6 addStringParameter()

IrisParameterBuilder iris::IrisInstanceFactoryBuilder::addStringParameter (  
    const std::string & name,  
    const std::string & description ) [inline]

Add a new string parameter.

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Description of the parameter.</td>
</tr>
</tbody>
</table>

Returns

An IrisParameterBuilder object which can be used to set further metadata for this parameter. The object is valid until another parameter is added.

8.18.3.7 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.
### 8.18.3.8 `getParameterInfo()`

```cpp
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`

### 8.19 `iris::IrisInstanceImage` Class Reference

Image loading add-on for `IrisInstance`.

```cpp
#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<uint64_t> &data, uint64_t &count)`
  
  Read file data into a `uint64_t` array and record the number of bytes read.

### 8.19.1 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.
8.19.2 Constructor & Destructor Documentation

8.19.2.1 IrisInstanceImage()

iris::IrisInstanceImage::IrisInstanceImage ( 
    IrisInstance * irisInstance = 0 )

Construct a new IrisInstanceImage.

Parameters

| irisInstance | The IrisInstance to attach this add-on to. |

8.19.3 Member Function Documentation

8.19.3.1 attachTo()

void iris::IrisInstanceImage::attachTo ( 
    IrisInstance * irisInstance )

Attach this IrisInstance add-on to a specific IrisInstance.

Parameters

| irisInstance | The IrisInstance to attach this add-on to. |

8.19.3.2 readFileData()

static IrisErrorCode iris::IrisInstanceImage::readFileData ( 
    const std::string & fileName, 
    std::vector< uint64_t > & data, 
    uint64_t & count ) [static]

Read file data into a uint64_t array and record the number of bytes read.

Parameters

| fileName | Name of the file to read. |
| data | A reference to a vector which is populated with the file contents. |
| count | A reference to a variable which is set to the number of bytes that were read. |
Returns

An error code indicating success or failure.

### 8.19.3.3 setLoadImageDataDelegate()

```cpp
void iris::IrisInstanceImage::setLoadImageDataDelegate (ImageLoadDataDelegate delegate)
```

Set image loading from (pushed/pulled) data delegate.

**Parameters**

| delegate | The delegate that will be called to load an image from a data buffer. |

### 8.19.3.4 setLoadImageFileDelegate()

```cpp
void iris::IrisInstanceImage::setLoadImageFileDelegate (ImageLoadFileDelegate delegate)
```

Set image loading from file delegate.

**Parameters**

| delegate | The delegate that will be called to load an image from a file. |

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

- IrisInstanceImage.h

### 8.20 iris::IrisInstanceImage_Callback Class Reference

Image loading add-on for IrisInstance clients implementing image_loadDataRead().

```cpp
#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 read.
Protected Member Functions

- void impl_image_loadDataRead (IrisReceivedRequest &request)
  Implementation of the Iris function image_loadDataRead().

8.20.1 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.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 IrisInstanceImage_Callback()

iris::IrisInstanceImage_Callback::IrisInstanceImage_Callback ( IrisInstance * irisInstance = 0 )

Construct an IrisInstanceImage_Callback add-on.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irisInstance</td>
<td>The IrisInstance to attach this add-on to.</td>
</tr>
</tbody>
</table>

8.20.3 Member Function Documentation

8.20.3.1 attachTo()

void iris::IrisInstanceImage_Callback::attachTo ( IrisInstance * irisInstance )

Attach this IrisInstance add-on to a specific IrisInstance.
Parameters

irisInstance The IrisInstance to attach this add-on to.

8.20.3.2 openImage()

uint64_t iris::IrisInstanceImage_Callback::openImage (const std::string & fileName)

Open an image for read.

Parameters

fileName 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

8.21 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.

8.21.1 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:

```cpp
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

8.21.2 Constructor & Destructor Documentation

8.21.2.1 IrisInstanceMemory()

```cpp
iris::IrisInstanceMemory::IrisInstanceMemory ( IrisInstance * irisInstance = 0 )
```

Construct an IrisInstanceMemory.

Optionally attaches to an IrisInstance.
Parameters

| irisInstance | The IrisInstance to attach to. |

8.21.3 Member Function Documentation

8.21.3.1 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

<table>
<thead>
<tr>
<th>inSpaceId</th>
<th>Memory space id for the input memory space of this translation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>outSpaceId</td>
<td>Memory space id for the output memory space of this translation.</td>
</tr>
<tr>
<td>description</td>
<td>A human-readable description of this translation.</td>
</tr>
</tbody>
</table>

Returns

A reference to an `AddressTranslationInfoAndAccess` object for the new translation. This reference is valid until the next time `addAddressTranslation()` is called.

8.21.3.2 addMemorySpace()

```
SpaceInfoAndAccess iris::IrisInstanceMemory::addMemorySpace ( 
    const std::string & name )
```

Add meta information for one memory space.

Parameters

| name | 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.
8.21.3.3 attachTo()

void iris::IrisInstanceMemory::attachTo ( IrisInstance * irisInstance )

Attach this IrisInstance add-on to a specific IrisInstance.

Parameters

| irisInstance | The IrisInstance to attach to. |

8.21.3.4 setDefaultGetSidebandInfoDelegate()

void iris::IrisInstanceMemory::setDefaultGetSidebandInfoDelegate ( MemoryGetSidebandInfoDelegate delegate = MemoryGetSidebandInfoDelegate() ) [inline]

Set the default delegate to retrieve sideband information.

Parameters

| delegate | Delegate object which will be called to get sideband information for a memory space. |

8.21.3.5 setDefaultReadDelegate()

void iris::IrisInstanceMemory::setDefaultReadDelegate ( MemoryReadDelegate delegate = MemoryReadDelegate() ) [inline]

Set default read function for all subsequently added memory spaces.

Parameters

| delegate | Delegate object which will be called to read memory. |

8.21.3.6 setDefaultTranslateDelegate()

void iris::IrisInstanceMemory::setDefaultTranslateDelegate ( MemoryAddressTranslateDelegate delegate = MemoryAddressTranslateDelegate() ) [inline]

Set the default memory translation delegate.
Parameters

| delegate | Delegate object which will be called to translate addresses. |

8.21.3.7 `setDefaultWriteDelegate()`

```cpp
template<>
void iris::IrisInstanceMemory::setDefaultWriteDelegate (MemoryWriteDelegate delegate = MemoryWriteDelegate()) [inline]
```

Set default write function for all subsequently added memory spaces.

Parameters

| delegate | Delegate object which will be called to write memory. |

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

- IrisInstanceMemory.h

8.22 iris::IrisInstancePerInstanceExecution Class Reference

Per-instance execution control add-on for IrisInstance.

```cpp
#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.

8.22.1 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.
8.22 Constructor & Destructor Documentation

8.22.1 IrisInstancePerInstanceExecution()

iris::IrisInstancePerInstanceExecution::IrisInstancePerInstanceExecution ( IrisInstance * irisInstance = nullptr )

Construct an IrisInstancePerInstanceExecution add-on.

Parameters

| irisInstance | The IrisInstance to attach this add-on to. |

8.22.3 Member Function Documentation

8.22.3.1 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

| irisInstance | The IrisInstance to attach this add-on to. |

8.22.3.2 setExecutionStateGetDelegate()

void iris::IrisInstancePerInstanceExecution::setExecutionStateGetDelegate ( PerInstanceExecutionStateGetDelegate delegate )

Set the delegate for getting execution state.

Parameters

| delegate | A delegate object which will be called to get the current execution state for the attached instance. |
8.22.3.3 setExecutionStateSetDelegate()

void iris::IrisInstancePerInstanceExecution::setExecutionStateSetDelegate (PerInstanceExecutionStateSetDelegate delegate)

Set the delegate for setting execution state.

Parameters

| delegate | 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

8.23 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 rscId)
  
  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 rscId, const std::string &tag)
  
  Set a tag for a specific resource.
Static Public Member Functions

- 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_read** (IrisReceivedRequest &request)
- void **impl_resource_write** (IrisReceivedRequest &request)

8.23.1 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.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 IrisInstanceResource()

iris::IrisInstanceResource::IrisInstanceResource ( IrisInstance * irisInstance = 0 )

Construct an IrisInstanceResource.

Optionally attaches to an IrisInstance.

Parameters

| **irisInstance** | The IrisInstance to attach to. |

8.23.3 Member Function Documentation
8.23.3.1 addResource()

```
ResourceInfoAndAccess& iris::IrisInstance::addResource (  
    const std::string & type,  
    const std::string & name,  
    const std::string & description )
```

Add a new resource.

**Parameters**

<table>
<thead>
<tr>
<th>type</th>
<th>The type of the resource. This should be one of:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;numeric&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;numericFp&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;String&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;noValue&quot;</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>The name of the resource.</td>
</tr>
<tr>
<td>description</td>
<td>A human-readable description of the resource.</td>
</tr>
</tbody>
</table>

**Returns**

A reference to a `ResourceInfoAndAccess` object for this new resource. This reference is valid until the next time `addResource()` is called.

8.23.3.2 attachTo()

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

Attach this `IrisInstance` add-on to a specific `IrisInstance`.

**Parameters**

| irisInstance | The `IrisInstance` to attach to. |

8.23.3.3 beginResourceGroup()

```
void iris::IrisInstance::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**

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the resource group.</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A description of this resource group.</td>
</tr>
<tr>
<td>subRscIdStart</td>
<td>If not IRIS_UINT64_MAX start counting from this subRscId when new resources are added.</td>
</tr>
<tr>
<td>cname</td>
<td>A C identifier version of the resource name if different from <code>name</code>.</td>
</tr>
</tbody>
</table>

### 8.23.3.4 getResourceInfo()

```cpp
ResourceInfoAndAccess* iris::IrisInstanceResource::getResourceInfo ( 
    ResourceId rscId )
```

Get the resource info for a resource that was already added.

**Parameters**

| rscId | 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`.

### 8.23.3.5 makeNamesHierarchical()

```cpp
static void iris::IrisInstanceResource::makeNamesHierarchical ( 
    std::vector< ResourceInfo > & resourceInfos ) [static]
```

Make name and `cname` of RegisterInfos hierarchical.

Modify name and `cname` in `resourceInfos` such that a child register X of parent FLAGS gets name=FLAGS.X and `CNAME=FLAGS_X`. This functionality is not an Iris interface but just a convenience function for simple clients.

The name and `cname` attributes of all child resources are modified. No attributes are modified for non-child resources.

This is a static member function and does not modify any state in any `IrisInstance`. It is usually only useful to simplistic clients that use hierarchical names.

Note that no errors are generated for missing parent resources. `parentRscId` links to missing parent resources are ignored. The intended usage is to call this function on a list containing all resources or all registers of an instance.
**Parameters**

| resourceInfos | Array of all ResourceInfos of an instance |

8.23.3.6 setNextSubRscId()

```cpp
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**

| nextSubRscId_ | Next subRscId |

8.23.3.7 setTag()

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

Set a tag for a specific resource.

**Parameters**

| rscId_ | Resource Id for the resource that will have this tag set. |
| tag | 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
8.24 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_CA←
  LL_EXTENSION event.

- **IrisInstanceSemihosting (IrisInstance *iris_instance=nullptr, IrisInstanceEvent *inst_event=nullptr)**
  
  Read data for a given file descriptor.

- **std::vector< uint8_t > readData (uint64_t fDes, size_t max_size=0, uint64_t flags=semihost::DEFAULT)**

  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, size_t size)**

8.24.1 Member Function Documentation

8.24.1.1 attachTo()

```cpp
void iris::IrisInstanceSemihosting::attachTo ( 
    IrisInstance * iris_instance )
```

Attach this IrisInstance add-on to a specific IrisInstance.

**Parameters**

- **iris_instance** The instance to attach to.

8.24.1.2 readData()

```cpp
std::vector< uint8_t > iris::IrisInstanceSemihosting::readData ( 
    uint64_t fDes, 
    size_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 method blocks. If max_size is not zero, then at most max_size bytes will be returned.
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fDes</td>
<td>File descriptor to read from. Usually semihost::STDIN.</td>
</tr>
<tr>
<td>max_size</td>
<td>The maximum amount of bytes to read or zero for no limit.</td>
</tr>
<tr>
<td>flags</td>
<td>A bitwise OR of Semihosting data request flag constants</td>
</tr>
</tbody>
</table>

Returns

A vector of data that was read.

8.24.1.3 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation</td>
<td>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.</td>
</tr>
<tr>
<td>parameter</td>
<td>A parameter to the operation. This meaning of this parameter is defined by the operation.</td>
</tr>
</tbody>
</table>

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.

8.24.1.4 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler</td>
<td>The event add-on for this Iris instance.</td>
</tr>
</tbody>
</table>
An IrisInstance add-on that adds simulation functions for the SimulationEngine instance.

```cpp
#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 setConnectionInterface (IrisConnectionInterface *connection_interface_)**
  
  Set the IrisConnectionInterface to use for the instantiation.

- **void setEventHandler (IrisInstanceEvent *handler)**
  
  Set up IRIS_SIM_PHASE+ events.

- **void setGetParameterInfoDelegate (SimulationGetParameterInfoDelegate delegate, bool cache_result=true)**
  
  Set the getParameterInfo() delegate.

- **void setGetParameterInfoDelegate (T *instance, bool cache_result=true)**
  
  Set the getParameterInfo() delegate.

- **void setInstantiateDelegate (SimulationInstantiateDelegate delegate)**
  
  Set the instantiate() delegate.

- **void setRequestShutdownDelegate (SimulationRequestShutdownDelegate delegate)**
  
  Set the requestShutdown() delegate.
• void setResetDelegate (SimulationResetDelegate delegate)
  Set the reset() delegate.

• template<typename T, IrisErrorCode(T::*)(const IrisSimulationResetContext &)> void setResetDelegate (T *instance)
  Set the reset() delegate.

• template<IrisErrorCode(*)(const IrisSimulationResetContext &)> void setResetDelegate ()
  Set the reset() 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.

• template<IrisErrorCode(*)(const InstantiationParameterValue &)> void setSetParameterValueDelegate ()
  Set the setParameterValue() delegate.

8.25.1 Detailed Description

An IrisInstance add-on that adds simulation functions for the SimulationEngine instance.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 IrisInstanceSimulation()

iris::IrisInstanceSimulation::IrisInstanceSimulation ( IrisInstance * iris_instance = nullptr, IrisConnectionInterface * connection_interface = nullptr )

Construct an IrisInstanceSimulation add-on.

Parameters

<table>
<thead>
<tr>
<th>iris_instance</th>
<th>The IrisInstance to attach this add-on to.</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection_interface</td>
<td>The connection interface that will be used when the simulation is instantiated.</td>
</tr>
</tbody>
</table>

8.25.3 Member Function Documentation

8.25.3.1 attachTo()

void iris::IrisInstanceSimulation::attachTo ( IrisInstance * iris_instance )

Attach this IrisInstance add-on to a specific IrisInstance.
Parameters

| **iris_instance** | The IrisInstance to attach to. |

### 8.25.3.2 enterPostInstantiationPhase()

```cpp
def 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().

### 8.25.3.3 notifySimPhase()

```cpp
def notifySimPhase(time: uint64_t, phase: IrisSimulationPhase)
```

Emit an IRIS_SIM_PHASE* event for the supplied phase.

**Parameters**

| **time** | The simulation time at which the event occurred. |
| **phase** | The simulation phase that was reached. |

### 8.25.3.4 setConnectionInterface()

```cpp
def setConnectionInterface(connection_interface: IrisConnectionInterface)
```

Set the IrisConnectionInterface to use for the instantiation.

This will be passed to the instantiate() delegate when the simulation is instantiated.

### 8.25.3.5 setEventHandler()

```cpp
def setEventHandler(handler: IrisInstanceEvent)
```

Set up IRIS_SIM_PHASE* events.
Parameters

| handler | An IrisInstanceEvent add-on that is attached to the same instance as this add-on. |

8.25.3.6 setGetParameterInfoDelegate() [1/3]

```cpp
void iris::IrisInstanceSimulation::setGetParameterInfoDelegate |
    SimulationGetParameterInfoDelegate delegate, |
    bool cache_result = true } [inline]
```

Set the getParameterInfo() delegate.

Parameters

| delegate | A delegate object that is called to get instantiation parameter information for the simulation. |
| cache_result | If true, the delegate is only called once and the result is cached and used for subsequent calls to simulation_getInstantiationParameterInfo(). If false, the result is not cached and the delegate is called every time. |

8.25.3.7 setGetParameterInfoDelegate() [2/3]

```cpp
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

| T | Class that defines a getParameterInfo delegate method. |
| METHOD | A method of class T that is a getParameterInfo delegate. |

Parameters

| instance | An instance of class T on which METHOD should be called. |
| cache_result | If true, the delegate is called once and the result is cached and used for subsequent calls to simulation_getInstantiationParameterInfo(). If false, the result is not cached and the delegate is called every time. |
8.25.3.8 "setGetParameterInfoDelegate()"

```cpp
// Template parameters
<
IrisErrorCode(*)(std::vector<ResourceInfo>&) FUNC>

// Function definition
void iris::IrisInstanceSimulation::setGetParameterInfoDelegate(
    bool cache_result = true) [inline]
```

Set the getParameterInfo() delegate.

Set the delegate to a global function.

**Template Parameters**

| FUNC | A function that is a getParameterInfo delegate. |

**Parameters**

| cache_result | If true, the delegate is only called once and the result is cached and used for subsequent calls to simulation_getInstantiationParameterInfo(). If false, the result is not cached and the delegate is called every time. |

8.25.3.9 "setInstantiateDelegate()"

```cpp
void iris::IrisInstanceSimulation::setInstantiateDelegate(
    SimulationInstantiateDelegate delegate) [inline]
```

Set the instantiate() delegate.

**Parameters**

| delegate | A delegate object that will be called to instantiate the simulation. |

8.25.3.10 "setInstantiateDelegate()"

```cpp
// Template parameters
<
typename T, IrisErrorCode(T::*)(InstantiationResult&) METHOD>

// Function definition
void iris::IrisInstanceSimulation::setInstantiateDelegate(
    T* instance) [inline]
```

Set the instantiate() delegate.

Set the delegate to call a method in class T.

**Template Parameters**

| T | Class that defines an instantiate delegate method. |

| METHOD | A method of class T that is an instantiate delegate. |
Parameters

instance | An instance of class \textit{T} on which \textit{METHOD} should be called.

8.25.3.11  
\textbf{setInstantiateDelegate()}  [3/3]

\begin{verbatim}
template<\textit{IrisErrorCode}(\textit{\ast})(\textit{InstantiationResult} &) \textit{FUNC}>
void iris::IrisInstanceSimulation::setInstantiateDelegate () [inline]
\end{verbatim}

Set the instantiate() delegate.

Set the delegate to a global function.

Template Parameters

\textbf{FUNC} | A function that is an instantiate delegate.

8.25.3.12  
\textbf{setRequestShutdownDelegate()}  [1/3]

\begin{verbatim}
void iris::IrisInstanceSimulation::setRequestShutdownDelegate ( SimulationRequestShutdownDelegate \textit{delegate} ) [inline]
\end{verbatim}

Set the requestShutdown() delegate.

Parameters

\textbf{delegate} | A delegate object that will be called to request that the simulation be shut down.

8.25.3.13  
\textbf{setRequestShutdownDelegate()}  [2/3]

\begin{verbatim}
template<\textit{typename \textit{T}}, \textit{IrisErrorCode}(\textit{T::\ast})() \textit{METHOD}>
void iris::IrisInstanceSimulation::setRequestShutdownDelegate ( 
        \textit{T} \ast \textit{instance} ) [inline]
\end{verbatim}

Set the requestShutdown() delegate.

Set the delegate to call a method in class \textit{T}.

Template Parameters

\begin{tabular}{|c|c|}
\hline
\textit{T} & Class that defines a requestShutdown delegate method. \\
\hline
\textit{METHOD} & A method of class \textit{T} that is a requestShutdown delegate. \\
\hline
\end{tabular}
Parameters

| instance | An instance of class T on which METHOD should be called. |

8.25.3.14  setRequestShutdownDelegate() [3/3]

```cpp
template<iris::IrisErrorCode(iris::IrisError(l::IrisError)::*FUNC)>
void iris::IrisInstanceSimulation::setRequestShutdownDelegate ( ) [inline]
```

Set the requestShutdown() delegate.

Set the delegate to a global function.

Template Parameters

| FUNC | A function that is a requestShutdown delegate. |

8.25.3.15  setResetDelegate() [1/3]

```cpp
void iris::IrisInstanceSimulation::setResetDelegate (simulationResetDelegate delegate ) [inline]
```

Set the reset() delegate.

Parameters

| delegate | A delegate object which will be called to reset the simulation. |

8.25.3.16  setResetDelegate() [2/3]

```cpp
template<typename T , iris::IrisErrorCode(T::*METHOD)>
void iris::IrisInstanceSimulation::setResetDelegate (T * instance ) [inline]
```

Set the reset() delegate.

Set the delegate to call a method in class T.

Template Parameters

| T | Class that defines a reset delegate method. |
| METHOD | A method of class T that is a reset delegate. |
8.25.3.17 setResetDelegate() [3/3]

```
template<IrisErrorCode(*)(const IrisSimulationResetContext &)> FUNC>
void iris::IrisInstanceSimulation::setResetDelegate ( ) [inline]
```

Set the reset() delegate.

Set the delegate to a global function.

**Template Parameters**

- **FUNC** A function that is a reset delegate.

8.25.3.18 setSetParameterValueDelegate() [1/3]

```
void iris::IrisInstanceSimulation::setSetParameterValueDelegate ( SimulationSetParameterValueDelegate delegate ) [inline]
```

Set the setParameterValue() delegate.

**Parameters**

- **delegate** A delegate object that is called to set instantiation parameter values before instantiation.

8.25.3.19 setSetParameterValueDelegate() [2/3]

```
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**

- **T** Class that defines a setParameterValue delegate method.
- **METHOD** A method of class T that is a setParameterValue delegate.
Parameters

| instance | An instance of class T on which METHOD should be called. |

8.25.3.20  

**setSetParameterValueDelegate() [3/3]**

```cpp
template<typename IrisErrorCode(T::*)(uint64_t &, uint64_t &, bool &), IrisErrorCode((const InstantiationParameterValue &)* FUNC>)
void iris::IrisInstanceSimulationTime::setSetParameterValueDelegate ( ) [inline]
```

Set the setParameterValue() delegate.

Set the delegate to a global function.

Template Parameters

| FUNC | A function that is a setParameterValue delegate. |

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

- IrisInstanceSimulation.h

8.26  

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** (TIME_EVENT_REASON reason=TIME_EVENT_UNKNOWN)
  
  *Generate the IRIS_SIMULATION_TIME_EVENT event callback.*

- void **setEventHandler** (IrisInstanceEvent *handler)
  
  *Set the event handler to use to send simulation time-related events.*

- 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.*

- template<IrisErrorCode((const InstantiationParameterValue &)* FUNC>)
  
  void **setSimTimeGetDelegate** ()

  *Set the getTime() delegate.*
• void setSimTimeRunDelegate (SimulationTimeRunDelegate delegate)
  
  * Set the run() delegate.

• template<typename T, IrisErrorCode(T::*)(METHOD)>
  void setSimTimeRunDelegate (T *instance)

  * Set the run() delegate.

• template<typename T, IrisErrorCode(T::*)(FUNC)>
  void setSimTimeRunDelegate ()

  * Set the run() delegate.

• void setSimTimeStopDelegate (SimulationTimeStopDelegate delegate)
  
  * Set the stop() delegate.

• template<typename T, IrisErrorCode(T::*)(METHOD)>
  void setSimTimeStopDelegate (T *instance)

  * Set the stop() delegate.

• template<typename T, IrisErrorCode(T::*)(FUNC)>
  void setSimTimeStopDelegate ()

  * Set the stop() delegate.

### 8.26.1 Detailed Description

Simulation time add-on for IrisInstance.

### 8.26.2 Constructor & Destructor Documentation

#### 8.26.2.1 IrisInstanceSimulationTime()

```cpp
iris::IrisInstanceSimulationTime::IrisInstanceSimulationTime ( IrisInstance * iris_instance = nullptr,
                                                             IrisInstanceEvent * inst_event = nullptr )
```

Construct an IrisInstanceSimulationTime add-on.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iris_instance</td>
<td>An IrisInstance to attach this add-on to.</td>
</tr>
<tr>
<td>inst_event</td>
<td>An IrisInstanceEvent add-on that is already attached to IrisInstance. This is used to set up simulation time events.</td>
</tr>
</tbody>
</table>

### 8.26.3 Member Function Documentation

#### 8.26.3.1 attachTo()

```cpp
void iris::IrisInstanceSimulationTime::attachTo ( IrisInstance * irisInstance )
```
Attach this `IrisInstance` add-on to a specific `IrisInstance`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>irisInstance</code></td>
<td>An <code>IrisInstance</code> to attach this add-on to.</td>
</tr>
</tbody>
</table>

### 8.26.3.2 `setEventHandler()`

```cpp
void iris::IrisInstanceSimulationTime::setEventHandler ( IrisInstanceEvent * handler )
```

Set the event handler to use to send simulation time-related events.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>handler</code></td>
<td>An <code>IrisInstanceEvent</code> add-on that is already attached to <code>IrisInstance</code>. This is used to set up simulation time events.</td>
</tr>
</tbody>
</table>

### 8.26.3.3 `setSimTimeGetDelegate()` [1/3]

```cpp
void iris::IrisInstanceSimulationTime::setSimTimeGetDelegate ( SimulationTimeGetDelegate delegate ) [inline]
```

Set the `getTime()` delegate.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>delegate</code></td>
<td>A delegate that is called to get the current simulation time.</td>
</tr>
</tbody>
</table>

### 8.26.3.4 `setSimTimeGetDelegate()` [2/3]

```cpp
template<typename T , IrisErrorCode(T::*)(uint64_t &, uint64_t &, bool &) METHOD>
void iris::IrisInstanceSimulationTime::setSimTimeGetDelegate ( T * instance ) [inline]
```

Set the `getTime()` delegate.

**Template Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T</code></td>
<td>Class that defines a <code>getTime</code> delegate method.</td>
</tr>
<tr>
<td><code>METHOD</code></td>
<td>A method of class <code>T</code> that is a <code>getTime</code> delegate.</td>
</tr>
</tbody>
</table>
Parameters

| instance | An instance of class T on which METHOD should be called. |

8.26.3.5 setSimTimeGetDelegate() [3/3]

```cpp
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

| FUNC | A function that is a getTime delegate. |

8.26.3.6 setSimTimeRunDelegate() [1/3]

```cpp
void iris::IrisInstanceSimulationTime::setSimTimeRunDelegate (SimulationTimeRunDelegate delegate ) [inline]
```

Set the run() delegate.

Parameters

| delegate | A delegate that is called to start/resume progress of simulation time. |

8.26.3.7 setSimTimeRunDelegate() [2/3]

```cpp
template<typename T , IrisErrorCode(T::*)() METHOD>
void iris::IrisInstanceSimulationTime::setSimTimeRunDelegate (T * instance ) [inline]
```

Set the run() delegate.

Template Parameters

| T | Class that defines a run delegate method. |
| METHOD | A method of class T that is a run delegate. |
Parameters

| instance | An instance of class $T$ on which $METHOD$ should be called. |

8.26.3.8 setSimTimeRunDelegate() [3/3]

```cpp
template<iris::ErrorCode(*)(*) FUNC>
void iris::IrisInstanceSimulationTime::setSimTimeRunDelegate(); [inline]
```

Set the run() delegate.

Set the delegate to a global function.

Template Parameters

| FUNC | A function that is a run delegate. |

8.26.3.9 setSimTimeStopDelegate() [1/3]

```cpp
void iris::IrisInstanceSimulationTime::setSimTimeStopDelegate(SimulationTimeStopDelegate delegate); [inline]
```

Set the stop() delegate.

Parameters

| delegate | A delegate that is called to stop the progress of simulation time. |

8.26.3.10 setSimTimeStopDelegate() [2/3]

```cpp
template<typename T, IrisErrorCode(T::*)(*) METHOD>
void iris::IrisInstanceSimulationTime::setSimTimeStopDelegate(T * instance); [inline]
```

Set the stop() delegate.

Template Parameters

| T | Class that defines a stop delegate method. |
| METHOD | A method of class $T$ that is a stop delegate. |
Parameters

| instance | An instance of class T on which METHOD should be called. |

8.26.3.11 setSimTimeStopDelegate() [3/3]

```
template<
    IrisErrorCode(*)(())
>
void iris::IrisInstanceSimulationTime::setSimTimeStopDelegate() [inline]
```

Set the stop() delegate.

Set the delegate to a global function.

Template Parameters

| FUNC | A function that is a stop delegate. |

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

- IrisInstanceSimulationTime.h

8.27 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.

8.27.1 Detailed Description

Step add-on for IrisInstance.

This is used by instances to support stepping functionality.

This class implements all Iris step*() functions.
8.27.2 Constructor & Destructor Documentation

8.27.2.1 IrisInstanceStep()

```cpp
iris::IrisInstanceStep::IrisInstanceStep ( 
    IrisInstance * irisInstance = nullptr )
```

Construct an IrisInstanceStep add-on.

Parameters

| irisInstance | The IrisInstance to attach this add-on to. |

8.27.3 Member Function Documentation

8.27.3.1 attachTo()

```cpp
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

| irisInstance | The IrisInstance to attach this add-on to. |

8.27.3.2 setRemainingStepGetDelegate()

```cpp
void iris::IrisInstanceStep::setRemainingStepGetDelegate ( 
    RemainingStepGetDelegate delegate )
```

Set the delegate for getting the remaining steps.

Parameters

| delegate | A delegate object that is called to get the remaining steps for the attached instance. |
8.27.3.3 setRemainingStepSetDelegate()

```cpp
void iris::IrisInstanceStep::setRemainingStepSetDelegate (
    RemainingStepSetDelegate delegate)
```

Set the delegate for setting the remaining steps.

Parameters

| delegate | A delegate object that is called to set the remaining steps for the attached instance. |

8.27.3.4 setStepCountGetDelegate()

```cpp
void iris::IrisInstanceStep::setStepCountGetDelegate (
    StepCountGetDelegate delegate)
```

Set the delegate for getting the step count.

Parameters

| delegate | 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

8.28 iris::IrisInstanceTable Class Reference

Table add-on for IrisInstance.

```cpp
#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.
8.28.1 Detailed Description

Table add-on for IrisInstance.

This is used by instances to support table functionality.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 IrisInstanceTable()

iris::IrisInstanceTable::IrisInstanceTable ( IrisInstance ∗ irisInstance = nullptr )

Construct an IrisInstanceTable add-on.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irisInstance</td>
<td>The IrisInstance to attach this add-on to.</td>
</tr>
</tbody>
</table>

8.28.3 Member Function Documentation

8.28.3.1 addTableInfo()

TableInfoAndAccess& iris::IrisInstanceTable::addTableInfo ( const std::string & name )

Add metadata for one table.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of this table.</td>
</tr>
</tbody>
</table>

Returns

A reference to a TableInfoAndAccess object that can be used to set metadata and access delegates for this table.

8.28.3.2 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irisInstance</td>
<td>The IrisInstance to attach this add-on to.</td>
</tr>
</tbody>
</table>

8.28.3.3 setDefaultReadDelegate()

```cpp
void iris::IrisInstanceTable::setDefaultReadDelegate (  
    TableReadDelegate delegate = TableReadDelegate() )  
    [inline]
```

Set the default delegate for reading table data.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delegate</td>
<td>A delegate object that is called to read table data for tables in the attached instance that did not set a table-specific delegate.</td>
</tr>
</tbody>
</table>

8.28.3.4 setDefaultWriteDelegate()

```cpp
void iris::IrisInstanceTable::setDefaultWriteDelegate (  
    TableWriteDelegate delegate = TableWriteDelegate() )  
    [inline]
```

Set the default delegate for writing table data.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delegate</td>
<td>A delegate object that is called to write table data for tables in the attached instance that did not set a table-specific delegate.</td>
</tr>
</tbody>
</table>

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

- IrisInstanceTable.h

8.29 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.
- 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.
- template<typename T>
  void getParameter (const std::string &name, T &value)
  Get the value of an instantiation parameter.
- void getParameter (const std::string &name, std::vector<uint64_t> &value)
  Get the value of a large numeric instantiation parameter.
- uint64_t getRecommendedInstanceFlags () const
  Get the flags to use when registering the instance being instantiated.
- IrisInstantiationContext * getSubcomponentContext (const std::string &child_name)
  Get an IrisInstanceContext pointer for a subcomponent instance.
- IrisInstantiationContext (IrisConnectionInterface *connection_interface_, InstantiationResult &result_,
  const std::vector<ResourceInfo> &param_info_, const std::vector<InstantiationParameterValue> &param_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 &parameterName, const char *format,...) INTERNAL_IRIS_PRINTF(4
  Add an error to the InstantiationResult.
- void void parameterWarning (const std::string &code, const std::string &parameterName, 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.

8.29.1 Detailed Description

Provides context when instantiating an Iris instance from a factory.

8.29.2 Member Function Documentation

8.29.2.1 error()

void void void iris::IrisInstantiationContext::error (const std::string &code, const char *format,...)

Add an error to the InstantiationResult.

See also

parameterError

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>code</th>
<th>An error code symbol. This should be one of the codes specified for the instantiation error object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>A printf-style format string.</td>
</tr>
<tr>
<td>...</td>
<td>Printf substitution arguments.</td>
</tr>
</tbody>
</table>

8.29.2.2 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()`.

8.29.2.3 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()`.

8.29.2.4 getParameter() [1/2]

template<typename T>
void iris::IrisInstantiationContext::getParameter ( 
    const std::string & name,
    T & value ) [inline]

Get the value of an instantiation parameter.

Template Parameters

| T     | The type of the `value`. This must be a type that is appropriate to receive the value of this parameter. |
8.29 iris::IrisInstantiationContext Class Reference

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>A reference to a value of type ( T ) that receives the value of the named parameter.</td>
</tr>
</tbody>
</table>

8.29.2.5 getParameter() [2/2]

```cpp
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

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>A reference to a value of type ( T ) that receives the value of the named parameter.</td>
</tr>
</tbody>
</table>

8.29.2.6 getRecommendedInstanceFlags()

```cpp
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().

8.29.2.7 getSubcomponentContext()

```cpp
IrisInstantiationContext* iris::IrisInstantiationContext::getSubcomponentContext (  
    const std::string & child_name )
```

Get an IrisInstantiationContext 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

| child_name | The name of a child instance. |

Returns

A pointer to an IrisInstantiationContext object for the named child.

8.29.2.8 parameterError()

```cpp
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

| code       | An error code symbol. This should be one of the codes specified for the InstantiationError object. |
| parameterName | The name of the parameter this error relates to. |
| format     | A printf-style format string. |
| ...        | Printf substitution arguments. |

8.29.2.9 parameterWarning()

```cpp
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

<table>
<thead>
<tr>
<th>code</th>
<th>An error code symbol. This should be one of the codes specified for the InstantiationError object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameterName</td>
<td>The name of the parameter this warning relates to.</td>
</tr>
<tr>
<td>format</td>
<td>A printf-style format string.</td>
</tr>
<tr>
<td>...</td>
<td>Print substitution arguments.</td>
</tr>
</tbody>
</table>

8.29.2.10 warning()

```cpp
void iris::IrisInstantiationContext::warning (  
    const std::string & code,  
    const char * format,  
    ... )
```

Add a warning to the InstantiationResult.

See also

`parameterWarning`

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

- IrisInstantiationContext.h

8.30 iris::IrisParameterBuilder Class Reference

Helper class to construct instantiation parameters.

```cpp
#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 (uint64_t value)
  Set the default value for a numeric parameter.
• IrisParameterBuilder & setDefault (const std::vector<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 (int64_t value)
  Set the default value for a numericSigned parameter.
• IrisParameterBuilder & setDefaultSigned (const std::vector<uint64_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 & setInitOnly (bool value=true)
  Set the initOnly field.
• IrisParameterBuilder & setMax (uint64_t max)
  Set the max field.
• IrisParameterBuilder & setMax (const std::vector<uint64_t> &max)
  Set the max field.
• IrisParameterBuilder & setMaxFloat (double max)
  Set the max field for floating-point parameters.
• IrisParameterBuilder & setMaxSigned (int64_t max)
  Set the max field.
• IrisParameterBuilder & setMaxSigned (const std::vector<uint64_t> &max)
  Set the max field.
• IrisParameterBuilder & setMin (uint64_t min)
  Set the min field.
• IrisParameterBuilder & setMin (const std::vector<uint64_t> &min)
  Set the min field.
• IrisParameterBuilder & setMinFloat (double min)
  Set the min field for floating-point parameters.
• IrisParameterBuilder & setMinSigned (int64_t min)
  Set the min field.
• IrisParameterBuilder & setMinSigned (const std::vector<uint64_t> &min)
  Set the min field.
• IrisParameterBuilder & setName (const std::string &name)
  Set the name field.
• IrisParameterBuilder & setRange (uint64_t min, uint64_t max)
  Set both the min field and the max 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 & setRangeFloat (double min, double 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.
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 & 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.

### 8.30.1 Detailed Description

Helper class to construct instantiation parameters.

### 8.30.2 Constructor & Destructor Documentation

#### 8.30.2.1 IrisParameterBuilder()

iris::IrisParameterBuilder::IrisParameterBuilder ( ResourceInfo & info_ ) [inline]

Construct a parameter builder for a given parameter resource.

Parameters

- info_ The resource info object for the parameter being built.

### 8.30.3 Member Function Documentation

#### 8.30.3.1 addEnum()

IrisParameterBuilder & iris::IrisParameterBuilder::addEnum ( const std::string & symbol,
Add an enum symbol for this parameter.

Parameters

<table>
<thead>
<tr>
<th>symbol</th>
<th>The enum symbol that is being added.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value associated with the symbol.</td>
</tr>
<tr>
<td>description</td>
<td>A description explaining the meaning of the symbol.</td>
</tr>
</tbody>
</table>

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.2 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

<table>
<thead>
<tr>
<th>value</th>
<th>The value associated with the symbol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A description explaining the meaning of the symbol.</td>
</tr>
</tbody>
</table>

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.3 setBitWidth()

IrisParameterBuilder& iris::IrisParameterBuilder::setBitWidth ( 
    uint64_t bitWidth ) [inline]

Set the bitWidth field.

Parameters

| bitWidth | The bitWidth field of the ResourceInfo object. |
Returns
A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.4 setDefault() [1/3]

IrisParameterBuilder& iris::IrisParameterBuilder::setDefault (const std::string & value) [inline]

Set the default value for a string parameter.

Parameters

| value | The defaultString field of the ParameterInfo object. |

Returns
A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.5 setDefault() [2/3]

IrisParameterBuilder& iris::IrisParameterBuilder::setDefault (uint64_t value) [inline]

Set the default value for a numeric parameter.

Parameters

| value | The defaultData field of the ParameterInfo object. |

Returns
A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.6 setDefault() [3/3]

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 \( \geq 2^{64} \).
Parameters

| value | The defaultData field of the ParameterInfo object |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

### 8.30.3.7 setDefaultFloat()

```
iris::IrisParameterBuilder& iris::IrisParameterBuilder::setDefaultFloat ( double value ) [inline]
```

Set the default value for a numericFp parameter.

Parameters

| value | The defaultData field of the ParameterInfo object |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

### 8.30.3.8 setDefaultSigned() [1/2]

```
iris::IrisParameterBuilder& iris::IrisParameterBuilder::setDefaultSigned ( int64_t value ) [inline]
```

Set the default value for a numericSigned parameter.

Parameters

| value | The defaultData field of the ParameterInfo object |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

### 8.30.3.9 setDefaultSigned() [2/2]

```
iris::IrisParameterBuilder& iris::IrisParameterBuilder::setDefaultSigned ( const std::vector<uint64_t> & value ) [inline]
```

Generated by Doxygen
Set the default value for a numericSigned parameter.

Use this variant for values that are out of range for int64_t.

Parameters

| value | The defaultData field of the ParameterInfo object. |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

---

### 8.30.3.10 setDescr()

IrisParameterBuilder& iris::IrisParameterBuilder::setDescr (const std::string & description) [inline]

Set the description field.

Parameters

| description | The description field of the ResourceInfo object. |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

---

### 8.30.3.11 setFormat()

IrisParameterBuilder& iris::IrisParameterBuilder::setFormat (const std::string & format) [inline]

Set the format field.

Parameters

| format | The format field of the ResourceInfo object. |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.
8.30.12 setInitOnly()  

IrisParameterBuilder& iris::IrisParameterBuilder::setInitOnly (  
  bool value = true ) [inline]  

Set the initOnly field.

Parameters  

| value | The initOnly field of the ParameterInfo object. |

Returns  

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.13 setMax() [1/2]  

IrisParameterBuilder& iris::IrisParameterBuilder::setMax (  
  uint64_t max ) [inline]  

Set the max field.

Parameters  

| max | The max field of the ParameterInfo object. |

Returns  

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.14 setMax() [2/2]  

IrisParameterBuilder& iris::IrisParameterBuilder::setMax (  
  const std::vector< uint64_t >& max ) [inline]  

Set the max field.

Use this variant to set values that are >= 2**64.

Parameters  

| max | The max field of the ParameterInfo object. |
Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.15 \texttt{setMaxFloat()}

\begin{verbatim}
IrisParameterBuilder& iris::IrisParameterBuilder::setMaxFloat ( double max ) [inline]
\end{verbatim}

Set the \texttt{max} field for floating-point parameters.

This implies that the parameter type is "numericFp".

Parameters

\begin{verbatim}
max | The max field of the ParameterInfo object.
\end{verbatim}

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.16 \texttt{setMaxSigned() [1/2]}

\begin{verbatim}
IrisParameterBuilder& iris::IrisParameterBuilder::setMaxSigned ( int64_t max ) [inline]
\end{verbatim}

Set the \texttt{max} field.

This implies that the parameter type is "numericSigned".

Parameters

\begin{verbatim}
max | The max field of the ParameterInfo object.
\end{verbatim}

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.17 \texttt{setMaxSigned() [2/2]}

\begin{verbatim}
IrisParameterBuilder& iris::IrisParameterBuilder::setMaxSigned ( const std::vector<uint64_t>& max ) [inline]
\end{verbatim}
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

- max
  The max field of the ParameterInfo object.

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.18 setMin() [1/2]

IrisParameterBuilder& iris::IrisParameterBuilder::setMin ( uint64_t min ) [inline]

Set the min field.

Parameters

- min
  The min field of the ParameterInfo object.

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.19 setMin() [2/2]

IrisParameterBuilder& iris::IrisParameterBuilder::setMin ( const std::vector<uint64_t>& min ) [inline]

Set the min field.

Use this variant to set values that are $\geq 2^{64}$.

Parameters

- min
  The min field of the ParameterInfo object.

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.
8.30.3.20 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

| min | The min field of the ParameterInfo object. |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.21 setMinSigned() [1/2]

IrisParameterBuilder& iris::IrisParameterBuilder::setMinSigned ( int64_t min ) [inline]

Set the min field.

This implies that the parameter type is "numericSigned".

Parameters

| min | The min field of the ParameterInfo object. |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.22 setMinSigned() [2/2]

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

\textbf{\texttt{min}} \hspace{1cm} The \texttt{min} field of the ParameterInfo object.

Returns

A reference to this \texttt{IrisParameterBuilder} object allowing calls to be chained together.

\subsection{8.30.3.23 \texttt{setName()}}

\texttt{IrisParameterBuilder\& iris::IrisParameterBuilder::setName (const std::string &\texttt{name}) [inline]}

Set the \texttt{name} field.

Parameters

\textbf{\texttt{name}} \hspace{1cm} The \texttt{name} field of the ResourceInfo object.

Returns

A reference to this \texttt{IrisParameterBuilder} object allowing calls to be chained together.

\subsection{8.30.3.24 \texttt{setRange()} [1/2]}

\texttt{IrisParameterBuilder\& iris::IrisParameterBuilder::setRange (uint64_t \texttt{min},
uint64_t \texttt{max}) [inline]}

Set both the \texttt{min} field and the \texttt{max} field.

Parameters

\textbf{\texttt{min}} \hspace{1cm} The \texttt{min} field of the ParameterInfo object.
\textbf{\texttt{max}} \hspace{1cm} The \texttt{max} field of the ParameterInfo object.

Returns

A reference to this \texttt{IrisParameterBuilder} object allowing calls to be chained together.
8.30.3.25  setRange() [2/2]

    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 \( \geq 2^{64} \).

Parameters

<table>
<thead>
<tr>
<th>min</th>
<th>The min field of the ParameterInfo object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>The max field of the ParameterInfo object.</td>
</tr>
</tbody>
</table>

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.26  setRangeFloat() [1/2]

    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

<table>
<thead>
<tr>
<th>min</th>
<th>The min field of the ParameterInfo object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>The max field of the ParameterInfo object.</td>
</tr>
</tbody>
</table>

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.27  setRangeSigned() [1/2]

    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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>min</strong></td>
<td>The <code>min</code> field of the <code>ParameterInfo</code> object.</td>
</tr>
<tr>
<td><strong>max</strong></td>
<td>The <code>max</code> field of the <code>ParameterInfo</code> object.</td>
</tr>
</tbody>
</table>

Returns

A reference to this `IrisParameterBuilder` object allowing calls to be chained together.

8.30.3.28 `setRangeSigned()` [2/2]

```cpp
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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>min</strong></td>
<td>The <code>min</code> field of the <code>ParameterInfo</code> object.</td>
</tr>
<tr>
<td><strong>max</strong></td>
<td>The <code>max</code> field of the <code>ParameterInfo</code> object.</td>
</tr>
</tbody>
</table>

Returns

A reference to this `IrisParameterBuilder` object allowing calls to be chained together.

8.30.3.29 `setRwMode()`

```cpp
IrisParameterBuilder& iris::IrisParameterBuilder::setRwMode(
    const std::string& rwMode) [inline]
```

Set the `rwMode` field.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rwMode</strong></td>
<td>The <code>rwMode</code> field of the <code>ResourceInfo</code> object.</td>
</tr>
</tbody>
</table>

Returns

A reference to this `IrisParameterBuilder` object allowing calls to be chained together.
8.30.3.30  setSubRscId()

IrisParameterBuilder& iris::IrisParameterBuilder::setSubRscId (  
    uint64_t subRscId  ) [inline]

Set the subRscId field.

Parameters

| subRscId | The subRscId field of the ResourceInfo object. |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.31  setTag() [1/2]

IrisParameterBuilder& iris::IrisParameterBuilder::setTag (  
    const std::string & tag ) [inline]

Set a boolean tag for this parameter resource.

Parameters

| tag | The name of the tag to set. |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.32  setTag() [2/2]

IrisParameterBuilder& iris::IrisParameterBuilder::setTag (  
    const std::string & tag,  
    const IrisValue & value ) [inline]

Set a tag for this parameter resource.

Parameters

| tag | The name of the tag to set. |
| value | The value to set for this tag. |
Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.33 setTopology()

IrisParameterBuilder& iris::IrisParameterBuilder::setTopology (bool value = true) [inline]

Set the topology field.

Parameters

| value | The topology field of the ParameterInfo object. |

Returns

A reference to this IrisParameterBuilder object allowing calls to be chained together.

8.30.3.34 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

| type | 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

8.31 iris::IrisPluginFactory < PLUGIN_INSTANCE > Class Template Reference

Public Member Functions

- IrisPluginFactory (IrisC_Functions *iris_c_functions, const std::string &plugin_name)
- IrisErrorCode unregisterInstance ()
Static Public Member Functions

- static int64_t initPlugin (IrisC_Functions ∗functions, const std::string &plugin_name)

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

- IrisPluginFactory.h

### 8.32 iris::IrisPluginFactoryBuilder Class Reference

Set metadata 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.

### 8.32.1 Detailed Description

Set metadata for instantiating a plug-in instance.

### 8.32.2 Constructor & Destructor Documentation

#### 8.32.2.1 IrisPluginFactoryBuilder()

iris::IrisPluginFactoryBuilder::IrisPluginFactoryBuilder ( const std::string & name ) [inline]
Parameters

| name | The name of the plug-in to build. |

### 8.32.3 Member Function Documentation

#### 8.32.3.1 `getDefaultInstanceName()`

```cpp
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.

#### 8.32.3.2 `getInstanceNamePrefix()`

```cpp
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.

#### 8.32.3.3 `getPluginName()`

```cpp
const std::string& iris::IrisPluginFactoryBuilder::getPluginName ( ) const [inline]
```

Get the plug-in name.

**Returns**

The name of the plug-in.

#### 8.32.3.4 `setDefaultInstanceName()`

```cpp
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.
Class Documentation

Parameters

| name | The default name for plug-in instances. |

### 8.32.3.5 setInstanceNamePrefix()

```cpp
def 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

| prefix | The prefix that will be used for instances of this plug-in. |

### 8.32.3.6 setPluginName()

```cpp
def 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

| name | The name of the plug-in. |

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

- IrisPluginFactory.h

### 8.33 iris::IrisRegisterReadEventEmitter< REG_T, ARGS > Class Template Reference

An EventEmitter class for register read events.

```cpp
#include <IrisRegisterEventEmitter.h>
```

Inherits IrisRegisterEventEmitterBase.
Public Member Functions

- void operator() (Resourceld rsclid, bool debug, REG_T value, ARGS... args)
  
  Emit an event.

8.33.1 Detailed Description

template<typename REG_T, typename... ARGS>

class iris::IrisRegisterReadEventEmitter< REG_T, ARGS >

An EventEmitter class for register read events.

Template Parameters

| REG_T | The type of the register being read. |
| ARGS  | 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;
}

8.33.2 Member Function Documentation

8.33.2.1 operator()()
Emit an event.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rscId</td>
<td>Resource id for the register that was accessed.</td>
</tr>
<tr>
<td>debug</td>
<td>True if this access originated from a debug access.</td>
</tr>
<tr>
<td>value</td>
<td>The register value that was read during this event.</td>
</tr>
<tr>
<td>args</td>
<td>Any additional custom fields for this event.</td>
</tr>
</tbody>
</table>

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

- IrisRegisterEventEmitter.h

8.34 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.

8.34.1 Detailed Description

\[
\text{template< typename REG}_T, \text{ typename... ARGS>}
\]

\[
\text{class iris::IrisRegisterUpdateEventEmitter< REG}_T, \text{ ARGS >}
\]

An EventEmitter class for register update events.

Template Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REG... _T</td>
<td>The type of the register being read.</td>
</tr>
<tr>
<td>ARGS</td>
<td>Types of any custom fields that this event source defines, in addition to the standard fields defined for register update events.</td>
</tr>
</tbody>
</table>

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->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);
}

8.34.2 Member Function Documentation

8.34.2.1 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rscId</td>
<td>Resource id for the register that was accessed.</td>
</tr>
<tr>
<td>debug</td>
<td>True if this access originated from a debug access.</td>
</tr>
<tr>
<td>old_value</td>
<td>The register value before the event.</td>
</tr>
<tr>
<td>new_value</td>
<td>The register value after the event.</td>
</tr>
<tr>
<td>args</td>
<td>Any additional custom fields for this event.</td>
</tr>
</tbody>
</table>

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

- IrisRegisterEventEmitter.h

8.35 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)

8.35.1 Detailed Description

Provides context to a reset delegate call.

8.35.2 Member Function Documentation

8.35.2.1 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

8.36 iris::IrisTcpClient Class Reference

Inherits IrisInterface, IrisProcessEventsInterface, and IrisConnectionInterface.
Public Member Functions

- IrisErrorCode **connect**(const std::string &hostname, uint16_t port, unsigned timeoutInMs, std::string &error)
- IrisErrorCode **disconnect**()
- impl::IrisRpcAdapterTcp::Format **getEffectiveSendingFormat** () const
  
  Get effective sending format that Rpc adapter uses.
- virtual IrisInterface * **getIrisInterface** () override
- IrisInterface * **getSendingInterface** ()
  
  Get interface for sending messages to the server.
- void **initServiceServer**(impl::IrisTcpSocket *socket_)
- IrisTcpClient ()
  
  Client constructor.
- IrisTcpClient (const service::IrisServiceTcpServer *)
  
  Service constructor to initialize IrisService Server on IrisService side.
- IrisTcpClient (const std::string &hostname, uint16_t port)
  
  Construct a connection to an Iris server.
- void **loadPlugin**(const std::string &plugin_name)
- virtual IrisErrorCode **processAsyncMessages**(bool waitForAMessage) override
- virtual void **processEvents** () override
- uint64_t **registerChannel**(IrisC_CommunicationChannel *channel)
- uint64_t **registerChannel**(IrisC_CommunicationChannel *channel, const ::std::string &path)
- virtual uint64_t **registerIrisInterfaceChannel**(IrisInterface *iris_interface) override
- void **setPreferredSendingFormat**(impl::IrisRpcAdapterTcp::Format p)
  
  Set preferred sending format that Rpc adapter uses.
- virtual void **stopWaitForEvent** () override
- void **unloadPlugin**()
- void **unregisterChannel**(uint64_t channelId)
- virtual void **unregisterIrisInterfaceChannel**(uint64_t channelId) override
- virtual void **waitForEvent** () override
- virtual ~IrisTcpClient ()
  
  Destructor.

8.36.1 Constructor & Destructor Documentation

8.36.1.1 IrisTcpClient()

```cpp
iris::IrisTcpClient::IrisTcpClient {
    const std::string & hostname,
    uint16_t port } [inline]
```

Construct a connection to an Iris server.
Parameters

<table>
<thead>
<tr>
<th>hostname</th>
<th>Hostname of the Iris server. This can be an IP address. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• &quot;192.168.0.5&quot; IP address of a different host.</td>
</tr>
<tr>
<td></td>
<td>• &quot;127.0.0.1&quot; Loopback IP address to connect to a server on the same machine.</td>
</tr>
<tr>
<td></td>
<td>• &quot;localhost&quot; Hostname of the loopback interface.</td>
</tr>
<tr>
<td></td>
<td>• &quot;foo.bar.com&quot; Hostname of a remote machine.</td>
</tr>
<tr>
<td>port</td>
<td>Server port number to connect to on the host.</td>
</tr>
</tbody>
</table>

8.36.2 Member Function Documentation

8.36.2.1 connect()

IrisErrorCode iris::IrisTcpClient::connect (  
    const std::string & hostname,  
    uint16_t port,  
    unsigned timeoutInMs,  
    std::string & errorResponseOut ) [inline]

Connect to server. Note that it is currently necessary to call processEvents() repeatedly in the background for connect() to succeed, because connect() calls instanceRegistry_registerInstance() using the usual mechanism.

8.36.2.2 disconnect()

IrisErrorCode iris::IrisTcpClient::disconnect ( ) [inline]

Disconnect from server. (Only for mode IRIS_TCP_CLIENT.)

8.36.2.3 initServiceServer()

void iris::IrisTcpClient::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.

8.36.2.4 loadPlugin()

void iris::IrisTcpClient::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
8.36.2.5 processEvents()

```cpp
virtual void iris::IrisTcpClient::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.

8.36.2.6 stopWaitForEvent()

```cpp
virtual void iris::IrisTcpClient::stopWaitForEvent() [inline], [override], [virtual]
```

Stop waiting in `waitForEvent()`. Return from `waitForEvent()` as soon as possible even without a socket event.

8.36.2.7 waitForEvent()

```cpp
virtual void iris::IrisTcpClient::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()`.

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

- IrisTcpClient.h

8.37 iris::IrisInstanceBuilder::MemorySpaceBuilder Class Reference

Used to set metadata for a memory space.

```cpp
#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_)**
  
  Set the default value for an attribute in the `attrib` field.

- **MemorySpaceBuilder & setCanonicalMsn (uint64_t canonicalMsn)**
  
  Set the `description` 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.

- **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 &)> 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 &)> MemorySpaceBuilder & setSidebandDelegate (T *instance)**
  
  Set the delegate to read sideband information.

- **template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector<std::string > &)> MemorySpaceBuilder & setSidebandDelegate ()**
  
  Set the delegate to read sideband information.

- **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 &)> MemorySpaceBuilder & setWriteDelegate (T *instance)**
  
  Set the delegate to write to this memory space.

- **template<IrisErrorCode(·)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *), MemoryWriteResult &)> MemorySpaceBuilder & setWriteDelegate ()**
  
  Set the delegate to write to this memory space.
8.37.1 Detailed Description

Used to set metadata for a memory space.

8.37.2 Member Function Documentation

8.37.2.1 addAttribute()

```cpp
MemorySpaceBuilder& iris::IrisInstanceBuilder::MemorySpaceBuilder::addAttribute (  
    const std::string & name,  
    AttributeInfo attrib )  [inline]
```

Add an attribute to the `attrib` field.

Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>The name of this attribute.</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrib</td>
<td>AttributeInfo for this attribute.</td>
</tr>
</tbody>
</table>

Returns

A reference to this `MemorySpaceBuilder` object allowing calls to be chained together.

8.37.2.2 getSpaceId()

```cpp
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.

8.37.2.3 setAttributeDefault()

```cpp
MemorySpaceBuilder& iris::IrisInstanceBuilder::MemorySpaceBuilder::setAttributeDefault (  
    const std::string & name,  
    IrisValue value )  [inline]
```

Set the default value for an attribute in the `attrib` field.
Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>The name of this attribute.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Default value of the named attribute.</td>
</tr>
</tbody>
</table>

Returns

A reference to this `MemorySpaceBuilder` object allowing calls to be chained together.

8.37.2.4 setCanonicalMsn()

```cpp
MemorySpaceBuilder& iris::IrisInstanceBuilder::MemorySpaceBuilder::setCanonicalMsn (uint64_t canonicalMsn) [inline]
```

Set the `description` field.

Parameters

| canonicalMsn | The canonicalMsn field of the MemorySpaceInfo object. |

Returns

A reference to this `MemorySpaceBuilder` object allowing calls to be chained together.

8.37.2.5 setDescription()

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

Set the `description` field.

Parameters

| description | The description field of the MemorySpaceInfo object. |

Returns

A reference to this `MemorySpaceBuilder` object allowing calls to be chained together.

8.37.2.6 setEndianness()

```cpp
MemorySpaceBuilder& iris::IrisInstanceBuilder::MemorySpaceBuilder::setEndianness (const std::string & endianness) [inline]
```
Set the **endianness** field.

**Parameters**

| endianness | The endianness field of the MemorySpaceInfo object. |

**Returns**

A reference to this `MemorySpaceBuilder` object allowing calls to be chained together.

### 8.37.2.7 setMaxAddr()

```cpp
MemorySpaceBuilder& iris::IrisInstanceBuilder::MemorySpaceBuilder::setMaxAddr ( uint64_t maxAddr ) [inline]
```

Set the **maxAddr** field.

**Parameters**

| maxAddr | The maxAddr field of the MemorySpaceInfo object. |

**Returns**

A reference to this `MemorySpaceBuilder` object allowing calls to be chained together.

### 8.37.2.8 setMinAddr()

```cpp
MemorySpaceBuilder& iris::IrisInstanceBuilder::MemorySpaceBuilder::setMinAddr ( uint64_t minAddr ) [inline]
```

Set the **minAddr** field.

**Parameters**

| minAddr | The minAddr field of the MemorySpaceInfo object. |

**Returns**

A reference to this `MemorySpaceBuilder` object allowing calls to be chained together.
8.37.2.9 setName()

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

Set the name field.

Parameters

| name | The name field of the MemorySpaceInfo object. |

Returns

A reference to this MemorySpaceBuilder object allowing calls to be chained together.

8.37.2.10 setReadDelegate()

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

| delegate | MemoryReadDelegate object. |

Returns

A reference to this MemorySpaceBuilder object allowing calls to be chained together.

8.37.2.11 setReadDelegate() [2/3]

template<
type, 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

<table>
<thead>
<tr>
<th>T</th>
<th>A class that defines a method with the right signature to be a memory read delegate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A memory read delegate method in class T.</td>
</tr>
</tbody>
</table>

Parameters

| instance | The instance of class T on which to call METHOD. |

Returns

A reference to this MemorySpaceBuilder object allowing calls to be chained together.

8.37.2.12 setReadDelegate() [3/3]

```cpp
template<
    IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &)
> 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

| FUNC | A memory read delegate function. |

Returns

A reference to this MemorySpaceBuilder object allowing calls to be chained together.

8.37.2.13 setSidebandDelegate() [1/3]

```cpp
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

| delegate | MemoryGetSidebandInfoDelegate object. |

Returns

A reference to this `MemorySpaceBuilder` object allowing calls to be chained together.

8.37.2.14  setSidebandDelegate() [2/3]

template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, 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

| T | A class that defines a method with the right signature to be a memory sideband information delegate. |
| METHOD | A memory sideband information delegate method in class T. |

Parameters

| instance | The instance of class T on which to call METHOD. |

Returns

A reference to this `MemorySpaceBuilder` object allowing calls to be chained together.

8.37.2.15  setSidebandDelegate() [3/3]

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

Generated by Doxygen
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

<table>
<thead>
<tr>
<th>FUNC</th>
<th>A memory sideband information delegate function.</th>
</tr>
</thead>
</table>

Returns

A reference to this MemorySpaceBuilder object allowing calls to be chained together.

8.37.2.16 setWriteDelegate() [1/3]

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

<table>
<thead>
<tr>
<th>delegate</th>
<th>MemoryWriteDelegate object.</th>
</tr>
</thead>
</table>

Returns

A reference to this MemorySpaceBuilder object allowing calls to be chained together.

8.37.2.17 setWriteDelegate() [2/3]

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

Generated by Doxygen
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

<table>
<thead>
<tr>
<th>T</th>
<th>A class that defines a method with the right signature to be a memory write delegate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A memory write delegate method in class T.</td>
</tr>
</tbody>
</table>

Parameters

instance The instance of class T on which to call METHOD.

Returns

A reference to this MemorySpaceBuilder object allowing calls to be chained together.

8.37.2.18 setWriteDelegate() [3/3]

template< IrisErrorCode( ) { const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, 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

| FUNC | A memory write delegate function. |

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
# 8.38 iris::IrisInstanceBuilder::ParameterBuilder Class Reference

Used to set metadata on a parameter.

```cpp
#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 getRscId()** const
  
  Return the rscId that was allocated for this resource.

- **ParameterBuilder & getRscId** (ResourceId &rscIdOut)
  
  Get the rscId that was allocated for this resource.

- **ParameterBuilder (IrisInstanceResource::ResourceInfoAndAccess &info_)**

- **ParameterBuilder & setBitWidth** (uint64_t bitWidth)
  
  Set the `bitWidth` field.

- **ParameterBuilder & setCname** (const std::string &cname)
  
  Set the `cname` field.

- **ParameterBuilder & setDefaultData** (uint64_t value)
  
  Set the default value for numeric parameter to a value <= 64 bit.

- **template<typename T> ParameterBuilder & setDefaultData** (std::initializer_list<T> &&t)
  
  Set the default value for wide numeric parameters.

- **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 & setDescr** (const std::string &description)
  
  Set the description field.

- **ParameterBuilder & setFormat** (const std::string &format)
  
  Set the format field.

- **ParameterBuilder & setInitOnly** (bool initOnly=true)
  
  Set the initOnly flag of a parameter.

- **ParameterBuilder & setMax** (uint64_t value)
  
  Set the max field to a value <= 64 bit.

- **template<typename T> ParameterBuilder & setMax** (std::initializer_list<T> &&t)
  
  Set the max field for wide numeric parameters.

- **template<typename Container> ParameterBuilder & setMaxFromContainer** (const Container &container)
  
  Set the max field for wide numeric parameters.

- **ParameterBuilder & setMin** (uint64_t value)
  
  Set the min field to a value <= 64 bit.

- **template<typename T> ParameterBuilder & setMin** (std::initializer_list<T> &&t)
Set the \texttt{min} field for wide numeric parameters.

- \texttt{template<typename Container> ParameterBuilder & setMinFromContainer (const Container &container)}
  
  Set the \texttt{min} field for wide numeric parameters.

- \texttt{ParameterBuilder & setName (const std::string &name)}
  
  Set the \texttt{name} field.

- \texttt{ParameterBuilder & setParentRscId (ResourceId parentRscId)}
  
  Set the \texttt{parentRscId} field.

- \texttt{ParameterBuilder & setReadDelegate (ResourceReadDelegate readDelegate)}
  
  Set the delegate to read the resource.

- \texttt{template<IrisErrorCode(*)(const ResourceInfo &, ResourceReadResult &) FUNC> ParameterBuilder & setReadDelegate ()}
  
  Set the delegate to read the resource.

- \texttt{template<typename T, IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD> ParameterBuilder & setReadDelegate (T *instance)}
  
  Set the delegate to read the resource.

- \texttt{ParameterBuilder & setRwMode (const std::string &rwMode)}
  
  Set the \texttt{rwMode} field.

- \texttt{ParameterBuilder & setSubRscId (uint64_t subRscId)}
  
  Set the \texttt{subRscId} field.

- \texttt{ParameterBuilder & setTag (const std::string &tag, const IrisValue &value)}
  
  Set a tag to the specified value.

- \texttt{ParameterBuilder & setTag (const std::string &tag)}
  
  Set the named boolean tag to true (e.g. \texttt{isPc})

- \texttt{ParameterBuilder & setType (const std::string &type)}
  
  Set the \texttt{type} field.

- \texttt{ParameterBuilder & setWriteDelegate (ResourceWriteDelegate writeDelegate)}
  
  Set the delegate to write the resource.

- \texttt{template<IrisErrorCode(*)(const ResourceInfo &, const ResourceWriteValue &) FUNC> ParameterBuilder & setWriteDelegate ()}
  
  Set the delegate to write the resource.

- \texttt{template<typename T, IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &) METHOD> ParameterBuilder & setWriteDelegate (T *instance)}
  
  Set the delegate to write the resource.

8.38.1 Detailed Description

Used to set metadata on a parameter.

8.38.2 Member Function Documentation
8.38.2.1 addEnum()

```cpp
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**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>The symbol string to be associated with the specified value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The value of this symbol.</td>
</tr>
<tr>
<td>Description</td>
<td>A description of this symbol.</td>
</tr>
</tbody>
</table>

**Returns**

A reference to this `ParameterBuilder` object allowing calls to be chained together.

8.38.2.2 addStringEnum()

```cpp
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**

<table>
<thead>
<tr>
<th>Value</th>
<th>The string value of this symbol. This is also used as the symbols string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>A description of this symbol.</td>
</tr>
</tbody>
</table>
Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

8.38.2.3 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.

8.38.2.4 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.

8.38.2.5 setBitWidth()

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

Set the bitWidth field.

Parameters

| bitWidth | The bitWidth field of the ResourceInfo object. |
### 8.38.2.6 setCname()

```
ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setCname (const std::string & cname) [inline]
```

Set the `cname` field.

#### Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cname</code></td>
<td>The <code>cname</code> field of the ResourceInfo object.</td>
</tr>
</tbody>
</table>

### Returns

A reference to this `ParameterBuilder` object allowing calls to be chained together.

### 8.38.2.7 setDefaultData() [1/2]

```
ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setDefaultData (uint64_t value) [inline]
```

Set the default value for numeric parameter to a value $\leq 64$ 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>value</code></td>
<td>The <code>defaultData</code> field of the ParameterInfo object.</td>
</tr>
</tbody>
</table>

### Returns

A reference to this `ParameterBuilder` object allowing calls to be chained together.
8.38.2.8  setDefaultData() [2/2]

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

| t | Braced initializer-list. |

Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

8.38.2.9  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

| container | Container containing the value in 64-bit chunks. |

### Returns

A reference to this `ParameterBuilder` object allowing calls to be chained together.

---

### 8.38.2.10 `setDefaultString()`

```cpp
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

| defaultString | The defaultString field of the ParameterInfo object. |

### Returns

A reference to this `ParameterBuilder` object allowing calls to be chained together.

---

### 8.38.2.11 `setDescr()`

```cpp
ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setDescr (
    const std::string & description ) [inline]
```

Set the `description` field.

### Parameters

| description | The description field of the ResourceInfo object. |

### Returns

A reference to this `ParameterBuilder` object allowing calls to be chained together.
8.38.2.12 setFormat()

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

Set the `format` field.

**Parameters**

| format | The format field of the ResourceInfo object. |

**Returns**

A reference to this `ParameterBuilder` object allowing calls to be chained together.

8.38.2.13 setInitOnly()

```cpp
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**

| initOnly | The initOnly flag of a parameter. |

**Returns**

A reference to this `ParameterBuilder` object allowing calls to be chained together.

8.38.2.14 setMax() [1/2]

```cpp
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.
### 8.38.2.15 setMax() [2/2]

```
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**

| `t` | Braced initializer-list. |

**Returns**

A reference to this `ParameterBuilder` object allowing calls to be chained together.

---

### 8.38.2.16 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

| container | Container containing the value in 64-bit chunks. |

### Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

---

### 8.38.2.17 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

| t | Braced initializer-list. |
Returns
A reference to this ParameterBuilder object allowing calls to be chained together.

8.38.2.18 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

| value | min value of the parameter |

Returns
A reference to this ParameterBuilder object allowing calls to be chained together.

8.38.2.19 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**

| container | Container containing the value in 64-bit chunks. |

**Returns**

A reference to this ParameterBuilder object allowing calls to be chained together.

### 8.38.2.20 setName()

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

Set the `name` field.

**Parameters**

| name | The name field of the ResourceInfo object. |

**Returns**

A reference to this ParameterBuilder object allowing calls to be chained together.

### 8.38.2.21 setParentRscId()

```cpp
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

| parent ← | RsclId | The rsclId of the parent register. |

Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

8.38.22  setReadDelegate()  [1/3]

```cpp
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

<table>
<thead>
<tr>
<th>T</th>
<th>A class that defines a method with the right signature to be a resource read delegate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A resource read delegate method in class T.</td>
</tr>
</tbody>
</table>
### Parameters

| instance | The instance of class T on which to call METHOD. |

### Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

---

#### 8.38.2.23 setReadDelegate() [2/3]

```cpp
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

| FUNC | A resource read delegate function. |

### Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

---

#### 8.38.2.24 setReadDelegate() [3/3]

```cpp
ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setReadDelegate ( ResourceReadDelegate readDelegate ) [inline]
```

Set the delegate to read the resource.

---

Generated by Doxygen
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceReadDelegate

Parameters

<table>
<thead>
<tr>
<th>readDelegate</th>
<th>ResourceReadDelegate object.</th>
</tr>
</thead>
</table>

Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

### 8.38.2.25 setRwMode()

ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setRwMode (const std::string & rwMode) [inline]

Set the rwMode field.

Parameters

<table>
<thead>
<tr>
<th>rwMode</th>
<th>The rwMode field of the ResourceInfo object.</th>
</tr>
</thead>
</table>

Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

### 8.38.2.26 setSubRscId()

ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setSubRscId (uint64_t subRscId) [inline]

Set the subRscId field.
Parameters

| subRscId | The subRscId field of the ResourceInfo object. |

Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

8.38.2.27 setTag() [1/2]

ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setTag (  
    const std::string & tag,  
    const IrisValue & value ) [inline]

Set a tag to the specified value.

Parameters

| tag | The name of the tag to set. |
| value | The value to set the tag to. |

Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

8.38.2.28 setTag() [2/2]

ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setTag (  
    const std::string & tag ) [inline]

Set the named boolean tag to true (e.g. isPc)

Parameters

| tag | The name of the tag to set. |

Returns

A reference to this ParameterBuilder object allowing calls to be chained together.
8.38.2.29  setType()

ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setType (  
    const std::string & type ) [inline]

Set the type field.

Parameters

| type | The type field of the ResourceInfo object. |

Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

8.38.2.30  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

| FUNC | A resource write delegate function. |

Returns

A reference to this ParameterBuilder object allowing calls to be chained together.
setWriteDelegate() [2/3]

\[
\text{template}\langle \text{typename } T, \text{ IrisErrorCode}(T::*)(\text{const ResourceInfo } &, \text{ const ResourceWriteValue } &) \text{ METHOD}\rangle
\]

\begin{verbatim}
ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setWriteDelegate ( 
    T* instance ) [inline]
\end{verbatim}

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

\begin{tabular}{|c|c|}
\hline
\textit{T} & A class that defines a method with the right signature to be a resource write delegate. \\
\hline
\textit{METHOD} & A resource write delegate method in class T. \\
\hline
\end{tabular}

Parameters

\begin{tabular}{|c|c|}
\hline
\textit{instance} & The instance of class T on which to call METHOD. \\
\hline
\end{tabular}

Returns

A reference to this ParameterBuilder object allowing calls to be chained together.

setWriteDelegate() [3/3]

\begin{verbatim}
ParameterBuilder& iris::IrisInstanceBuilder::ParameterBuilder::setWriteDelegate ( 
    ResourceWriteDelegate writeDelegate ) [inline]
\end{verbatim}

Generated by Doxygen
Set the delegate to write the resource.

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

See also

IrisInstanceBuilder::setDefaultResourceWriteDelegate

Parameters

| writeDelegate | ResourceWriteDelegate object. |

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

8.39 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 *resource_, IrisInstanceBuilder *instance_builder_)
  - RegisterBuilder & setAddressOffset (uint64_t addressOffset)
    Set the addressOffset field.
  - RegisterBuilder & setBitWidth (uint64_t bitWidth)
    Set the bitWidth 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)
    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<typename T, IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &)> METHOD
    RegisterBuilder & setReadDelegate (T *instance)
    Set the delegate to read the resource.
  - 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>
    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<typename T, IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &)> RegisterBuilder & setWriteDelegate (T *instance)`
  
  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 &)> 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.

8.39.1 Detailed Description

Used to set metadata on a register resource.

8.39.2 Member Function Documentation

8.39.2.1 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

<table>
<thead>
<tr>
<th>symbol</th>
<th>The symbol string to be associated with the specified value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of this symbol.</td>
</tr>
<tr>
<td>description</td>
<td>A description of this symbol.</td>
</tr>
</tbody>
</table>
Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.2    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

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the register field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsbOffset</td>
<td>The bit offset of this field inside its parent register.</td>
</tr>
<tr>
<td>bitWidth</td>
<td>The size of the field.</td>
</tr>
<tr>
<td>description</td>
<td>Description of this field.</td>
</tr>
</tbody>
</table>

Returns

A FieldBuilder object that allows the caller to set attributes for this field.

8.39.2.3    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

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the register field.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitWidth</td>
<td>The size of the field.</td>
</tr>
<tr>
<td>description</td>
<td>Description of this field.</td>
</tr>
</tbody>
</table>
Returns

A FieldBuilder object that allows the caller to set attributes for this field.

8.39.2.4 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

<table>
<thead>
<tr>
<th>value</th>
<th>The string value of this symbol. This is also used as the symbols string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A description of this symbol.</td>
</tr>
</tbody>
</table>

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.5 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.
8.39.2.6 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.

8.39.2.7 setAddressOffset()

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

Set the addressOffset field.

Parameters

| addressOffset | The addressOffset field of the RegisterInfo object. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.8 setBitWidth()

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

Set the bitWidth field.

Parameters

| bitWidth | The bitWidth field of the ResourceInfo object. |
Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.9 setCanonicalRn()

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setCanonicalRn (uint64_t canonicalRn_) [inline]

Set the canonicalRn field.

Note: Use setCanonicalRnElfDwarf() when using the "ElfDwarf" scheme.

Parameters

| canonicalRn | The canonicalRn field of the RegisterInfo object. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.10 setCanonicalRnElfDwarf()

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setCanonicalRnElfDwarf (uint16_t architecture, uint16_t dwarfRegNum ) [inline]

Set the canonicalRn field for "ElfDwarf" scheme.

Parameters

| architecture | ELF EM_+ constant for architecture. |
| dwarfRegNum  | DWARF register number for architecture. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.
8.39.2.11  setCname()

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setCname (const std::string & cname) [inline]

Set the \texttt{cname} field.

Parameters

\begin{tabular}{|l|p{0.75\textwidth}|}
\hline
\textit{cname} & The \texttt{cname} field of the ResourceInfo object. \\
\hline
\end{tabular}

Returns

A reference to this \texttt{RegisterBuilder} object allowing calls to be chained together.

8.39.2.12  setDescr()

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setDescr (const std::string & description) [inline]

Set the \texttt{description} field.

Parameters

\begin{tabular}{|l|p{0.75\textwidth}|}
\hline
\textit{description} & The \texttt{description} field of the ResourceInfo object. \\
\hline
\end{tabular}

Returns

A reference to this \texttt{RegisterBuilder} object allowing calls to be chained together.

8.39.2.13  setFormat()

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

Set the \texttt{format} field.

Parameters

\begin{tabular}{|l|p{0.75\textwidth}|}
\hline
\textit{format} & The \texttt{format} field of the ResourceInfo object. \\
\hline
\end{tabular}
Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.14 setLsbOffset()

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setLsbOffset (uint64_t lsbOffset) [inline]

Set the lsbOffset field.

Parameters

| lsbOffset | The lsbOffset field of the RegisterInfo object. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.15 setName()

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

Set the name field.

Parameters

| name | The name field of the ResourceInfo object. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.16 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

<table>
<thead>
<tr>
<th>parent</th>
<th>The rsclid of the parent register.</th>
</tr>
</thead>
</table>

Returns

A reference to this `RegisterBuilder` object allowing calls to be chained together.

8.39.2.17 `setReadDelegate()` [1/3]

```cpp
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

<table>
<thead>
<tr>
<th>FUNC</th>
<th>A resource read delegate function.</th>
</tr>
</thead>
</table>

Returns

A reference to this `RegisterBuilder` object allowing calls to be chained together.
8.39.2.18 setReadDelegate() [2/3]

```cpp
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

| readDelegate | ResourceReadDelegate object. |

Returns

A reference to this `RegisterBuilder` object allowing calls to be chained together.

8.39.2.19 setReadDelegate() [3/3]

```cpp
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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>A class that defines a method with the right signature to be a resource read delegate.</td>
</tr>
<tr>
<td>METHOD</td>
<td>A resource read delegate method in class T.</td>
</tr>
</tbody>
</table>

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>The instance of class T on which to call METHOD.</td>
</tr>
</tbody>
</table>

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.20 setResetData() [1/2]

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setResetData ( uint64_t value ) [inline]

Set the resetData field to a value \(_{\leq} 64\) 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>resetData value of the register.</td>
</tr>
</tbody>
</table>

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.21 setResetData() [2/2]

template<typename T >
RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setResetData ( std::initializer_list<T> && t ) [inline]

Generated by Doxygen
Set the `resetData` field for wide registers.

This function accepts a braced initializer-list and is otherwise identical to

```cpp
setResetDataFromContainer().
```

Each element will be promoted/narrowed to `uint64_t`.

**Parameters**

| t | Braced initializer-list. |

**Returns**

A reference to this `RegisterBuilder` object allowing calls to be chained together.

### 8.39.2.22 setResetDataFromContainer()

```cpp
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

| container | Container containing the value in 64-bit chunks. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.23 setResetString()

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setResetString (const std::string & resetString) [inline]

Set the resetString field.

Set the reset value for string registers.

Parameters

| resetString | The resetString field of the RegisterInfo object. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.24 setRwMode()

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setRwMode (const std::string & rwMode) [inline]

Set the rwMode field.

Parameters

| rwMode | The rwMode field of the ResourceInfo object. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.
8.39.2.25 setSubRscId()

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setSubRscId (  
    uint64_t subRscId ) [inline]

Set the subRscId field.

Parameters

| subRscId | The subRscId field of the ResourceInfo object. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.26 setTag() [1/2]

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setTag (  
    const std::string & tag ) [inline]

Set the named boolean tag to true (e.g. isPc)

Parameters

| tag | The name of the tag to set. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.27 setTag() [2/2]

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setTag (  
    const std::string & tag,  
    const IrisValue & value ) [inline]

Set a tag to the specified value.

Parameters

| tag | The name of the tag to set. |
| value | The value to set the tag to. |
Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.28 setType()

RegisterBuilder& iris::IrisInstanceBuilder::RegisterBuilder::setType (
    const std::string & type ) [inline]

Set the type field.

Parameters

| type | The type field of the ResourceInfo object. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.29 setWriteDelegate() [1/3]

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
A class that defines a method with the right signature to be a resource write delegate.

A resource write delegate method in class T.

The instance of class T on which to call METHOD.

A reference to this RegisterBuilder object allowing calls to be chained together.

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

A resource write delegate function.

A reference to this RegisterBuilder object allowing calls to be chained together.
8.39.2.31 setWriteDelegate() [3/3]

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

| writeDelegate | ResourceWriteDelegate object. |

Returns

A reference to this RegisterBuilder object allowing calls to be chained together.

8.39.2.32 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

| t | Braced initializer-list. |

Returns

A reference to this `RegisterBuilder` object allowing calls to be chained together.

8.39.2.33 `setWriteMask()` [2/2]

```cpp
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

| value | writeMask value of the register. |

Returns

A reference to this `RegisterBuilder` object allowing calls to be chained together.

8.39.2.34 `setWriteMaskFromContainer()`

```cpp
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

| container | 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

8.40 iris::IrisInstanceResource::ResourceInfoAndAccess Struct Reference

Entry in 'resourceInfos'.

#include <IrisInstanceResource.h>

Public Attributes

- ResourceReadDelegate readDelegate
- ResourceInfo resourceInfo
- ResourceWriteDelegate writeDelegate

8.40.1 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
8.41 iris::ResourceWriteValue Struct Reference

#include <IrisInstanceResource.h>

Public Attributes

- const uint64_t * data {}
- const std::string * str {}

Non-null for non-string resources.

8.41.1 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

8.42 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_CA←LL_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 uint8_t *data, size_t size)
- bool writeData (uint64_t fDes, const std::vector< uint8_t > &data)

8.42.1 Detailed Description

semihosting_apis IrisInstanceBuilder semihosting APIs

Manage semihosting functionality
8.42.2 Member Function Documentation

8.42.2.1 readData()

```cpp
std::vector<
    uint8_t>
iris::IrisInstanceBuilder::SemihostingManager::readData (  
    uint64_t fDes,  
    size_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 method blocks. If max_size is not zero, then at most max_size bytes will be returned.

Parameters

- **fDes**: File descriptor to read from. Usually semihost::STDIN.
- **max_size**: The maximum amount of bytes to read or zero for no limit.
- **flags**: A bitwise OR of Semihosting data request flag constants.

Returns

A vector of data that was read.

8.42.2.2 semihostedCall()

```cpp
std::pair<
    bool, uint64_t>
iris::IrisInstanceBuilder::SemihostingManager::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

- **operation**: 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.
- **parameter**: 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

8.43 iris::IrisInstanceMemory::SpaceInfoAndAccess Struct Reference

Entry in 'spaceInfos'.

#include <IrisInstanceMemory.h>

Public Attributes

- MemoryReadDelegate readDelegate
- MemoryGetSidebandInfoDelegate sidebandDelegate
- MemorySpaceInfo spaceInfo
- MemoryWriteDelegate writeDelegate

8.43.1 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

8.44 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.
- **TableBuilder & setReadDelegate** (TableReadDelegate delegate)
  Set the delegate to read the table.
  
  ```cpp
  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 & setWriteDelegate** (TableWriteDelegate delegate)
  Set the delegate to write to the table.
  
  ```cpp
  template<typename T, IrisErrorCode(T::*)(const TableInfo &, const TableRecords &, TableWriteResult &)> METHOD
  TableBuilder & setWriteDelegate (T *instance)
  ```
  Set the delegate to write to the table.

8.44.1 Detailed Description

Used to set metadata for a table.

8.44.2 Member Function Documentation
### 8.44.2.1 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**

| name | The name of the new column. |

**Returns**

A TableColumnBuilder object that can be used to add metadata for the new column.

### 8.44.2.2 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**

| columnInfo | A preconstructed TableColumnInfo object for the new column. |

**Returns**

A reference to this TableBuilder allowing calls to be chained together.

### 8.44.2.3 setDescription()

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

Set the description of the table.

Call this to set the description of the table.

See also

setMeta

**Parameters**

| description | The description of the table. |

**Returns**

A reference to this TableBuilder allowing calls to be chained together.
Set the `description` field.
Parameters

| description | The description field of the TableInfo object. |

Returns

A reference to this `TableBuilder` allowing calls to be chained together.

### 8.44.2.4 setFormatLong()

```
TableBuilder& iris::IrisInstanceBuilder::TableBuilder::setFormatLong (const std::string & format) [inline]
```

Set the `formatLong` field.

Parameters

| format | The `formatLong` field of the TableInfo object. |

Returns

A reference to this `TableBuilder` allowing calls to be chained together.

### 8.44.2.5 setFormatShort()

```
TableBuilder& iris::IrisInstanceBuilder::TableBuilder::setFormatShort (const std::string & format) [inline]
```

Set the `formatShort` field.

Parameters

| format | The `formatShort` field of the TableInfo object. |

Returns

A reference to this `TableBuilder` allowing calls to be chained together.

### 8.44.2.6 setIndexFormatHint()

```
TableBuilder& iris::IrisInstanceBuilder::TableBuilder::setIndexFormatHint (const std::string & hint) [inline]
```

Generated by Doxygen
Set the `indexFormatHint` field.
Parameters

| hint | The indexFormatHint field of the TableInfo object. |

Returns

A reference to this TableBuilder allowing calls to be chained together.

8.44.2.7 setMaxIndex()

```cpp
TableBuilder& iris::IrisInstanceBuilder::TableBuilder::setMaxIndex ( 
    uint64_t maxIndex ) [inline]
```

Set the maxIndex field.

Parameters

| maxIndex | The maxIndex field of the TableInfo object. |

Returns

A reference to this TableBuilder allowing calls to be chained together.

8.44.2.8 setMinIndex()

```cpp
TableBuilder& iris::IrisInstanceBuilder::TableBuilder::setMinIndex ( 
    uint64_t minIndex ) [inline]
```

Set the minIndex field.

Parameters

| minIndex | The minIndex field of the TableInfo object. |

Returns

A reference to this TableBuilder allowing calls to be chained together.

8.44.2.9 setName()

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

Generated by Doxygen
Set the `name` field.
Parameters

| name          | The name field of the TableInfo object. |

Returns

A reference to this TableBuilder allowing calls to be chained together.

8.44.2.10 setReadDelegate() [1/3]

```cpp
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

| delegate     | TableReadDelegate object. |

Returns

A reference to this TableBuilder object allowing calls to be chained together.

8.44.2.11 setReadDelegate() [2/3]

```cpp
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

<table>
<thead>
<tr>
<th>T</th>
<th>A class that defines a method with the right signature to be a table read delegate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A table read delegate method in class T.</td>
</tr>
</tbody>
</table>

Parameters

| instance | The instance of class T on which to call METHOD. |

Returns

A reference to this TableBuilder object allowing calls to be chained together.

8.44.2.12 setReadDelegate() [3/3]

```cpp
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

| FUNC | A table read delegate function. |

Returns

A reference to this TableBuilder object allowing calls to be chained together.

8.44.2.13 setWriteDelegate() [1/3]

```cpp
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

**delegate**
A `TableWriteDelegate` object.

### Returns
A reference to this `TableBuilder` object allowing calls to be chained together.

---

### 8.44.2.14 `setWriteDelegate()` [2/3]

```cpp
template<
    typename T,
    IrisErrorCode(T::*)(const TableInfo &, const TableRecords &, TableWriteResult &) 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

<table>
<thead>
<tr>
<th>T</th>
<th>A class that defines a method with the right signature to be a table write delegate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>A table write delegate method in class T.</td>
</tr>
</tbody>
</table>

### Parameters

**instance**
The instance of class T on which to call METHOD.

### Returns
A reference to this `TableBuilder` object allowing calls to be chained together.

---

### 8.44.2.15 `setWriteDelegate()` [3/3]

```cpp
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

| FUNC | A table write delegate function. |

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

### 8.45 iris::IrisInstanceBuilder::TableColumnBuilder Class Reference

Used to set metadata for a table column.

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

#### Public Member Functions

- **addColumn** (const std::string &name)
  Add another new column.

- **addColumnInfo** (const TableColumnInfo &columnInfo)
  Add another column with a preconstructed TableColumnInfo.

- **endColumn** ()
  Stop building this column and go back to the parent table.

- **setBitWidth** (uint64_t bitWidth)
  Set the bitWidth field.

- **setDescription** (const std::string &description)
  Set the description field.

- **setFormat** (const std::string &format)
  Set the format field.

- **setFormatLong** (const std::string &format)
  Set the formatLong field.

- **setFormatShort** (const std::string &format)
  Set the formatShort field.

- **setName** (const std::string &name)
  Set the name field.

- **setRwMode** (const std::string &rwMode)
  Set the rwMode field.

- **setType** (const std::string &type)
  Set the type field.

**TableColumnBuilder (TableBuilder &parent_, TableColumnInfo &info_)**

#### 8.45.1 Detailed Description

Used to set metadata for a table column.
8.45.2 Member Function Documentation

8.45.2.1 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

| name  | The name of the new column. |

Returns

A TableColumnBuilder object that can be used to add metadata for the new column.

8.45.2.2 addColumnInfo()

TableBuilder& iris::IrisInstanceBuilder::TableColumnBuilder::addColumnInfo (const TableColumnInfo & columnInfo) [inline]

Add another column with a preconstructed TableColumnInfo.

See also
TableBuilder::addColumn
addColumn

Parameters

| columnInfo | A preconstructed TableColumnInfo object for the new column. |

Returns

A reference to the parent TableBuilder for this table.
8.45.2.3  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.

8.45.2.4  setBitWidth()

*TableColumnBuilder* iris::IrisInstanceBuilder::TableColumnBuilder::setBitWidth (  
  *uint64_t* bitWidth ) [inline]

Set the *bitWidth* field.

Parameters

| *bitWidth* | The *bitWidth* field of the *TableColumnInfo* object. |

Returns

  A *TableColumnBuilder* object that can be used to add metadata for the new column.

8.45.2.5  setDescription()

*TableColumnBuilder* iris::IrisInstanceBuilder::TableColumnBuilder::setDescription (  
  *const std::string &* description ) [inline]

Set the *description* field.

Parameters

| *description* | The *description* field of the *TableColumnInfo* object. |

Returns

  A *TableColumnBuilder* object that can be used to add metadata for the new column.
8.45.2.6 setFormat()

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

Set the format field.

Parameters

| format | The format field of the TableColumnInfo object. |

Returns

A TableColumnBuilder object that can be used to add metadata for the new column.

8.45.2.7 setFormatLong()

```cpp
TableColumnBuilder& iris::IrisInstanceBuilder::TableColumnBuilder::setFormatLong ( const std::string & format ) [inline]
```

Set the formatLong field.

Parameters

| format | The formatLong field of the TableColumnInfo object. |

Returns

A TableColumnBuilder object that can be used to add metadata for the new column.

8.45.2.8 setFormatShort()

```cpp
TableColumnBuilder& iris::IrisInstanceBuilder::TableColumnBuilder::setFormatShort ( const std::string & format ) [inline]
```

Set the formatShort field.

Parameters

| format | The formatShort field of the TableColumnInfo object. |
Returns

A TableColumnBuilder object that can be used to add metadata for the new column.

### 8.45.2.9 setName()

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

Set the name field.

**Parameters**

- **name** The name field of the TableColumnInfo object.

Returns

A TableColumnBuilder object that can be used to add metadata for the new column.

### 8.45.2.10 setRwMode()

```cpp
TableColumnBuilder iris::IrisInstanceBuilder::TableColumnBuilder::setRwMode (const std::string & rwMode) [inline]
```

Set the rwMode field.

**Parameters**

- **rwMode** The rwMode field of the TableColumnInfo object.

Returns

A TableColumnBuilder object that can be used to add metadata for the new column.

### 8.45.2.11 setType()

```cpp
TableColumnBuilder iris::IrisInstanceBuilder::TableColumnBuilder::setType (const std::string & type) [inline]
```

Set the type field.

Generated by Doxygen
Parameters

| type | 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

8.46   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.

8.46.1   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 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 <string>

Classes

• class iris::IrisCConnection
  
  Provide an IrisConnectionInterface which loads an IrisC library.

9.1.1 Detailed Description

IrisConnectionInterface implementation based on IrisC.

Copyright

Copyright (C) 2017-2019 Arm Limited. All rights reserved.

9.2 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_irq = 0x2800000081, ARM_SPSR_irq = 0x2800000082, ARM_SPSR_abt = 0x2800000083,
  ARM_SPSR_irq = 0x2800000084, ARM_SPSR_svc = 0x2800000085, ARM_R8_irq = 0x2800000086, ARM_R9_irq = 0x2800000087,
  ARM_R10_irq = 0x2800000088, ARM_R11_irq = 0x2800000089, ARM_R12_irq = 0x280000008a, ARM_R13_irq = 0x280000008b,
  ARM_R14_irq = 0x280000008c, ARM_R15_irq = 0x280000008d, ARM_R16_irq = 0x280000008e, ARM_R17_irq = 0x280000008f,
  ARM_R18_irq = 0x2800000090, ARM_R19_irq = 0x2800000091, ARM_R20_irq = 0x2800000092, ARM_R21_irq = 0x2800000093,
  ARM_R22_irq = 0x2800000094, ARM_R23_irq = 0x2800000095, ARM_R24_irq = 0x2800000096, ARM_R25_irq = 0x2800000097,
  ARM_R26_irq = 0x2800000098, ARM_R27_irq = 0x2800000099, ARM_R28_irq = 0x280000009a, ARM_R29_irq = 0x280000009b,
  ARM_R30_irq = 0x280000009c, ARM_R31_irq = 0x280000009d, ARM_R32_irq = 0x280000009e, ARM_R33_irq = 0x280000009f,
  ARM_R34_irq = 0x28000000a0, ARM_R35_irq = 0x28000000a1, ARM_R36_irq = 0x28000000a2, ARM_R37_irq = 0x28000000a3, ARM_R38_irq = 0x28000000a4,
  ARM_R39_irq = 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_X31 = 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,
9.3 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.

9.3.1 Detailed Description

A utility class for emitting Iris events.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.
9.4 IrisGlobalInstance.h File Reference

Central instance which lives in the simulation engine and distributes all Iris messages.

```cpp
#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

9.4.1 Detailed Description

Central instance which lives in the simulation engine and distributes all Iris messages.

Date

Copyright ARM Limited 2014-2019 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.

9.5 IrisInstance.h File Reference

Boilerplate code for an Iris instance, including clients and components.

```cpp
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisCppAdapter.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisFunctionDecoder.h"
#include "iris/IrisInstanceEvent.h"
#include <cassert>
#include <mutex>
#include "iris/IrisInstanceBuilder.h"
```
9.5 IrisInstance.h File Reference 297

Classes

• class iris::IrisInstance

Macros

• #define irisRegisterEventCallback(instancePtr, instanceType, functionName, description) registerEventCallback\<instanceType, &instanceType::impl_##functionName\>(instancePtr, functionName, description, \#instanceType)

  Register an event callback function using an EventCallbackDelegate.

• #define irisRegisterFunction(instancePtr, instanceType, functionName, functionInfoJson) registerFunction\(instancePtr, \#functionName, \&instanceType::impl_##functionName, functionInfoJson, \#instanceType\)

  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\<uint64_t, const IrisValueMap &, uint64_t, uint64_t, bool, std::string & \> iris::EventCallbackDelegate

  Event callback delegate.

9.5.1 Detailed Description

Boilerplate code for an Iris instance, including clients and components.

Copyright

Copyright (C) 2015 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.

9.5.2 Typedef Documentation
9.5.2.1 EventCallbackDelegate

typedef IrisDelegate<
    uint64_t, const IrisValueMap&, uint64_t, uint64_t, bool, std::string&>
iris::EventCallbackDelegate

Event callback delegate.

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"
);

9.6 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

• 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.
9.6.1 Detailed Description

Breakpoint add-on to IrisInstance.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

The IrisInstanceBreakpoint class:

- Implements all breakpoint-related Iris functions.
- Maintains and provides breakpoint information, for example type, address, and rsclId.
- Converts between Iris breakpoint functions (breakpoint∗()) and various C++ access functions.

9.6.2 Typedef Documentation

9.6.2.1 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.

9.6.2.2 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.
9.7 IrisInstanceBuilder.h File Reference

A high level interface to build up functionality on an IrisInstance.

```c
#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/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.

9.7.1 Detailed Description

A high level interface to build up functionality on an IrisInstance.

Copyright

Copyright (C) 2016-2019 Arm Limited. All rights reserved.
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.

9.8.1 Detailed Description

IrisInstance add-on to implement debuggableState functions.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.8.2 Typedef Documentation

9.8.2.1 DebuggableStateGetAcknowledgeDelegate

typedef IrisDelegate<bool &> iris::DebuggableStateGetAcknowledgeDelegate

Interface to stop the simulation time progress.

IrisErrorCode getAcknowledge(bool &acknowledge_out);
9.8.2.2 DebuggableStateSetRequestDelegate

typedef IrisDelegate<bool> iris::DebuggableStateSetRequestDelegate

Delegate to set the debuggable-state-request flag.

IrisErrorCode setRequest(bool request);

9.9 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 <cstdio>

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.

9.9.1 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.
9.10 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.

**Typedefs**

- **typedef IrisDelegate< EventStream *, const EventSourceInfo &, const std::vector<std::string> & > iris::EventStreamCreateDelegate**
  Delegate to create an EventStream.

9.10.1 Detailed Description

Event add-on to IrisInstance.

**Copyright**

Copyright (C) 2016 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.
9.10.2 Typedef Documentation

9.10.2.1 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.

9.11 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.

9.11.1 Detailed Description

A helper class to build instantiation parameter metadata.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.
9.12 IrisInstanceImage.h File Reference

Image-loading add-on to IrisInstance and image-loading callback add-on to the caller.

```cpp
#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< uint64_t > &, uint64_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.

9.12.1 Detailed Description

Image-loading add-on to IrisInstance and image-loading callback add-on to the caller.

Copyright

Copyright (C) 2016 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.

9.12.2 Typedef Documentation

Generated by Doxygen
9.12.2.1 ImageLoadDataDelegate

typedef IrisDelegate<
const std::vector<uint64_t>&, uint64_t> iris::ImageLoadDataDelegate

Delegate to load an image from the given data.
Bytes are stored in little-endian format.

IrisErrorCode loadImage(const std::vector<uint64_t> &data, uint64_t dataSize)

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.

9.12.2.2 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.

9.13 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"
9.13 IrisInstanceMemory.h File Reference

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, MemoryReadResult &> iris::MemoryReadDelegate
  
  Delegate to read memory data.

- typedef IrisDelegate<const MemorySpaceInfo &, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &> iris::MemoryWriteDelegate
  
  Delegate to write memory data.

9.13.1 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.

9.13.2 Typedef Documentation
9.13.2.1 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.

9.13.2.2 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.

9.13.2.3 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.
### 9.14.2.4 MemoryWriteDelegate

```cpp
typedef IrisDelegate<
    const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &>
iris::MemoryWriteDelegate
```

Delegate to write memory data.

```cpp
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.

### 9.14 IrisInstancePerInstanceExecution.h File Reference

Per-instance execution control add-on to IrisInstance.

```cpp
#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.

### 9.14.1 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.
9.14.2 Typedef Documentation

9.14.2.1 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.

9.14.2.2 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.

9.15 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

- void iris::resourceWriteBitField (T &parentValue, uint64_t fieldValue, const ResourceInfo &resourceInfo)
- template<class T>
  void iris::resourceWriteBitField (T &parentValue, uint64_t fieldValue, const ResourceInfo &resourceInfo)

9.15.1 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.

9.15.2 Typedef Documentation

9.15.2.1 ResourceReadDelegate

typedef IrisDelegate<const ResourceInfo&, ResourceReadResult&> iris::ResourceReadDelegate
Delegate to read resources.

IrisErrorCode read(const ResourceInfo &resourceInfo, ResourceReadResult &result)

resourceInfo.rscld is guaranteed to be valid.

Typical implementations inspect the rscld, 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.
  - 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.
9.15.2.2 ResourceWriteDelegate

typedef IrisDelegate<const ResourceInfo&, const ResourceWriteValue&> iris::ResourceWriteDelegate

Delegate to write resources.

IrisErrorCode write(const ResourceInfo &resourceInfo, const ResourceWriteValue &value)

resourceInfo.rscId is guaranteed to be valid.

Typical implementations inspect the rscId, 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.

9.15.3 Function Documentation

9.15.3.1 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.

9.15.3.2 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.

9.16 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

Variables

- static const uint64_t iris::semihost::BLOCK = (0 << 1)
  
  Block until enough data has been provided based on the specified size hint.

- static const uint64_t iris::semihost::COOKED = (0 << 0)
  
  Request that the client sends data when the user presses the return key.

- static const uint64_t iris::semihost::DEFAULT = COOKED | BLOCK | EMIT_EVENT
  
  The default flags used if none are specified.

- static const uint64_t iris::semihost::EMIT_EVENT = (0 << 2)
  
  If the buffer is empty, emit an event to request input.

- static const uint64_t iris::semihost::NO_EVENT = (1 << 2)
  
  Do not request input. Return only the data currently buffered.

- static const uint64_t iris::semihost::NONBLOCK = (1 << 1)
  
  Do not block if sufficient data is not available.

- static const uint64_t iris::semihost::RAW = (1 << 0)
  
  Request that the client sends data to the model as soon as it is available.

- static const uint64_t iris::semihost::STDERR = 2
  
  File descriptor used with writeData() for semihosted standard error output.

- static const uint64_t iris::semihost::STDIN = 0
  
  Usual file descriptor used for semihosted input with readData().

- static const uint64_t iris::semihost::STDOUT = 1
  
  File descriptor used with writeData() for semihosted standard output.

9.16.1 Detailed Description

IrisInstance add-on to implement semihosting functionality.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.17 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* {
  
  
  List of IRIS_SIMULATION_PHASE events.

Variables

- static const size_t *iris::IrisSimulationPhase_total* = 20

9.17.1 Detailed Description

*IrisInstance* add-on to implement simulation_\* functions.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.
9.17.2 Typedef Documentation

9.17.2.1 SimulationGetParameterInfoDelegate

typedef IrisDelegate<std::vector<ResourceInfo>>& iris::SimulationGetParameterInfoDelegate

Delegate to get a list of parameter information.

IrisErrorCode getInstantiationParameterInfo(std::vector<ResourceInfo> &parameters_out)

9.17.2.2 SimulationInstantiateDelegate

typedef IrisDelegate<InstantiationResult&> iris::SimulationInstantiateDelegate

Delegate to instantiate the simulation.

IrisErrorCode instantiate(InstantiationResult &result_out)

9.17.2.3 SimulationRequestShutdownDelegate

typedef IrisDelegate iris::SimulationRequestShutdownDelegate

Delegate to request that the simulation be shut down.

IrisErrorCode requestShutdown()

9.17.2.4 SimulationResetDelegate

typedef IrisDelegate<const IrisSimulationResetContext&> iris::SimulationResetDelegate

Delegate to reset the simulation.

IrisErrorCode reset(const IrisSimulationResetContext &)
9.17.2.5 SimulationSetParameterValueDelegate

typedef IrisDelegate<const InstantiationParameterValue&> iris::SimulationSetParameterValueDelegate

Delegate to set the value of an instantiation parameter.

IrisErrorCode setInstantiationParameterValue(const InstantiationParameterValue &value)

9.18 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>

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_UNKNOWN, iris::TIME_EVENT_STOP, iris::TIME_EVENT_BREAKPOINT,
  iris::TIME_EVENT_TRACE_COUNTER_OVERFLOW }
  The reasons why the simulation time stopped.

9.18.1 Detailed Description

IrisInstance add-on to implement simulationTime functions.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.
9.18.2 Typedef Documentation

9.18.2.1 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);

9.18.2.2 SimulationTimeRunDelegate

typedef IrisDelegate iris::SimulationTimeRunDelegate

Delegate to resume the simulation time progress.

IrisErrorCode run();

9.18.2.3 SimulationTimeStopDelegate

typedef IrisDelegate iris::SimulationTimeStopDelegate

Delegate to stop the simulation time progress.

IrisErrorCode stop();

9.18.3 Enumeration Type Documentation

9.18.3.1 TIME_EVENT_REASON

enum iris::TIME_EVENT_REASON

The reasons why the simulation time stopped.
### Enumerator

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_EVENT_UNKNOWN</td>
<td>Simulation stopped for a different reason.</td>
</tr>
<tr>
<td>TIME_EVENT_STOP</td>
<td>simulationTime_stop() was called.</td>
</tr>
<tr>
<td>TIME_EVENT_BREAKPOINT</td>
<td>Breakpoint was hit.</td>
</tr>
<tr>
<td>TIME_EVENT_TRACE_COUNTER_OVERFLOW</td>
<td>CounterMode.overflowStopSim.</td>
</tr>
</tbody>
</table>

## 9.19 IrisInstanceStep.h File Reference

Stepping-related add-on to an IrisInstance.

```cpp
#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.

## 9.19.1 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.

## 9.19.2 Typedef Documentation
9.19.2.1 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.

9.19.2.2 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.

9.19.2.3 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.

9.20 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'.

Generated by Doxygen
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.*

9.20.1 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.

9.20.2 Typedef Documentation

9.20.2.1 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.

9.20.2.2 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.
9.21 IrisInstantiationContext.h File Reference

Helper class used to instantiate Iris instances from generic factories.

```cpp
#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.

9.21.1 Detailed Description

Helper class used to instantiate Iris instances from generic factories.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.22 IrisParameterBuilder.h File Reference

Helper class to construct instantiation parameters.

```cpp
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisObjects.h"
#include <string>
#include <vector>
```

### Classes

- **class iris::IrisParameterBuilder**

  Helper class to construct instantiation parameters.

9.22.1 Detailed Description

Helper class to construct instantiation parameters.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.
9.23 IrisPluginFactory.h File Reference

A generic plug-in factory for instantiating plug-in instances.

```cpp
#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::IrisPluginFactory< PLUGIN_INSTANCE >**
- **class iris::IrisPluginFactoryBuilder**
  
  Set metadata for instantiating a plug-in instance.

### Macros
- **#define IRIS_PLUGIN_FACTORY(plugin_instance)**

  Use this macro to create an Iris plug-in entry point.

9.23.1 Detailed Description

A generic plug-in factory for instantiating plug-in instances.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.23.2 Macro Definition Documentation

9.23.2.1 IRIS_PLUGIN_FACTORY

```cpp
#define IRIS_PLUGIN_FACTORY( 
  plugin_instance ) 
```

**Value:**

```cpp
extern "C" IRIS_EXPORT int64_t irishInitPlugin(IrisC_Functions* functions)
{
  return ::iris::IrisPluginFactory<plugin_instance>::initPlugin(functions, #plugin_instance);
}
```

Use this macro to create an Iris plug-in entry point.
Parameters

| plugin_instance | Objects of this type are instantiated for each plug-in instance created. |

9.24 IrisRegisterEventEmitter.h File Reference

Utility classes for emitting register read and register update events.

```cpp
#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.

9.24.1 Detailed Description

Utility classes for emitting register read and register update events.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

9.25 IrisTcpClient.h File Reference

TCP client used by Iris clients which connect via TCP.

```cpp
#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/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::IrisTcpClient

Functions

- NAMESPACE_IRIS_INTERNAL_START (service) class IrisServiceTcpServer

9.25.1 Detailed Description

TCP client used by Iris clients which connect via TCP.

Date

Copyright ARM Limited 2015-2019 All Rights Reserved.

TCP client used by Iris clients which connect via TCP.

The TCP client requires an IrisInstance (or similar) to send or receive any Iris calls.